# Especificação em Estelle Tempo-Real de um Sistema de Transmissão CBR

Gardel Moreira Delfino
gardel@gta.ufrj.br

José Valentim dos Santos Filho
valentim@gta.ufrj.br

Rodrigo de Carvalho Roman
roman@gta.ufrj.br

Jorge Lopes de Souza Leão
leao@gta.ufrj.br

Aloysio de C. P. Pedroza
aloysio@gta.ufrj.br

Grupo de Teleinformática e Automação – GTA
Universidade Federal do Rio de Janeiro
COPPE/EE - Programa de Engenharia Elétrica
C. P. 68504 - CEP 21945-970 - Rio de Janeiro - RJ - Brasil

**Resumo**

A Técnica de Descrição Formal Estelle mostrou-se muito eficiente na especificação de protocolos tradicionais. Entretanto, sua expressividade torna-se limitada quando pensamos em descrever sistemas distribuídos com requisitos temporais, tais como as aplicações multimídia. Para este tipo de sistema, foi criada uma extensão à Linguagem Estelle, que contempla alguns requisitos de Qualidade de Serviço (QoS). No presente trabalho, utilizamos um subconjunto desta extensão para especificar um exemplo típico de aplicação com restrições temporais. Nesta especificação, mostramos um sistema simples de transmissão CBR (Constant Bit Rate), onde podemos visualizar características comuns a sistemas multimídia sendo descritas, como a vazão desejada, por exemplo.

**Abstract**

The Formal Description Technicque Estelle has proven its efficience in specifying traditional protocols. Nevertheless, its expressiveness becomes limited when we think about describing distributed systems with timing requirements, like multimedia applications. For this type of system, it has been created an extension to the Estelle language, that supports some Quality of Service (QoS) Requirements. In the present work, we use a subset of this extension to specify a typical example of applications with timing constraints. In this specification, we show a simple CBR (Constant Bit Rate) transmission system, where we can visualize the description of characteristics commonly seen in multimedia systems, like the desired throughput, for example.

Palavras-chave: Protocolos de Comunicação,Técnicas de Descrição Formal, Estelle, Requisitos de QoS, Multimídia.

# 1 Introdução

O crescimento significativo de aplicações distribuídas que exigem requisitos de QoS foi a principal motivação para o desenvolvimento de uma extensão à linguagem Estelle Padrão [1], denominada *Real-Time Estelle* [2]. Esta extensão se fez necessária, devido ao fato de que Estelle não é apropriada para especificar restrições temporais, já que a cláusula *Delay* de Estelle se refere apenas aos tempos de disparo associados a uma dada transição, considerada de maneira isolada. Não podemos, portanto, desta maneira especificar restrições temporais fim-a-fim [3].

No presente trabalho utilizamos um subconjunto desta extensão à TDF (Técnica de Descrição Formal) Estelle, para especificar uma aplicação semelhante a muitos possíveis sistemas onde restrições temporais se fazem tão ou mais importantes que os aspectos funcionais destes sistemas. Neste exemplo, descrevemos um Sistema de Transmissão CBR, com apenas um emissor e um receptor, e mostramos como pode ser usada a linguagem Estelle Tempo-Real, para expressar tais restrições.

Neste relatório descreveremos, inicialmente os aspectos relevantes da linguagem Estelle Tempo-Real. Em seguida, será detalhado o processo de especificação deste protocolo simples, enfatizando como foram descritas as metas de QoS desejadas para este sistema.

# 2 A Técnica de Descrição Formal Estelle Tempo-Real

A extensão utilizada para tempo real é um subconjunto da extensão *Real-Time Estelle*, que teve sua sintaxe construída visando manter a mesma estrutura da Estelle Padrão. A descrição das restrições temporais é realizada por fórmulas de lógica temporal [4], o que levou a incorporação de novas palavras-chave à linguagem. A escolha dos novos operadores e palavras-chave foi feita de modo a permitir especificar os requisitos de QoS mais comuns, tais como: Taxa de transmissão, Atraso, *Jitter*(variações de atraso) e Taxa de Erro, conforme exemplos verificados na literatura [5]. Este conjunto de acréscimos à TDF Estelle é mostrado na tabela 1.

