Referências Bibliográficas

1 3GPP. **TR 45.050-621: Background for Radio Frequency (RF) requirements**, mar. 2005. Disponível em: <http://www.3gpp.org/ftp/Specs/html-info/45-series.htm>. Acesso em: 03 fev. 2006.

2 BALAZS, A. **Push-to-talk Parformance over GPRS**. Proceedings of the 7th ACM international symposium on Modeling, out. 2004.

3 BRADY, P. T. **A model for generating on-off speech patterns in twoway conversation**. Bell System Technical Journal, set. 1969.

4 CARVALHO, L. et al. **An E-Model Implementation for speech Quality Evaluation in VoIP Systems**. 10th IEEE Symposium on Computers and Communications, jun. 2005.

5 COTA, N. **Apostila do Curso ST III** do Departamento de Engenharia Eletrônica e das Comunicações do Istituto Superior de Engenharia de Lisboa (ISEL), 2000. Disponível em: <http://www.deetc.isel.ipl.pt/sistemastele/ST3/index.htm>. Acesso em: 03 fev. 2006.

6 DINIZ, M. C.; SILVA, E. S. **Especificação e Geração de Modelos Markovianos para Análise de Desempenho e Confiabilidade de Sistemas**. Revista Brasileira de Computação, 1991.

7 ERICSSON. **Requirements for GPRS Evolution Towards Providing 3G Services**, fev. 1999. Disponível em: <http://www.winlab.rutgers.edu/pub/symposiums/GPRS/Agenda.html>. Acesso em: 03 fev. 2006.

8 HALONEN, T; ROMERO, J.; MELERO, J. GSM, **GPRS and EDGE Performance (Evolution Towards 3G/UMTS)**. Editora Wiley, 2003.

9 HOENE, C.; KARL, H.; WOLISZ, A. **A Perceptual Quality Model for Adaptitative VoIP Applications**. Special issue Performance Evaluation of Wireless Networks and Communications of the Computer Communications Journal, maio 2005. Disponível em: <http://www.ece.ubc.ca/~mamun/research/research.html>. Acesso em: 02 fev. 2006.

10 HOLMA, H. et al. **Performance of Adaptive Multirate (AMR) Voice in GSM and WCDMA**. 57th IEEE Semiannual Vehicular Technology Conference. VTC 2003-Spring, abr. 2003.

11 ITU-T. **Recommendation G.107: The Emodel**, A Computational Model for Use in Transmission Planning, 2003.

12 JIANG, W.; SCHULZRINNE, H. **Analysis of On-Off atterns in VoIP and Their Effect on Voice Traffic Aggregation**. 09th IEEE International Conference on Computer Communications and Networks, out. 2000.

13 KIM, P.; BALAZS, A.; BROEK, E. **IMS-based Push-to-Talk over GPRS/UMTS**. IEEE Wireless Communications and Networking Conference, mar. 2005.

14 LAND. **Laboratório de Modelagem, Análise e Desenvolvimento de Redes e Sistemas de Computação da UFRJ**. Disponível em <http://www.land.ufrj.br/>. Acesso em: 01 fev. 2006.

15 LEÃO, R. M. M.; SILVA, E. S. **The TANGRAM-II Environment**. Proceedings of the 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools, mar. 2000.

16 LUCENT. **RLC/MAC Simulation for GPRS/EDGE**, fev. 1999. Disponível em: <http://www.winlab.rutgers.edu/pub/symposiums/ GPRS/Agenda.html>. Acesso em: 03 fev. 2006.

17 MATTA, J. et al. **User perception in audio: A source and Channel Rate Adaptation Algorithm for AMR in VoIP Using the Emodel**. Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video, jun. 2003.

18 NARBUTT, M.; MURPHY, L. **Improving Voice Over IP Subjective Call Quality**. IEEE Communications Letters, maio 2004.

19 POC. **Push to Talk over Cellular 2.0**. Comneon, Ericsson, Motorola, Nokia, Siemens, jun. 2004. Disponível em: <http://www.motorola.com/content/ 0,,2038,00.html>. Acesso em: 03 fev. 2005.

20 QUEIROZ, H. M. O.; GUEDES, L. G. R. **Ambiente de Simulação para o Sistema EDGE (E-GPRS) com Mecanismo de Decisão por Codificação**

por Limiar de Erro. I Encontro Regional de Modelagem e Análise de Sistemas promovido pela UCG, out. 2004. Disponível em: <http://wsmartins.net/ermacs/trabalhos_aceitos_1.htm>. Acessado em: 18 fev. 2006.

21 SEO, J.; WOO, S. J.; BAE, K. **A Study on the Application of an AMR Speech Codec to VoIP**. IEEE International Conference on Acoustics, Speech, and Signal Processing, maio 2001.

22 SHEN, Q. **Performance of VoIP over GPRS**. 17th International Conference on Advanced Information Networking and Applications, mar. 2003.

23 TGIF. **Tangram Graphic Interface Facility**. Disponível em: <http://bourbon.usc.edu:8001/tgif/>. Acesso em: 01 fev. 2006.

24 UWCC. **Enhanced Data-rates for Global Evolution (EDGE): An Overview**. Universal Wireless Communications Consortium, fev. 2000. Disponível em: <http://www.3gamericas.org/English/Technology_Center/Tutorials/>. Acesso em: 02 fev. 2006.

25 YACOUB, M. D. **Wireless Technology: Protocols, Standards, and Techniques**. Editora CRC Press, 2002.

26 YALLAPRAGADA, R.; KRIPALANI, V.; KRIPALANI, A. **EDGE: a Techology Assessment**. IEEE International Conference on Personal Wireless Communications, dez. 2002.

27 JARVINEN, K. **Standardization of the Adaptive Multi-Rate Codec**. Proceedings of European Conference on Signal Processing, 2000.

Glossário

Buffer Espaço de memória física utilizada para armazenamento de dados que aguardam algum tipo de processamento.

Encapsular Nas redes de pacotes, ato de adicionar cabeçalhos à informação no lado do transmissor para processamento pelas camadas pares no lado do receptor.

Full Duplex Comunicação em que a transmissão acontece em ambos os sentidos simultaneamente. Exemplo: telefonia convencional.

Half Duplex Comunicação em que a transmissão acontece apenas em um sentido de cada vez. Exemplo: rádio PTT, PoC.

Overhead Bits adicionados à informação para compor os cabeçalhos de cada camada (*overhead* de protocolo). Mensagens de controle e gerenciamento sem informação do usuário (*overhead* de sinalização).

PTT Serviço de comunicação instantânea *half duplex* baseado na transmissão de voz em modo analógico (*Push-to-Talk*).

Talkspurt Cada um dos trechos de fala, em meio aos trechos de silêncio, detectados pelo codificador de voz operando em modo DTX [19].

Talkburst Quantidade delimitada da fala capturada desde o momento em que o usuário inicia o discurso, isto é, aperta o botão, até o momento em que ele o solta. Um *talkbust* pode conter vários *talkspurts* [19].

TU-3 Ambiente definido como uma área urbana típica com o terminal móvel se movimentando a uma velocidade de 03 Km/h (*Typical Urban at 3 Km/h*).