Neste formalismo específico, a descrição funcional do sistema, ou seja, como ele progride em relação às interações possíveis, utiliza a mesma estrutura de uma especificação em Estelle de protocolos tradicionais, sem restrições temporais. Entretanto, no domínio dos requisitos de QoS desejados é que se estabelece a verdadeira extensão, que utiliza uma seção denominada *time constraints*, onde se descrevem as restrições específicas daquele subsistema em questão. Desta maneira, separa-se a análise de correção do comportamento do sistema da análise de seu desempenho, enquanto especificação de sistema com requisitos temporais relativamente rígidos.

A partir da tabela 1 podemos descrever a interpretação dos operadores lógicos e temporais citados, bem como das palavras-chave e descritores de estado. A palavra-chave *time constraints* indica o início de um bloco de declaração de restrições temporais referentes ao corpo de módulo onde se encontra. Restrições fim-a-fim devem ser descritas em um corpo de módulo pai que engloba as entidades comunicantes. Com isto, preservam-se as regras de escopo presentes na TDF Estelle.

Tabela 1: Acréscimos à TDF Estelle Padrão

| Tipo de Identificador | Identificadores |
|---|---|
| Palavras-Chave | *time constraints* |
| Operadores Lógicos | *and*<br>*or*<br>*otherwise*<br>*eventually*<br>*henceforth*<br>*leadsto*<br>*forbids*<br>*at* |
| Operadores Temporais | *now* |
| Descritores Simples de Estado | *sending of*<br>*receiving of* |

O operador temporal *now* tem funcionamento semelhante a uma função que busca a hora atual de um sistema. Ou seja, ele devolve o instante de tempo no qual o sistema se encontra, seja ele simulado ou encontrado nas funções de hora do sistema alvo em questão.

Os descritores simples de estado utilizados são uma maneira de se verificar o conteúdo dos pontos de interação do sistema em execução. O descritor *sending of* indica a saída de uma primitiva por um ponto, e *receiving of* a chegada de uma mensagem em um ponto. Nestes descritores também são identificados o módulo-filho ao qual a primitiva se refere, seguido do ponto de interação. Além de permitir o acréscimo de números de seqüência que indicam a ordem das mensagens enviadas.

Os operadores lógicos têm interpretações semelhantes às semânticas comumente encontradas em lógica temporal. Os operadores *and* e *or* têm seus significados conhecidos. O operador *otherwise* permite ao usuário definir um novo comportamento, caso a restrição inicial não tenha sido satisfeita. O operador unário *eventually* indica que um determinado evento irá ocorrer em algum instante de tempo, a partir do instante de observação do sistema. Já o outro operador unário *henceforth* indica que um dado evento sempre ocorrerá, também a partir do instante de observação do sistema.

Os outros operadores lógicos podem ser definidos com base nos elementos descritos anteriormente e em operadores comumente utilizados, com a interpretação esperada. Vejamos a seguir, onde a e b são fórmulas e t uma variável temporal:

1. a `leadsto` b $\equiv$ a `implies eventually` b
2. a `forbids` b $\equiv$ a `implies henceforth not` b
3. a `at` t $\equiv$ a `and` (t=now)

# 3 O Sistema de Transmissão CBR

O modelo que criamos para exemplificar a especificação das restrições temporais-foi um sistema de transmissão envolvendo uma fonte CBR, um meio de transmissão sujeito a perdas e um receptor passivo, que apenas recebe os dados enviados pelo emissor. Os requisitos de QoS impostos ao modelo são a taxa de transmissão da fonte e o *jitter* introduzido pelo meio de comunicação. A especificação completa deste modelo pode ser encontrada no apêndice A.

Nas subseção que segue, descreveremos primeiramente como é a arquitetura do sistema como um todo, enfatizando o modelo do serviço prestado, indicando os módulos envolvidos e as mensagens trocadas por seus canais de comunicação. Posteriormente detalharemos como é o comportamento dinâmico de cada um dos módulos do sistema. Finalizaremos a seção mostrando as restrições temporais presentes na especificação, explicando como elas foram descritas.

## 3.1 Modelo de Serviço

O presente sistema apresenta uma arquitetura muito simples, onde temos um módulo emissor que gera pacotes a taxa constante, donde a denominação de Sistema de Transmissão CBR. Existe um módulo emissor responsável apenas pela recepção dos pacotes gerados pelo emissor. O meio de comunicação recebe os pacotes do emissor, e os repassa ao receptor, podendo atrasar esta entrega ou eventualmente perder estes dados. Desta forma, simulamos problemas que existem em sistemas reais, em virtude das limitações de banda dos enlaces de comunicação e filas nos *buffers* dos roteadores de uma rede.