Apêndice A – Código Parcial do Objeto *PoC_Max*

Neste apêndice é apresentado parte do código-fonte dos objetos que compõem o modelo *PoC_Max* na plataforma do Tangram-II. Somente um exemplo de cada objeto é apresentado (*MS_Left*, *CH_Left*, *BSS_Left* e *Core*). O código completo pode ser obtido, de acordo com os direitos autorais deste trabalho, através dos endereços eletrônicos *claudio.lcs@gmail.com* e *lcs@cetuc.puc-rio.br* para uso em outros trabalhos acadêmicos.

Para a implementação deste modelo, a versão 3.0 do Tangram-II (*tangram2-3.0*) foi instalada em uma máquina executando o sistema operacional *Linux Red Hat® 9.0* (*kernel .2.2-5*) com processador AMD Duron 900 MHz e 384 MB de memória física.

```
/**********************************************************************************************************************/
Model PoC, object MS_Left (15.abr.05) by Luis Claudio dos Santos. This object simulates MAX_CL
participants in PoC sessions. The stations upload RLC Blocks to BSS according to C/I getting  from CH object.
/**********************************************************************************************************************/
Object_Desc MS_Left (
Declaration {
Const
Port: RX_CH, RX_BSS, TX_BSS, FLOOR;
Integer: MAX_CL, MAX_PDTCH, INACTIVE, TALKER, LISTENER, WAITING_RESPONSE,
WAIT_TBF_ASSIGN, WAIT_FLOOR_REQUEST, WAIT_FLOOR_RELEASE, ID, TYPE, DATA, PIG1, PIG2,
SIGNALING, USERDATA, FLOOR_TAKEN, FLOOR_RELEASING, FLOOR_RELEASED, DEBUG, PTIME,
AMR_TX, AMR_SID;
Float: EDGE_RATE, TBF_TIME, REQ_TIME, SCALE, SHAPE, MEAN01, MEAN23, MEAN45, MEAN67,
VARIANCE, COMPRESS;
Object: CH, BSS, BSS_R;

State Var
Integer: Application_Buffer[MAX_CL], TX_Buffer[MAX_CL], MS_Status[MAX_CL], MCS[28],
CH_dB_Status[MAX_CL], Next_MS[MAX_PDTCH], Total_Packets, Total_Bursts, Amr_TX[3], Amr_Index;
Float: Total_Delay, Talkburst[4], MS_Burst_Time[MAX_CL], MS_Block_Time[MAX_CL];
}

Initialization {
/*CONSTANTS*/
/*Parameter of Simulations*/
MAX_CL = 02                /*Max number of clients per sector.*/
MAX_PDTCH = 02             /*Max number of PDTCH per sector.*/
PTIME = 08                 /*Number of voice frames per packet.*/
AMR_TX = 148               /*103, 148, 244 for AMR 5.15, 7.40 or 12.2 Kbps.*/
AMR_SID = 39               /*Confort noice sending in voice gaps in the speech.*/
COMPRESS = 1               /*(0.2 or 1) for (64 or 320) overhead per packet.*/
DEBUG = 2                  /*Level of printed debug info.*/

/*Rates*/
EDGE_RATE = 50             /*One RLC Block each 20 ms (12 in 240 ms).*/
TBF_TIME = 0.36            /*Signaling for UL TBF assign lasts 360 ms (aver.).*/
REQ_TIME = 0.7             /*Signaling for req. floor lasts 700 ms (aver.).*/

/*Possible States of a MS*/
INACTIVE = 0
TALKER = 1
LISTENER = 2
```

```
WAITING_RESPONSE = 3
WAIT_TBF_ASSIGN = 4
WAIT_FLOOR_REQUEST = 5
WAIT_FLOOR_RELEASE = 6

/*Message Indexes*/
ID = 0          /*The user ID.*/
TYPE = 1        /*Signaling or userdata.*/
DATA = 2        /*The information.*/
PIG1 = 3        /*Additional piggybacking information.*/
PIG2 = 4        /*Additional piggybacking information.*/

/*Type of Messages*/
SIGNALING = 0
USERDATA = 1

/*Signaling Messages from BSS*/
FLOOR_TAKEN = 3                 /*Put MS in LISTENER state.*/
FLOOR_RELEASING = 4             /*Put MS in WAIT_FLOOR_RELEASE state.*/
FLOOR_RELEASED = 5              /*Put MS in INACTIVE state.*/

/*Parameters of Distribution*/
SCALE = 8.0             /*Size of talkbursts. Modeled by WEIB pdf.*/
SHAPE = 6.0
MEAN01 = 4.0            /*Wait time before talking. Modeled by LOGNORM pdf.*/
MEAN23 = 4.0
MEAN45 = 4.0
MEAN67 = 4.0
VARIANCE = 0.5

/*MCS from [ERIC 99]; iLA.*/
/*MCS = [0, 178, 225, 267, 310, 345, 391, 449, 499, 534, 561, 635, 705, 766, 813, 844, 863, 879, 929, 983,
1037, 1065, 1103, 1130, 1150, 1165, 1173, 1177]*/
/*MCS from [ERIC 99]; LA BLER < 1%.*/
/*MCS = [0, 174, 178, 178, 178, 178, 178, 178, 178, 178, 178, 178, 178, 225, 225, 279, 279, 325, 329, 329,
449, 449, 596, 596, 596, 898, 1173, 1177]*/
/*MCS from [ERIC 99]; LA BLER < 2%.*/
/*MCS = [0, 174, 178, 178, 178, 178, 178, 178, 178, 178, 178, 178, 225, 279, 279, 325, 325, 325, 449, 449,
604, 604, 604, 898, 898, 898, 1177, 1177]*/
/*MCS from [ERIC 99]; iLA BLER < 3%.*/
MCS = [0, 174, 178, 178, 178, 178, 178, 178, 178, 225, 225, 279, 279, 279, 325, 325, 325, 449, 449, 604, 604,
604, 898, 898, 1130, 1146, 1173, 1177]
```

```
/*Ports*/
RX_CH = ch_ms_l          /*To receive msg from CH.*/
TX_BSS = ms_bss_l        /*To transmit msg to BSS.*/
RX_BSS = bss_ms_l        /*To receive msg from BSS.*/
FLOOR = wire_floor       /*To floor signaling.*/

/*Objects*/
CH = CH_Left
BSS = BSS_Left
BSS_R = BSS_Right

/*VARIABLES*/
/*Integer ones*/
Application_Buffer[] = 0
TX_Buffer[] = 0
MS_Status[] = 0
CH_dB_Status[] = 13      /*Starts at 20 dB in the first 20 ms.*/
Next_MS = [0, 1]
Total_Packets[] = 0
Total_Bursts = 0
Amr_TX = [103, 148, 244]
Amr_Index = 0
/*Float ones*/
Total_Delay = 0
Talkburst = 0
MS_Burst_Time = 0
MS_Block_Time = 0
}

Events {
/*-----------------------------------------------------------------------------------------------*/
/* After the MS pass to INACTIVE state, the participant pushes button   */
/* in L seconds. L is defined by a log-normal pdf.                       */
/*-----------------------------------------------------------------------------------------------*/
event = push_button_0_1 (LOGNORM, MEAN01, VARIANCE)
condition = ((Talkburst[0] > 0) && (MAX_CL > 1))
action = {
    int i, data_vec[5], ms_status[MAX_CL], tx_buffer[MAX_CL];
    get_st (ms_status, "MS_Status[]");
    get_st (tx_buffer, "TX_Buffer[]");
    for (i = 0; i <= 1; i++) {
      if ((ms_status[i] == INACTIVE) && ((get_simul_time() - MS_Burst_Time[i]) > MS_Block_Time[i])){
        data_vec[ID] = i;
        data_vec[TYPE] = USERDATA;
        data_vec[DATA] = (int)Talkburst[0];
        data_vec[PIG1] = (int)Talkburst[1];
        data_vec[PIG2] = (int)Talkburst[2];
        tx_buffer[i] = tx_buffer[i] + data_vec[DATA];
        ms_status[i] = WAIT_TBF_ASSIGN;
        if (DEBUG > 1) {
          fprintf (stdout, "%f: ", get_simul_time());
          fprintf (stdout, "[MS_L] MS%d: Push (%d)\n", i, tx_buffer[i]);
        }
        msg (FLOOR, BSS_R, data_vec);
      }
    }

    set_st ("MS_Status[]", ms_status);
    set_st ("TX_Buffer[]", tx_buffer);
};

event = push_button_2_3 (LOGNORM, MEAN23, VARIANCE)
condition = ((Talkburst[0] > 0) && (MAX_CL > 3))
action = {
    int i, data_vec[5], ms_status[MAX_CL], tx_buffer[MAX_CL];
    get_st (ms_status, "MS_Status[]");
    get_st (tx_buffer, "TX_Buffer[]");

    for (i = 2; i <= 3; i++) {
      if ((ms_status[i] == INACTIVE) && ((get_simul_time() - MS_Burst_Time[i]) > MS_Block_Time[i])){
        data_vec[ID] = i;
        data_vec[TYPE] = USERDATA;
        data_vec[DATA] = (int)Talkburst[0] * 0.9;
        data_vec[PIG1] = (int)Talkburst[1];
        data_vec[PIG2] = (int)Talkburst[2];
        tx_buffer[i] = tx_buffer[i] + data_vec[DATA];
        ms_status[i] = WAIT_TBF_ASSIGN;
        if (DEBUG > 1) {
          fprintf (stdout, "%f: ", get_simul_time());
          fprintf (stdout, "[MS_L] MS%d: Push (%d)\n", i, tx_buffer[i]);
        }
        msg (FLOOR, BSS_R, data_vec);
      }
    }

    set_st ("MS_Status[]", ms_status);
    set_st ("TX_Buffer[]", tx_buffer);
};

event = push_button_4_5 (LOGNORM, MEAN45, VARIANCE)
condition = ((Talkburst[0] > 0) && (MAX_CL > 5))
action = {
    int i, data_vec[5], ms_status[MAX_CL], tx_buffer[MAX_CL];
    get_st (ms_status, "MS_Status[]");
    get_st (tx_buffer, "TX_Buffer[]");

    for (i = 4; i <= 5; i++) {
      if ((ms_status[i] == INACTIVE) && ((get_simul_time() - MS_Burst_Time[i]) > MS_Block_Time[i])){
        data_vec[ID] = i;
        data_vec[TYPE] = USERDATA;
        data_vec[DATA] = (int)Talkburst[0] * 0.8;
        data_vec[PIG1] = (int)Talkburst[1];
        data_vec[PIG2] = (int)Talkburst[2];
        tx_buffer[i] = tx_buffer[i] + data_vec[DATA];
        ms_status[i] = WAIT_TBF_ASSIGN;
        if (DEBUG > 1) {
          fprintf (stdout, "%f: ", get_simul_time());
          fprintf (stdout, "[MS_L] MS%d: Push (%d)\n", i, tx_buffer[i]);
        }
        msg (FLOOR, BSS_R, data_vec);
      }
    }

    set_st ("MS_Status[]", ms_status);
    set_st ("TX_Buffer[]", tx_buffer);
};

event = push_button_6_7 (LOGNORM, MEAN67, VARIANCE)
condition = ((Talkburst[0] > 0) && (MAX_CL > 7))
action = {
    int i, data_vec[5], ms_status[MAX_CL], tx_buffer[MAX_CL];
    get_st (ms_status, "MS_Status[]");
    get_st (tx_buffer, "TX_Buffer[]");
    for (i = 6; i <= 7; i++) {
      if ((ms_status[i] == INACTIVE) && ((get_simul_time() - MS_Burst_Time[i]) > MS_Block_Time[i])){
        data_vec[ID] = i;
        data_vec[TYPE] = USERDATA;
        data_vec[DATA] = (int)Talkburst[0] * 0.7;
        data_vec[PIG1] = (int)Talkburst[1];
        data_vec[PIG2] = (int)Talkburst[2];
        tx_buffer[i] = tx_buffer[i] + data_vec[DATA];
        ms_status[i] = WAIT_TBF_ASSIGN;
        if (DEBUG > 1) {
          fprintf (stdout, "%f: ", get_simul_time());
          fprintf (stdout, "[MS_L] MS%d: Push (%d)\n", i, tx_buffer[i]);
        }
        msg (FLOOR, BSS_R, data_vec);
      }
    }

    set_st ("MS_Status[]", ms_status);
    set_st ("TX_Buffer[]", tx_buffer);
};

/*-----------------------------------------------------------------------------------------------*/
/* Through downlink a TBF is stablished each 160 ms (mean expected). It */
/* lasts for all packets from the talkburst.                             */
/*-----------------------------------------------------------------------------------------------*/
event = tbf_uplink_assign (EXP, (1/TBF_TIME))
condition = (TRUE)
action = {
    int i, msg_vec[3], ms_status[MAX_CL];
    get_st (ms_status, "MS_Status[]");

    for (i = 0; i < MAX_CL; i++) {
      if (ms_status[i] == WAIT_TBF_ASSIGN){
        ms_status[i] = WAIT_FLOOR_REQUEST;
        if (DEBUG > 2) {
          fprintf (stdout, "%f: ", get_simul_time());
          fprintf (stdout, "[BSS_L] Activing MS%d!\n", i);
        }
      }
    }
    set_st ("MS_Status[]", ms_status);
};
```

```
/*-------------------------------------------------------------------------------------------*/
/* The signaling for floor request lasts 800 ms (mean expected). PoC    */
/* Release 2.0 recommend this signalling smaller than 1.6 s.                 */
/*-------------------------------------------------------------------------------------------*/
event = floor_request (EXP, (1/REQ_TIME))
condition = (TRUE)
action = {
    int i, ms_status[MAX_CL];
    get_st (ms_status, "MS_Status[]");

    for (i = 0; i < MAX_CL; i++) {
      if (ms_status[i] == WAIT_FLOOR_REQUEST){
        ms_status[i] = TALKER;
        if (DEBUG > 2) {
          fprintf (stdout, "%f: ", get_simul_time());
          fprintf (stdout, "[MS_L] MS%d requesting floor!\n", i);
        }
      }
    }

    set_st ("MS_Status[]", ms_status);
};

/*-------------------------------------------------------------------------------------------------------*/
/* The MS uploads one RLC Block in each PDTCH each 20 ms (EDGE_RATE)    */
/* according to MCS performance.                                                              */
/*-------------------------------------------------------------------------------------------------------*/
event = upload_rlc_blocks (DET, EDGE_RATE)
condition = (TRUE)
action = {
    int i, aux, ms_id, data_vec[3], tx_buffer[MAX_CL], ms_status[MAX_CL], next_ms[MAX_PDTCH];
    float ms_waiting_resp[MAX_CL];
    aux = 0;

    get_st (tx_buffer, "TX_Buffer[]");
    get_st (ms_status, "MS_Status[]");
    get_st (next_ms, "Next_MS[]");

    for (i = 0; i < MAX_PDTCH; i++) {
      ms_id = next_ms[i];
      aux = CH_dB_Status[ms_id];
      if (ms_status[ms_id] == TALKER) {
        if (tx_buffer[ms_id] >= MCS[aux]) {
          data_vec[ID] = ms_id;
          data_vec[TYPE] = USERDATA;
          data_vec[DATA] = MCS[aux];
          tx_buffer[ms_id] = tx_buffer[ms_id] - MCS[aux];
          msg (TX_BSS, BSS, data_vec);
          if (DEBUG > 3) {
             fprintf (stdout, "%f: ", get_simul_time());
             fprintf (stdout, "[MS_L] Sending %d to MS%d\n", data_vec[DATA], ms_id);
          }
        }
        else {
        if (tx_buffer[ms_id] > 0) {