As mensagens ou primitivas de serviço em Estelle estão relacionadas às funções de transmissão e recepção de dados, bem como ao início e fim desta transmissão. Como primitivas do módulo emissor, temos:

- $START$: pedido de início de transmissão pelo usuário ao emissor.
- $STOP$: pedido de finalização de transmissão pelo usuário ao mesmo emissor.

Para o meio de comunicação, temos apenas primitiva de pedido de envio de dados pelo emissor ($DATREQ$) e de concretização da entrega do pacote ao receptor ($DATIND$). O serviço prestado neste caso é sem conexão e sem confirmação. Como o emissor não presta serviço a nenhuma entidade em particular, ele não possui primitivas associadas ao módulo que expressa seu comportamento. Podemos visualizar a arquitetura modular do sistema em questão na figura 1.

Nesta figura, estão representados os canais de comunicação entre os módulos e os seus respectivos pontos de interação. Pode-se verificar a existência de um módulo-pai (*SysSndRecv*) que engloba os módulos *SENDER*, *RECEIVER* e *MEDIUM*, pois desta forma torna-se possível expressar restrições temporais fim-a-fim neste sistema. Estas devem ser descritas necessariamente no escopo do corpo de módulo pai, para que se tenha acesso a objetos de todos os módulos envolvidos, como conteúdo das filas nos ip de módulos diferentes que estão se comunicando.

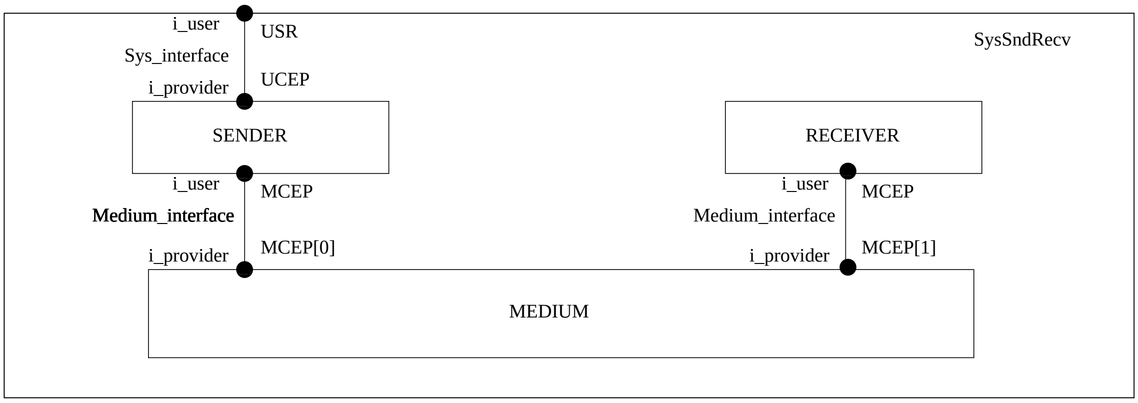


Figura 1: Modelo de Serviço do Sistema de Transmissão CBR.

Através do ponto de interação USR, é feita a chamada das primitivas do módulo emissor pelo usuário do sistema. Este encontra-se conectado pelo canal Sys_interface ao ip (ponto de interação) UCEP do emissor. O emissor por sua vez envia suas mensagens de pedido ao meio pelo ip MCEP, conectado pelo canal Medium_interface ao ip MCEP[0] do meio. Da mesma forma, o meio entrega os pacotes de dados ao receptor, pelo ip MCEP[1], que se conecta ao ip MCEP do receptor por um outro canal Medium_interface.

## 3.2 Módulos Emissor e Receptor

A figura 2 mostra o modelo, em Máquina de Estados Finita (MEF), do comportamento do módulo emissor. Este emissor inicia sua execução em seu estado de repouso (*SenderIdle*). Quando este emissor recebe o pedido do usuário para iniciar a transmissão, pela primitiva *STOP*, ele inicia a transmissão de seus dados, no estado *StartTransfer*, enviando em intervalos de 30 ms, através da mensagem *DATREQ*, onde o único parâmetro é o número de seqüência ($n$), que é acrescido a cada envio. Este ciclo é interrompido se o usuário entrar com um pedido de interrupção da transmissão (*STOP*) ou se o número de seqüência chegou ao máximo que existe para transmitir (*MAXSEQ*). Após isto, o emissor volta ao repouso.

O trecho de código abaixo mostra a especificação das restrições temporais associadas a este módulo:

```
time constraints
    nameTransferRate:
        forall t:time;
        forall seqnr:integer;
        henceforth(sending of MCEP.DATREQ [seqnr] at t leadsto
        sending of MCEP.DATREQ [seqnr + 1000*RATE/PKTSIZE - 1]
        at (now < t + 1000));
    end;
```

A variável $t$ é definida como sendo pertencente ao domínio do tempo. Já *seqnr* é um número associado a cada pacote enviado pela fonte. Ajustando os valores

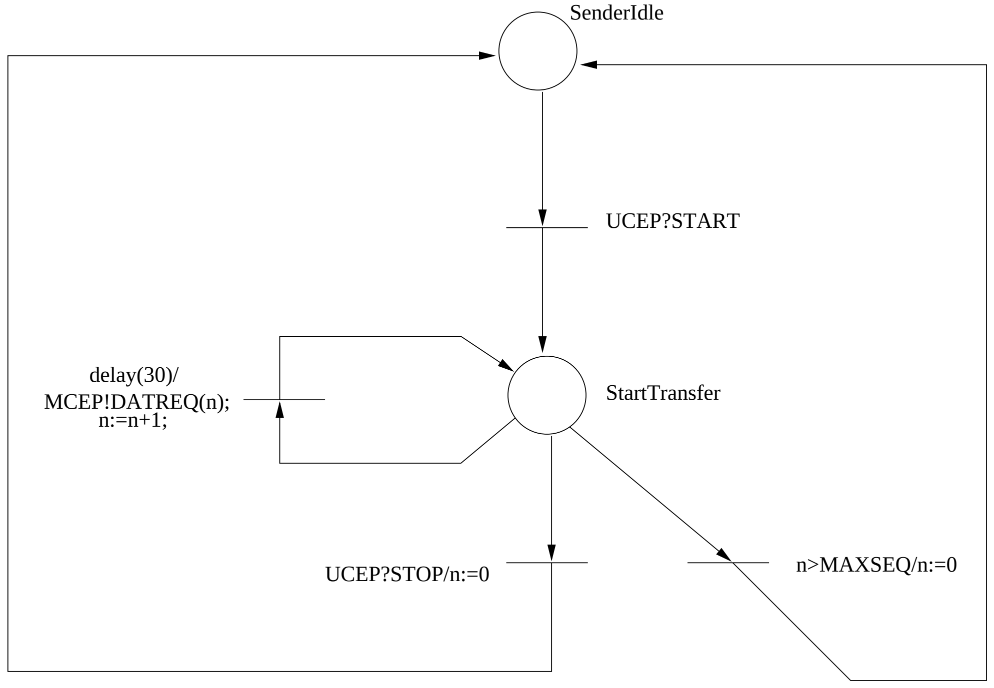


Figura 2: Comportamento do Módulo Emissor.

de *RATE* pode-se conseguir a taxa de transmissão desejada e a variável *PKTSIZE* representa o tamanho do pacote de dados, que serve para determinar, junto com a taxa, quantos pacotes devem ser enviados num dado intervalo de tempo para se conseguir a taxa ideal. A combinação dos operadores *henceforth* e *leadsto* diz que, sempre que houver o envio de um pacote em um tempo *t*, outro pacote com outro número de seqüência será enviado antes de *(t + 1000)* unidades de tempo, que nesta especificação, é dado em milissegundos.

No nosso exemplo, utilizamos, em particular, valores que levaram a uma taxa de transmissão de 96 Kbps, com um pacote de 2880 bits transmitido a cada 30 ms, valores típicos para aplicações de aúdio [6].

Como já foi mencionado, o receptor do nosso modelo é extremamente simples e possui um comportamento passivo. Desta forma, sua modelagem não necessita de uma MEF para sua representação, mas apenas de uma transição associada à recepção da mensagem *DATIND*, que simula a chegada dos pacotes enviados. A única ação executada por este módulo é anunciar a chegada do pacote ao usuário que esteja interagindo com o sistema.