          data_vec[ID] = ms_id;
          data_vec[TYPE] = USERDATA;
          data_vec[DATA] = tx_buffer[ms_id];
          tx_buffer[ms_id] = 0;
          msg (TX_BSS, BSS, data_vec);
          ms_status[ms_id] = WAITING_RESPONSE;
          ms_waiting_resp[ms_id] = get_simul_time();
          if (DEBUG > 3) {
            fprintf (stdout, "%f: ", get_simul_time());
            fprintf (stdout, "[MS_L] Sending %d to MS%d\n", data_vec[DATA], ms_id);
          }
        }
        }
      }
      /*** Set next MS to serve in channel i ***/
      next_ms[i] = next_ms[i] + MAX_PDTCH;
      if (next_ms[i] >= MAX_CL) next_ms[i] = i;
      /****************************************/
    }

    set_st ("Next_MS[]", next_ms);
    set_st ("MS_Status[]", ms_status);
    set_st ("TX_Buffer[]", tx_buffer);
};


/*-----------------------------------------------------------------------------------------------------------------------------*/
/* At push button action by the participant, the speech is W bits size. W is defined by a weibull pdf. */
```

```
/*-------------------------------------------------------------------------------------------------------------------------------*/
event = set_talkburst (WEIB, SCALE, SHAPE)
condition = (TRUE)
action = {
    float q, f_spurt, f_gap, p_spurt, p_gap, m1, m2, m3, talkburst[4];
    get_st (talkburst, "Talkburst[]");
    q = get_cr (size_next_talkburst);

    /* Voice frames and gap frames */
    f_spurt = (q / 0.02) * 0.4;
    f_gap = (q / 0.02) * 0.6;

    /* Voice packets and confort noise packets */
    p_spurt = f_spurt / PTIME;
    p_gap = f_gap / PTIME;

    /* Codifing frames */
    /*m1 = f_spurt * Amr_TX[Amr_Index];*/
    m1 = f_spurt * AMR_TX;
    m2 = f_gap * AMR_SID;

    /* Adding overhead RTP, UDP, IP */
    m3 = (p_spurt + p_gap) * (320 * COMPRESS);

    talkburst[0] = m1 + m2 + m3;
    talkburst[1] = q;
    talkburst[2] = (p_spurt + p_gap);
    talkburst[3] = talkburst[3] + q;
    /**************************************/

    if (DEBUG > 0) {
      fprintf (stdout, "%f: ", get_simul_time());
            fprintf (stdout, "[MS_L] TotalSpeech (%f) This (%f) AMR_TX (%d)\n", talkburst[3], talkburst[1],
       Amr_TX[Amr_Index]);
    }

    set_cr (size_next_talkburst, 0);
    set_st ("Talkburst[]", talkburst);
};
/*-------------------------------------------------------------------*/
}
Messages {
```

```
/*-------------------------------------------------------------------------------------------------------*/
/* Messages from BSS to set MS state (SIGNALING) or to send user spurts */
/* talkspurts codifing in RTP/UDP/IP packets.                                          */
/*-------------------------------------------------------------------------------------------------------*/
msg_rec = RX_BSS
action = {
      int id, data_vec[5], ms_status[MAX_CL], application_buffer[MAX_CL], packets, bursts;
      float delay, ms_burst_time[MAX_CL], ms_block_time[MAX_CL];
      get_msg_data (data_vec);
      get_st (ms_status, "MS_Status[]");
      get_st (application_buffer, "Application_Buffer[]");
      get_st (ms_burst_time, "MS_Burst_Time[]");
      get_st (ms_block_time, "MS_Block_Time[]");
      id = data_vec[ID];
      get_st (packets, "Total_Packets");
      get_st (bursts, "Total_Bursts");
      get_st (delay, "Total_Delay");

      if (data_vec[TYPE] == SIGNALING){
        switch (data_vec[DATA]){
        case FLOOR_TAKEN:
            ms_status[id] = LISTENER;
            ms_burst_time[id] = get_simul_time();
            ms_block_time[id] = data_vec[PIG1];
            packets = packets + data_vec[PIG2];
            if (DEBUG > 2) {
              fprintf (stdout, "%f: ", get_simul_time());
              fprintf (stdout, "[MS_L] MS%d has received FLOOR_TAKEN!\n", id);              }
        break;
        case FLOOR_RELEASING:
            ms_status[id] = WAIT_FLOOR_RELEASE;
            if (DEBUG > 2) {
              fprintf (stdout, "%f: ", get_simul_time());
              fprintf (stdout, "[MS_L] MS%d has received FLOOR_RELEASING!\n", id);
            }
        break;
        case FLOOR_RELEASED:
            ms_status[id] = INACTIVE;
            bursts++;
            delay = delay + (get_simul_time() - ms_burst_time[id]);
            if (DEBUG > 0) {
```

```
          fprintf (stdout, "%f: ", get_simul_time());
          fprintf (stdout, "[MS_L] TDelay (%f) TBursts (%d) TPackets (%d)\n", delay, bursts, packets);
        }
      break;
      }
    }
    else
    if (data_vec[TYPE] == USERDATA) {
       application_buffer[id] = application_buffer[id] + data_vec[DATA];
       if ((DEBUG > 2) & (data_vec[DATA] > 0)) {
         fprintf (stdout, "%f: ", get_simul_time());
         fprintf (stdout, "[MS_L] MS%d has received %d\n", id, application_buffer[id], id);
       }
    }

    set_st ("Total_Delay", delay);
    set_st ("Total_Bursts", bursts);
    set_st ("Total_Packets", packets);
    set_st ("MS_Block_Time[]", ms_block_time);
    set_st ("MS_Burst_Time[]", ms_burst_time);
    set_st ("Application_Buffer[]", application_buffer);
    set_st ("MS_Status[]", ms_status);
};

/*-----------------------------------------------------------------------------------------------*/
/* Message from CH. Set indexes of MCS[] and alter EDGE performance.    */
/*-----------------------------------------------------------------------------------------------*/
msg_rec = RX_CH
action = {
    int id, msg_vec[2], ch_db_status[MAX_CL], amr_index;
    get_msg_data (msg_vec);
    get_st (ch_db_status, "CH_dB_Status[]");
    get_st (amr_index, "Amr_Index");

   id = msg_vec[0];
    ch_db_status[id] = msg_vec[1];
    if (msg_vec[1] < 7) {
      amr_index = 0;
    }
    else {
    if (msg_vec[1] < 12) {
      amr_index = 1;
    }
    else {
      amr_index = 2;
    }
    }

    if (DEBUG > 3) {
       fprintf (stdout, "[MS_L] Change CH%d to dB%d\n", id, msg_vec[1]);
    }

    set_st ("Amr_Index", amr_index);
    set_st ("CH_dB_Status[]", ch_db_status);
};
/*---------------------------------------------------------------------*/
}
Rewards {

rate_reward = size_next_talkburst
condition = (TRUE)
value = 1.0;

}
)
```

/**/

Model PoC, object MS_Left (15.abr.05) by Luis Claudio dos Santos. This object simulates the air channel conditions. It sends messages to MS and BSS to change their channel coding and modulation schemes.

/**/

Object_Desc CH_Left (

```
Declaration {

Const
Port: TX_BSS, TX_MS;
Integer: MAX_CL, BAD, GOOD, dBmenorq8, dB8, dB9, dB10, dB11, dB12, dB13, dB14, dB15, dB16, dB17,
dB18, dB19, dB20, dB21, dB22, dB23, dB24, dB25, dB26, dB27, dB28, dB29, dB30, dB31, dB32, dB33,
dBmaiorq34;
 Float: CHANGE_CH_RATE, CHANGE_GR_RATE, PG1, PG2;
 Object: MS, BSS;

State Var
 Integer: Next_MS_CH, Next_MS_GR, MS_Current_Group[MAX_CL];
 Float: PCI[28];

}
Initialization {

/*CONSTANTS*/
MAX_CL = 02                /*Max numbers of PoC Clients.*/
PG1 = 0.1                  /*Prob to go to group G1.*/
PG2 = 0.1                  /*Prob to go to group G2.*/

/*Rates*/
CHANGE_CH_RATE = 25 /*One change each 40 ms.*/
CHANGE_GR_RATE = 1   /*One change each 1 s.*/

/*Possible Location Group of a MS*/
GOOD = 1
BAD = 2

/*Just the Matrix Indices*/
dBmenorq8 = 0
dB8 = 1
dB9 = 2
… … …
dB32 = 25
dB33 = 26
```

```
dBmaiorq34 = 27

/*Choose one modify change_ch_ms_current events.*/
/*C/I from [3GPP 45]; P[X <= 24dB] = 0.5.*/
/*PCI = [2.151, 0.717, 1.075, 1.075, 1.434, 1.075, 2.867, 2.151, 3.226, 2.867, 3.943, 2.867, 5.018, 3.226,
5.018, 3.943, 5.018, 2.329, 6.273, 3.943, 3.226, 3.943, 3.584, 2.867, 2.509, 2.509, 2.151, 18.996]*/

/*C/I from [LUCE 99]; P[X <= 17dB] = 0.5.*/
PCI = [0.000, 0.000, 1.487, 1.487, 2.974, 3.717, 10.409, 9.665, 7.435, 8.550, 4.276, 8.740, 5.204, 4.461,
3.717, 3.717, 3.717, 2.230, 2.230, 2.230, 1.859, 1.487, 1.487, 1.115, 1.115, 1.115, 0.743, 4.833]

/*C/I from [YALL 02]; P[X <= 16dB] = 0.5.*/
/*PCI = [0.000, 0.000, 0.374, 0.982, 2.760, 6.129, 10.058, 13.193, 13.333, 3.171, 20.127, 9.263, 7.064, 5.567,
4.164, 2.713, 1.099, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000]*/

/*Ports*/
TX_BSS = ch_bss_l
TX_MS = ch_ms_l

/*Objects*/
MS = MS_Left
BSS = BSS_Left

/*VARIABLES*/
Next_MS_CH = 0
Next_MS_GR = 0
MS_Current_Group = [1, 2]

}
Events {
/*-----------------------------------------------------------------------------------------------------------------------------------*/
Each 20 ms, if MS is in G2 group, change among smaller C/I values. The number of actions change according
to PCI[] choosed since PCI[dBmenorq8] + PCI [dB8] + ... + PCI[dB(N)] = 0.5.
/*-----------------------------------------------------------------------------------------------------------------------------------*/
event = change_ch_ms_current_in_G2 (DET, (CHANGE_CH_RATE * MAX_CL))
condition = (MS_Current_Group[Next_MS_CH] == BAD)
action = {
    int next, data_vec[2];
    next = Next_MS_CH;
          data_vec[0] = next;
      data_vec[1] = dBmenorq8;
      msg (TX_BSS, BSS, data_vec);
      msg (TX_MS, MS, data_vec);
      next = next + 1;
      if (next == MAX_CL) next = 0;
    set_st ("Next_MS_CH", next);
} : prob = PCI[dBmenorq8]/50;
{
    int next, data_vec[2];
    next = Next_MS_CH;
      data_vec[0] = next;
      data_vec[1] = dB8;
      msg (TX_BSS, BSS, data_vec);
      msg (TX_MS, MS, data_vec);
      next = next + 1;
      if (next == MAX_CL) next = 0;
    set_st ("Next_MS_CH", next);
} : prob = PCI[dB8]/50;
… … …
{
    int next, data_vec[2];
    next = Next_MS_CH;
      data_vec[0] = next;
      data_vec[1] = dB16;
      msg (TX_BSS, BSS, data_vec);
      msg (TX_MS, MS, data_vec);
      next = next + 1;
      if (next == MAX_CL) next = 0;
    set_st ("Next_MS_CH", next);
} : prob = PCI[dB16]/50;
{
    int next, data_vec[2];
    next = Next_MS_CH;
      data_vec[0] = next;
      data_vec[1] = dB17;
      msg (TX_BSS, BSS, data_vec);
      msg (TX_MS, MS, data_vec);
      next = next + 1;
      if (next == MAX_CL) next = 0;
    set_st ("Next_MS_CH", next);
} : prob = PCI[dB17]/50;

/*-------------------------------------------------------------------------------------------------------------------------------------*/
```