## 3.3 Modelagem do Meio de Transmissão

O meio de comunicação insere um atraso no envio dos pacotes e descarta-os sempre que a taxa de transmissão for maior do que a capacidade do meio, de absorvê-los. Para especificar o meio com o comportamento desejado, foi criado um *buffer*, no qual eram depositados os pacotes vindos da fonte, e o meio por sua vez, tem a função de retirar um a um esses pacotes do *buffer*, dar um atraso aleatório entre 100 e 200 milissegundos e envia-los para o receptor.

O meio só retira um outro pacote do *buffer* depois de ter enviado o primeiro ao receptor, e este primeiro pacote é retido no meio por um tempo aleatório, dentro do intervalo já citado. Portanto, a taxa com que o meio retira os pacotes do *buffer* também é aleatória. Dessa forma, os pacotes terão valores de atraso diferentes quando chegarem ao receptor. Isto é feito para tentar retratar de maneira fiel o comportamento de um meio de comunicação real, no qual, o atraso na chegada de cada pacote ao receptor depende da situação momentânea de congestionamento da rede de comunicação.

A figura 3 retrata este comportamento. A ação *PutQ* ali representada é a função responsável por esse enfileiramento aqui representado. Ao receber o primeiro pacote do emissor para ser transmitido, ele é enfileirado, e sempre que chegarem novos pacotes estes também o são. Na função *PutQ* também é simulada a perda dos pacotes no meio, que é representada pelos instantes em que a fila está cheia, e novos dados continuam chegando. Após retirar um pacote da fila para ser transmitido, o meio vai para um estado de verificação, onde ele avalia o valor da variável booleana *Qempty*, que assume valor verdadeiro, quando não há mais dados a transmitir. Com isto, consegue-se evitar que pacotes fiquem perdidos no meio, encerrando a transmissão apenas quando o último dado de um ciclo chegou ao receptor.

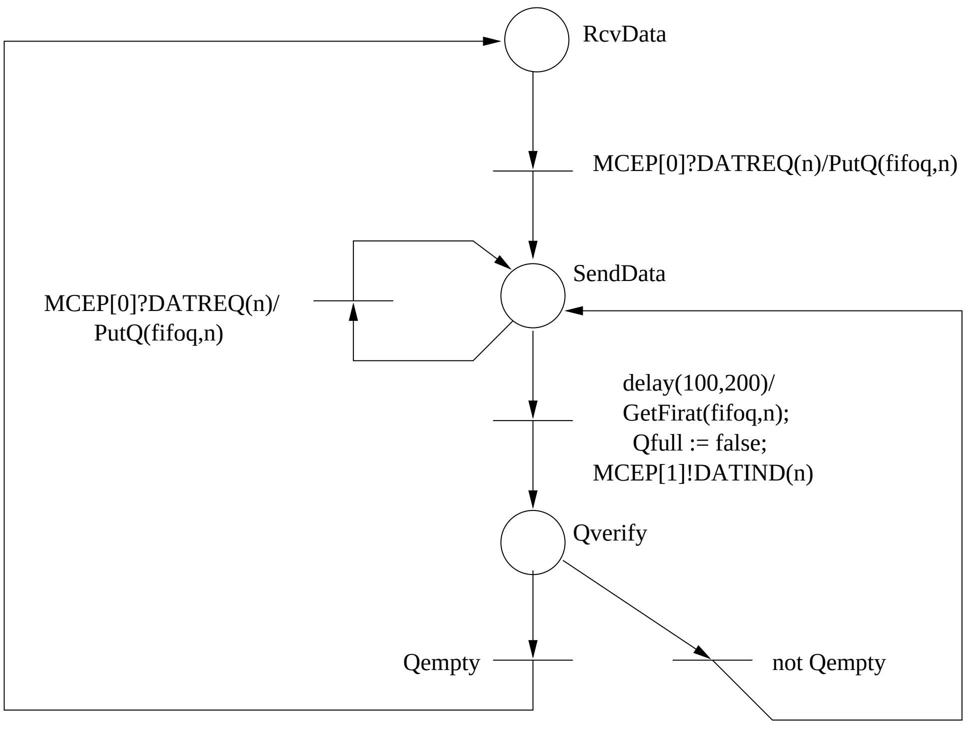


Figura 3: Comportamento do Meio de Comunicação.