Each 20ms, if MS is in G1 group, change among bigger C/I values. The number of actions change according to PCI[] choosed since PCI[dB(N+1)] + PCI [dB(N+2)] + ... + PCI[dBmaiorq34] = 0.5.

```
/*-----------------------------------------------------------------------------------------------------------------------------------*/
event = change_ch_ms_current_in_G1 (DET, (CHANGE_CH_RATE * MAX_CL))
condition = (MS_Current_Group[Next_MS_CH] == GOOD)
action = {
    int next, data_vec[2];
    next = Next_MS_CH;
          data_vec[0] = next;
      data_vec[1] = dB18;
      msg (TX_BSS, BSS, data_vec);
      msg (TX_MS, MS, data_vec);
      next = next + 1;
      if (next == MAX_CL) next = 0;
    set_st ("Next_MS_CH", next);
} : prob = PCI[dB18]/50;
{
    int next, data_vec[2];
    next = Next_MS_CH;
      data_vec[0] = next;
      data_vec[1] = dB19;
      msg (TX_BSS, BSS, data_vec);
      msg (TX_MS, MS, data_vec);
      next = next + 1;
      if (next == MAX_CL) next = 0;
     set_st ("Next_MS_CH", next);
} : prob = PCI[dB19]/50;
… … …
{
    int next, data_vec[2];
    next = Next_MS_CH;
      data_vec[0] = next;
      data_vec[1] = dB33;
      msg (TX_BSS, BSS, data_vec);
      msg (TX_MS, MS, data_vec);
      next = next + 1;
      if (next == MAX_CL) next = 0;
    set_st ("Next_MS_CH", next);
} : prob = PCI[dB33]/50;
{
    int next, data_vec[2];
    next = Next_MS_CH;
```

```
      data_vec[0] = next;
      data_vec[1] = dBmaiorq34;
      msg (TX_BSS, BSS, data_vec);
      msg (TX_MS, MS, data_vec);
      next = next + 1;
      if (next == MAX_CL) next = 0;
    set_st ("Next_MS_CH", next);
} : prob = PCI[dBmaiorq34]/50;

/*-------------------------------------------------------------------------------------------------------------------------------------------*/
Each 1 sec, if MS is in G1 group, change to G2 group with prob PG2 or stay in G1 group with prob (1-PG2).
/*-------------------------------------------------------------------------------------------------------------------------------------------*/
event = change_ms_in_g1 (DET, (CHANGE_GR_RATE * MAX_CL))
condition = (MS_Current_Group[Next_MS_GR] == GOOD)
action = {
    int next, ms_current_group[MAX_CL];
    get_st (ms_current_group, "MS_Current_Group[]");
    next = Next_MS_CH;

    ms_current_group[next] = BAD;

    next = next + 1;
    if (next == MAX_CL) next = 0;
    set_st ("Next_MS_CH", next);
    set_st ("MS_Current_Group[]", ms_current_group);
} : prob = PG2;
{
    int next, ms_current_group[MAX_CL];
    get_st (ms_current_group, "MS_Current_Group[]");
    next = Next_MS_CH;

    ms_current_group[next] = GOOD;

    next = next + 1;
    if (next == MAX_CL) next = 0;
    set_st ("Next_MS_CH", next);
    set_st ("MS_Current_Group[]", ms_current_group);
} : prob = (1 - PG2);

/*-------------------------------------------------------------------------------------------------------------------------------------------*/
Each 1 sec, if MS is in G2 group, change to G1 group with prob PG1 or stay in G2 group with prob (1-PG1).
/*-------------------------------------------------------------------------------------------------------------------------------------------*/
```

```
event = change_ms_in_g2 (DET, (CHANGE_GR_RATE * MAX_CL))
condition = (MS_Current_Group[Next_MS_GR] == BAD)
action = {
      int next, ms_current_group[MAX_CL];
      get_st (ms_current_group, "MS_Current_Group[]");
      next = Next_MS_CH;

      ms_current_group[next] = GOOD;

      next = next + 1;
      if (next == MAX_CL) next = 0;
      set_st ("Next_MS_CH", next);
      set_st ("MS_Current_Group[]", ms_current_group);
} : prob = PG1;
{
      int next, ms_current_group[MAX_CL];
      get_st (ms_current_group, "MS_Current_Group[]");
      next = Next_MS_CH;

      ms_current_group[next] = BAD;

      next = next + 1;
      if (next == MAX_CL) next = 0;
      set_st ("Next_MS_CH", next);
      set_st ("MS_Current_Group[]", ms_current_group);
} : prob = (1 - PG1);
/*----------------------------------------------------------------------*/
}
Messages {

}
Rewards {}
)
```

```
/*************************************************************************************************************/
Model PoC, object MS_Left (01.nov.05) by Luis Claudio dos Santos. This object simulates BTS and BSC
tasks. The BSS downloads RLC Blocks to MSs according to C/I conditions getting from CH object.
/*************************************************************************************************************/
Object_Desc BSS_Left (
Declaration {

Const
 Port: TX_MS, RX_MS, TX_CH, RX_CH, RX_NET, TX_NET, FLOOR;
          Integer: MAX_CL, MAX_PDTCH, TALKER, LISTENER, INACTIVE, WAITING_RESPONSE,
         WAIT_TBF_ASSIGN, WAIT_FLOOR_RELEASE, WAIT_FLOOR_REQUEST, ID, TYPE, DATA, PIG1,
         PIG2, SIGNALING, USERDATA, FLOOR_TAKEN, FLOOR_RELEASING, FLOOR_RELEASED,
         DEBUG;
 Float: EDGE_RATE, TBF_TIME, REL_TIME;
 Object: MS, CH, CORE;

State Var
         Integer: ToMS_Buffer[MAX_CL], MS_Status[MAX_CL], RX_Window[MAX_CL], MCS[28],
         CH_dB_Status[MAX_CL], MS_Channel[MAX_CL], Next_MS[MAX_PDTCH];
}
Initialization {

/*CONSTANTS*/
/*Max numbers*/
MAX_CL = 02
MAX_PDTCH = 02
DEBUG = 1

/*Rates*/
EDGE_RATE = 50                 /*One RLC Block each 20 ms.*/
TBF_TIME = 0.24                /*Signaling for DL TBF assign lasts 240 ms (aver.).*/
REL_TIME = 0.6                 /*Signaling for rel. floor lasts 600 ms (aver.).*/

/*Possible States of a MS*/
INACTIVE = 0
TALKER = 1
LISTENER = 2
WAITING_RESPONSE = 3
WAIT_TBF_ASSIGN = 4
WAIT_FLOOR_REQUEST = 5
WAIT_FLOOR_RELEASE = 6
```

```
/*Message Indexes*/
ID = 0
TYPE = 1
DATA = 2
PIG1 = 3
PIG2 = 4

/*Type of Signaling Messages*/
SIGNALING = 0
USERDATA = 1

/*Sgnaling Messages*/
FLOOR_TAKEN = 3
FLOOR_RELEASING = 4
FLOOR_RELEASED = 5

/*MCS from [ERIC 99]; iLA.*/
/*MCS = [0, 178, 225, 267, 310, 345, 391, 449, 499, 534, 561, 635, 705, 766, 813, 844, 863, 879, 929, 983,
1037, 1065, 1103, 1130, 1150, 1165, 1173, 1177]*/
/*MCS from [ERIC 99]; LA BLER < 1%.*/
/*MCS = [0, 174, 178, 178, 178, 178, 178, 178, 178, 178, 178, 178, 178, 225, 225, 279, 279, 325, 329, 329,
449, 449, 596, 596, 596, 898, 1173, 1177]*/
/*MCS from [ERIC 99]; LA BLER < 2%.*/
/*MCS = [0, 174, 178, 178, 178, 178, 178, 178, 178, 178, 178, 178, 225, 279, 279, 325, 325, 325, 449, 449,
604, 604, 604, 898, 898, 898, 1177, 1177]*/
/*MCS from [ERIC 99]; iLA BLER < 3%.*/
MCS = [0, 174, 178, 178, 178, 178, 178, 178, 178, 225, 225, 279, 279, 279, 325, 325, 325, 449, 449, 604, 604,
604, 898, 898, 1130, 1146, 1173, 1177]

/*Ports*/
TX_MS = bss_ms_l
RX_MS = ms_bss_l
TX_CH = bss_ch_l
RX_CH = ch_bss_l
TX_NET = bss_net_l
RX_NET = net_bss_l
FLOOR = wire_floor

/*Objects*/
MS = MS_Left
CH = CH_Left
CORE = Core
```

```
/*VARIABLES*/
ToMS_Buffer[] = 0
MS_Status[] = 0
RX_Window[] = 0
CH_dB_Status[] = 13
MS_Channel[] = 0
Next_MS = [0, 1]

}

Events {

/*------------------------------------------------------------------*/
event = tbf_downlink_assign (EXP, (1/TBF_TIME))
condition = (TRUE)
action = {
    int i, msg_vec[3], ms_status[MAX_CL], ms_channel[MAX_CL];
    get_st (ms_status, "MS_Status[]");
    get_st (ms_channel, "MS_Channel[]");

    for (i = 0; i < MAX_CL; i++) {
      if (ms_status[i] == WAIT_TBF_ASSIGN){
        ms_status[i] = LISTENER;
        /*************************************************/
        ms_channel[i] = i - ( ((int)(i / MAX_PDTCH)) * MAX_PDTCH);
        /*************************************************/
        if (DEBUG > 1) {
          fprintf (stdout, "%f: ", get_simul_time());
          fprintf (stdout, "[BSS_L] Activing MS%d!\n", i);
        }
      }
    }

    set_st ("MS_Channel[]", ms_channel);
    set_st ("MS_Status[]", ms_status);
};
/*------------------------------------------------------------------*/

event = download_rlc_blocks (DET, EDGE_RATE)
condition = (TRUE)
action = {
      int i, aux, ms_id, msg_vec[3], ms_status[MAX_CL], rx_window[MAX_CL], toms_buffer[MAX_CL],
      next_ms[MAX_PDTCH];
    aux = 0;

    get_st (ms_status, "MS_Status[]");
    get_st (rx_window, "RX_Window[]");
    get_st (toms_buffer, "ToMS_Buffer[]");
    get_st (next_ms, "Next_MS[]");

    for (i = 0; i < MAX_PDTCH; i++) {
      ms_id = next_ms[i];
      aux = CH_dB_Status[ms_id];
      if (ms_status[ms_id] == LISTENER) {
        if (toms_buffer[ms_id] > MCS[aux]){
           msg_vec[ID] = ms_id;
           msg_vec[TYPE] = USERDATA;
           msg_vec[DATA] = MCS[aux];
           msg (TX_MS, MS, msg_vec);
           rx_window[ms_id] = rx_window[ms_id] - MCS[aux];
           toms_buffer[ms_id] = toms_buffer[ms_id] - MCS[aux];
           if (DEBUG > 1) {
             fprintf (stdout, "%f: ", get_simul_time());
                       fprintf (stdout, "[BSS_L] Downloading %d to MS%d in CH%d!\n", msg_vec[DATA],
        ms_id, i);
           }
        }
        else {
        if (toms_buffer[ms_id] > 0) {
           msg_vec[ID] = ms_id;
           msg_vec[TYPE] = USERDATA;
           msg_vec[DATA] = toms_buffer[ms_id];
           msg (TX_MS, MS, msg_vec);
           if (DEBUG > 1) {
             fprintf (stdout, "%f: ", get_simul_time());
                       fprintf (stdout, "[BSS_L] Downloading %d to MS%d in CH%d!\n", msg_vec[DATA],
        ms_id, i);
           }
           rx_window[ms_id] = rx_window[ms_id] - toms_buffer[ms_id];
           toms_buffer[ms_id] = 0;
        }
        }
        if (rx_window[ms_id] == 0) {
           ms_status[ms_id] = WAIT_FLOOR_RELEASE;
           msg_vec[ID] = ms_id;
           msg_vec[TYPE] = SIGNALING;
           msg_vec[DATA] = FLOOR_RELEASING;
           msg (TX_MS, MS, msg_vec);
           if (DEBUG > 1) {
             fprintf (stdout,"%f: ", get_simul_time());
             fprintf (stdout, "[BSS_L] ### Finished BURST MS%d in CH%d!\n", ms_id, i);
           }
         }
       }
       /*** Set next MS to serve in channel i ***/
       next_ms[i] = next_ms[i] + MAX_PDTCH;
       if (next_ms[i] >= MAX_CL) next_ms[i] = i;
       /***************************************/
    }

    set_st ("Next_MS[]", next_ms);
    set_st ("ToMS_Buffer[]", toms_buffer);
    set_st ("RX_Window[]", rx_window);
    set_st ("MS_Status[]", ms_status);
};

/*------------------------------------------------------------------*/
event = floor_release (EXP, (1/REL_TIME))
condition = (TRUE)
action = {
    int i, msg_vec[3], ms_status[MAX_CL];

    get_st (ms_status, "MS_Status[]");

    for (i = 0; i < MAX_CL; i++) {
      if (ms_status[i] == WAIT_FLOOR_RELEASE){
        ms_status[i] = INACTIVE;
        msg_vec[ID] = i;
        msg_vec[TYPE] = SIGNALING;
        msg_vec[DATA] = FLOOR_RELEASED;
        msg (TX_MS, MS, msg_vec);
        if (DEBUG > 1) {
          fprintf (stdout, "%f: ", get_simul_time());
          fprintf (stdout, "[BSS_L] MS%d releasing floor!\n", i);
        }
      }
    }
    set_st ("MS_Status[]", ms_status);
};
/*--------------------------------------------------------*/
}

Messages {
/*-------------------------------------------------------------------------------------------------*/
/* Message from CORE with MS data. Increase buffer size to send data in                            */
/* donwlink through downlink_rlc_blocks event.                                                     */
/*-------------------------------------------------------------------------------------------------*/
msg_rec = RX_NET
action = {
    int id, aux, msg_vec[3], toms_buffer[MAX_CL];
    get_msg_data (msg_vec);
    get_st (toms_buffer, "ToMS_Buffer[]");
    id = msg_vec[ID];
    toms_buffer[id] = toms_buffer[id] + msg_vec[DATA];
    aux = msg_vec[DATA];
    if (DEBUG > 1) {