## 3.4 Restrições Temporais do Sistema

Nesta seção, vamos descrever as restrições temporais do sistema fim-a-fim, ou seja, essas restrições não são associadas a um módulo em particular, e sim, associadas à comunicação entre os diversos módulos que compõe o sistema. O trecho de código abaixo mostra a especificação do *jitter* global do sistema. A combinação dos operadores *henceforth* e *leadsto* permite-nos expressar que, se um pacote é enviado em um tempo $t$, este pacote chegará ao receptor em um tempo $(t{+}MIN < now < t{+}MAX)$. O *jitter* caracteriza a variação de atrasos entre os pacotes, este conceito é de fundamental importância para mídias que exigem cadência na transmissão das informações.

```
time constraints
    name jitter :
        forall t:time;
        forall seqnr: integer;
        henceforth (
        sending of SENDER.MCEP.DATREQ[seqnr] at t
        leadsto receiving of RECEIVER.MCEP.DATIND[seqnr]
        at (t+MIN<now<t+MAX));
    end;
end;
```

# 4 Comentários

Este relatório teve como objetivo apresentar um sistema de transmissão CBR que nos permitiu especificar requisitos de QoS bastante comuns entre as aplicações multimídias. Fizemos também uma análise semântica dos operadores utilizados para especificar as restrições temporais, descritas através de fórmulas de lógica temporal.

Como um teste inicial, foram feitas algumas simulações num Sistema de Auxílio ao Projeto de Protocolos na Linguagem Estelle (*CAD-Estelle*) [7]. Como este sistema no presente momento, não suporta totalmente a especificação de restrições temporais, somente verificamos, como ele se comportava, e se atendia ao modelo proposto. Os resultados indicaram o comportamento esperado, com o descarte de alguns pacotes e o envio de outros. Não foi possível, em virtude desta falta de suporte total ao *Estelle Tempo-Real*, verificar as possíveis violações de QoS.

O suporte a especificações de protocolos com restrições temporais está sendo incorporado ao CAD-Estelle,bem como uma interface visual de edição de protocolos e a possibilidade de análise dos parâmetros de QoS obtidos na simulação destes [8]. A interface visual permitirá a geração da especificação Estelle apartir da modelagem em MEF do protocolo. O Sistema de Transmissão será simulado neste ambiente novo, permitindo que se verifique seu desempenho, e sejam tirados dados quantitativos a seu respeito, como gráficos de vazão e atraso.

# Referências


[1] M. Diaz, *Formal Description Technique ESTELLE*. North-Holland, 1989.

[2] S. Fischer, "Real-Time ESTELLE", tech. rep., Universität Manheim, Germany, 1996.

[3] S. Fischer, "On the Suitability of ESTELLE for Multimedia Systems", tech. rep., Universität Manheim, Germany, 1995.

[4] R. Gotzhein, "Temporal Logic and Applications - A Tutorial", *Computer Networks and ISDN Systems*, vol. 24, pp. 203–218, 1992.

[5] S. Leue e S. Fischer, "Formal Methods for Broadband and Multimedia Systems", *Computer Networks and ISDN Systems*, vol. 30, pp. 865–899, 1998.

[6] J.-P. Courtiat e R. C. de Oliveira, "On RT-LOTOS and its application to multimedia protocol specification and validation", in *Proc. Int. Conf. on Multimedia Networking (MmNet'95)*, Aizu, Japan, 1992.

[7] A. C. P. Pedroza, C. C. Goulart, P. R. O. Valim e R. de Oliveira, "Um Sistema de Auxílio ao Projeto de Protocolos de Comuniação para Redes de Computadores", in *Seminário Franco-Brasileiro de Sistemas de Informação Distribuídos (SFBSID'89)*, Florianópolis, SC, Brasil, 1989.

[8] G. M. Delfino, J. V. dos Santos Filho, R. C. Roman, J. L. S. Leão e A. C. P. Pedroza, "Sistema Integrado de Projeto de Protocolos de Comunicação com Restrições Temporais (ProtCAD/RT)", tech. rep., GTA-UFRJ, 2000.

# A Especificação Completa do Sistema de Transmissão CBR

```
Specification Sender_Receiver Systemactivity;
default individual queue; timescale miliseconds;
{$ external NETUSER}
const
        MAXSEQ = 50; {Numero de sequencia maximo a ser transmitido -
                    marca o fim da transmissao}
        PKTSIZE = 2880;{Tamanho de pacotes de dados em bits}
        RATE = 96; {Taxa de transmissao em Kbps}
        MAXQ = 15; {Tamanho maximo da fila que simula perda no meio em numero de pacotes}
        MIN = 100; {Atraso minimo em ms, para transferencia fim a fim de dados}
        MAX = 150; {Atraso maximo em ms, no mesmo caso do anterior -
                    ambos servem para calcular jitter}
Type
        string = array [0..80] of char;
        data_type = integer; {Para simplificar a especificacao, os pacotes carregam
                    apenas um numero de identificacao deste}
        QType = array [0..MAXQ] of data_type;{Tipo de fila que simula a perda
                    no meio de transmissao}

(*************Procedimentos de interface com o usuario da simulacao********)

        procedure WriteLn; primitive;
        procedure WriteInt ( i : integer ); primitive;
        procedure WriteString ( s : string ); primitive;

(*************Canal entre usuario e entidade emissora****************)

        Channel Sys_interface(i_user, i_provider);
        by i_user:
        START; {Pedido do usuario para iniciar transmissao}
        STOP; {Pedido do usuario para finalizar transmissao}

(*************Canal entre emissor/receptor e meio****************)

        channel Medium_interface(i_user, i_provider);
        by i_user:
        DATREQ(n:data_type);
        by i_provider:
        DATIND(n:data_type);

(*************Definicao dos modulos que compoem a especificacao ********)


(***********Modulo mais externo que engloba emissor/receptor e meio***********)
```

```
module SendRecv activity;
        ip USR:Sys_interface(i_provider);
end;
body SendRecvBody for SendRecv;
        ip IUSR:Sys_interface(i_user);

(***********Modulo Emissor**********************)

module SenderType activity;
ip UCEP:Sys_interface(i_provider);
        MCEP:Medium_interface(i_user);
end;
body SenderBody for SenderType;
var
        n:data_type;
state
        SenderIdle,
        DtTransfer;

(********Inicializacao do modulo emissor************)

initialize
to SenderIdle
name SenderStart:
        begin
        n:=0;
        end;

(*********Comportamento do modulo emissor************)


(********Pedido de usuario do sistema para iniciar a transmissao******)

trans
from SenderIdle
        when UCEP.START
        to DtTransfer
        name StartTransfer:
begin
end;

(*******Pedido de usuario para finalizar transmissao*****************)

from DtTransfer
        when UCEP.STOP
to SenderIdle
        name EndTransfer:
```

```
                begin
                n:=0;
                end;

(*******Dados sendo enviados a taxa constante**********************)

        from DtTransfer
        delay(30)
        to same
        name Tranfering:
                begin
                output MCEP.DATREQ(n);
                n:=n+1;
                end;

(********Nada mais a transmitir**********************************)

        from DtTransfer
        provided n>MAXSEQ
        to SenderIdle
        name AllSent:
                begin
                n:=0;
                end;

(*******Restricoes Temporais do modulo emissor - Taxa ********)

        time constraints
        name TransferRate:
        forall t:time;
        forall seqnr:integer;
                henceforth(
                sending of MCEP.DATREQ[seqnr] at t
                leadsto sending of MCEP.DATREQ[seqnr+ 1000*RATE/PKTSIZE]
                at (now <t + 1000));
                end;
        end; {Corpo do emissor}

(**********************Modulo Receptor**********************************)
        module ReceiverType activity;
        ip MCEP:Medium_interface(i_user);
        end;
        body ReceiverBody for ReceiverType;
        var
                n:data_type;

(***************Inicialização do modulo receptor**********************)
```

```
        initialize
        name RecvInit:
        begin
        end;

(*******Comportamento do modulo receptor-apenas recebe dados****************)

        trans
        when MCEP.DATIND(n)
        name DtRecv:
        begin
        end;
        end; {Corpo do receptor}

(*******************Modulo Meio de Transmissao**************************)

        module MediumType activity;
        ip MCEP: array [0..1] of Medium_interface(i_provider);
        end;
        body Medium_Body for Medium_Type;
        var num,n:data_type;
                fifoq: QType; {Fila que simula perdas e atrasa pacotes para entrega}
                tail:integer; {Ponteiro para fim da fila}
                i: integer;
                Qempty: boolean; {Indica se fila esta vazia}
                Qfull: boolean; {Indica se fila esta cheia}
        state RcvData,
                Qverify,
        SndData;

(**************Inclusao de pacote na fila para transmissao**********)

        Procedure PutQ(var myQ:QType;elem:data_type);
        begin
                if not Qfull
                then begin
                myQ[tail] := elem;
                WriteString('Pacote enfileirado =');
                WriteInt(elem);
                WriteLn;
                if tail = 0 then Qempty := false;
                if tail < MAXQ then tail := tail+1
                else Qfull := true;
                end
                else begin
                WriteString ('Pacote ');
```

```
            WriteInt(elem);
            WriteString (' descartado.');
            WriteLn;
            end;
    end;

(***********Retirada do primeiro pacote da fila para ser transmitido******)

    Procedure GetFirst(var myQ:QType;var first:data_type);
            var i: integer;
            begin
            first := myQ[0];
            for i:= 0 to (tail-1)
            do myQ[i] := myQ[i+1];
            if not Qfull then tail := tail - 1;
            myQ[tail] := -1;
            if tail = 0 then Qempty := true;
            end;

(*********Inicializacao do modulo meio de transmissao******************)

    initialize
    to RcvData
    name MediumStart:
    begin
            tail:=0;
            Qempty := true;
            Qfull := false;
            for i:=0 to MAXQ
            do fifoq[i]:=-1;{Elemento=-1 marca lugar nao usado na fila}
            end;

(*******Comportamento do modulo meio***************)

(*******Chegam dados do receptor, meio enfileira e depois transmite*******)

    trans
    from RcvData
    when MCEP[0].DATREQ(n)
            to SndData
            name DtRcv:
            begin
            num := n;
            PutQ(fifoq,num);
            end;

(*******Novos dado chegando***********)
```

```
        from SndData
        when MCEP[0].DATREQ(n)
        name MoreData:
                begin
                num := n;
                PutQ(fifoq,num);
                end;

(******Encaminhamento de pacotes que estavam no meio*************)

        from SndData
        delay (100,200)
        to Qverify
        name DtDeliver:
                begin
                GetFirst(fifoq,num);
                QFull := false;
                WriteString('Pacote retirado =');
                WriteInt(num);
                WriteLn;
                output MCEP[1].DATIND(num);
        end;

(******Nao ha mais nada a ser transmitido**********)

        from Qverify
        provided Qempty
        to RcvData
        name verify:
        begin
        end;

(****Ainda ha pacotes esperando para serem enviados*****)

        provided not Qempty
        to SndData
                name morepkts:
                begin
                end;
                end; {Corpo do meio}

(********Variaveis que representam os modulos*******************)

        modvar SENDER:SenderType;
                RECEIVER:ReceiverType;
        MEDIUM:MediumType;
```

```
(*********Inicializacao dos modulo mais externo*******************)

        initialize
        name SndRcvStart:
                begin
                init SENDER with SenderBody;
                init RECEIVER with ReceiverBody;
                init MEDIUM with MediumBody;
                connect IUSR to SENDER.UCEP;
                connect SENDER.MCEP to MEDIUM.MCEP[0];
                connect RECEIVER.MCEP to MEDIUM.MCEP[1];
                end;

(*******Comportamento do modulo mais externo***********************)

        trans
        when USR.START
                name STARTTRANS:
                begin
                output IUSR.START;
                end;
        when USR.STOP
                name STOPTRANS:
                begin
                output IUSR.STOP;
                end;

(*******Restricoes temporais do modulo mais exteno - jitter fim-a-fim******)

        time constraints
                name jitter:
                forall t:time;time constraints
                forall seqnr: integer;
                henceforth (
                sending of SENDER.MCEP.DATREQ[seqnr] at t
                leadsto receiving of RECEIVER.MCEP.DATIND[seqnr]
                at (t+MIN<now<t+MAX));
        end;

end; {Corpo do modulo mais externo}

(*********Variavel modulo do sistema como um todo*************)

        modvar SysSndRecv:SendRecv;

(********Inicializacao da especificacao*********************)
```

```
    initialize
    name SysInit:
        begin
        init SysSndRecv with SendRecvBody;
        end;
end.{Especificacao}
```