      fprintf (stdout, "%f: ", get_simul_time());
      fprintf (stdout, "[BSS_L] More %d to MS%d\n", aux, id);
    }
    set_st("ToMS_Buffer[]", toms_buffer);
};

/*----------------------------------------------------------------------------------*/
/* Message from MS. Receive data from MS and send it to CORE processing.   */
/*----------------------------------------------------------------------------------*/
msg_rec = RX_MS
action = {
    int ms_id, msg_vec[3], ms_status[MAX_CL];

    get_msg_data (msg_vec);
    get_st (ms_status, "MS_Status[]");

    ms_id = msg_vec[ID];
    if (msg_vec[TYPE] == USERDATA) {
      msg (TX_NET, CORE, msg_vec);
    }
    set_st ("MS_Status[]", ms_status);
};

/*----------------------------------------------------------------------------------------------------*/
/* Message from CH. Set indexes from MCS[] and alter EDGE performance.   */
/*----------------------------------------------------------------------------------------------------*/
msg_rec = RX_CH
action = {
    int id, msg_vec[2], ch_db_status[MAX_CL];

    get_msg_data (msg_vec);
    get_st (ch_db_status, "CH_dB_Status[]");

    id = msg_vec[0];
    ch_db_status[id] = msg_vec[1];
    if (DEBUG > 1) {
      fprintf (stdout, "%f: ", get_simul_time());
      fprintf (stdout, "[BSS_L] Change CH%d to dB%d\n", id, msg_vec[1]);
    }

    set_st ("CH_dB_Status[]", ch_db_status);
};

/*------------------------------------------------------------------*/
/* Messages for floor signaling.                                   */
/*------------------------------------------------------------------*/
msg_rec = FLOOR
action = {
    int id, data, data_vec[5], ms_status[MAX_CL], rx_window[MAX_CL];

    get_msg_data (data_vec);
    get_st (ms_status, "MS_Status[]");
    get_st (rx_window, "RX_Window[]");

    id = data_vec[ID];
    data = data_vec[DATA];
    /*if (MAX_BUFFER - rx_window[id] >= data_vec[DATA]){*/
       rx_window[id] = rx_window[id] + data_vec[DATA];
       ms_status[id] = WAIT_TBF_ASSIGN;
       data_vec[ID] = data_vec[ID];
       data_vec[TYPE] = SIGNALING;
       data_vec[DATA] = FLOOR_TAKEN;
```

```
            data_vec[PIG2] = data_vec[PIG2];
            msg (TX_MS, MS, data_vec);
            if (DEBUG > 1) {
               printf ("[BSS_L] >>>MS%d will be RECEIVING %d\n", id, rx_window[id]);
            }
       /*}*/

       set_st("RX_Window[]", rx_window);
       set_st("MS_Status[]", ms_status);
};
/*---------------------------------------------------------------------*/
}
Rewards {}
)
```

```
/***********************************************************************************************************************/
Model PoC, object MS_Left (01.nov.05) by Luis Claudio dos Santos. This object handles packets between
BSSs and simulates processing in routers, PoC server, etc. The delay is exp modeled (mean = 280ms).
/***********************************************************************************************************************/
Object_Desc Core (
Declaration {

Const
 Integer: MAX_CL, ID, TYPE, DATA, SIGNALING, USERDATA, DEBUG;
 Port: TX_LEFT, TX_RIGHT, RX_LEFT, RX_RIGHT;
 Object: BSS_LEFT, BSS_RIGHT;
 Float: PROC_RATE;

State Var
 Integer: ToMS_R_Buffer[MAX_CL], ToMS_L_Buffer[MAX_CL];
}

Initialization {
/*CONSTANTS*/
/*Parameters of Simulations*/
MAX_CL = 02      /*Max number of clients per sector.*/
DEBUG = 1        /*Level of printed debug info.*/

/*Rates*/
PROC_RATE = 0.24 /*Expected 280 ms.*/

/*Message Indexes*/
ID = 0
TYPE = 1
DATA = 2

/*Type of Signaling Messages*/
SIGNALING = 0
USERDATA = 1

/*Ports*/
TX_LEFT = net_bss_l
RX_LEFT = bss_net_l
TX_RIGHT = net_bss_r
RX_RIGHT = bss_net_r

/*Objects*/
```

```
BSS_LEFT = BSS_Left
BSS_RIGHT = BSS_Right

/*VARIABLES*/
ToMS_L_Buffer[] = 0
ToMS_R_Buffer[] = 0
}

Events {
/*--------------------------------------------------------------------------------------------------------------------------------*/
Simulates the processing of information (data and signaling) in GWs, routers, PoC servers, etc from RIGHT to
LEFT of model. Delay expected is PROC_RATE seconds.
/*--------------------------------------------------------------------------------------------------------------------------------*/
event = procpacks_to_leftside (EXP, (MAX_CL/PROC_RATE))
condition = (TRUE)
action = {
    int i, msg_vec[3], toms_l_buffer[MAX_CL];
    get_st (toms_l_buffer, "ToMS_L_Buffer[]");

    for (i = 0; i < MAX_CL; i++) {
      msg_vec[ID] = i;
      msg_vec[TYPE] = USERDATA;
      msg_vec[DATA] = toms_l_buffer[i];
      msg (TX_LEFT, BSS_LEFT, msg_vec);
      toms_l_buffer[i] = 0;
      if (DEBUG > 1) {
        fprintf (stdout, "[CORE] Sending %d to LEFT (MS%d)!\n", msg_vec[DATA], i);
      }
    }

    set_st ("ToMS_L_Buffer[]", toms_l_buffer);
};

/*--------------------------------------------------------------------------------------------------------------------------------*/
Simulates the processing of information (data and signaling) in GWs, routers, PoC servers, etc from LEFT to
RIGHT of model. Delay expected is PROC_RATE seconds.
/*--------------------------------------------------------------------------------------------------------------------------------*/
event = procpacks_to_rightside (EXP, (MAX_CL/PROC_RATE))
condition = (TRUE)
action = {
    int i, msg_vec[3], toms_r_buffer[MAX_CL];
    get_st (toms_r_buffer, "ToMS_R_Buffer[]");
    for (i = 0; i < MAX_CL; i++) {
      msg_vec[ID] = i;
      msg_vec[TYPE] = USERDATA;
      msg_vec[DATA] = toms_r_buffer[i];
      msg (TX_RIGHT, BSS_RIGHT, msg_vec);
      toms_r_buffer[i] = 0;
      if (DEBUG > 1) {
        fprintf (stdout, "[CORE] Sending %d to LEFT (MS%d)!\n", msg_vec[DATA], i);
      }
    }

    set_st ("ToMS_R_Buffer[]", toms_r_buffer);
};
/*----------------------------------------------------------------------*/
}

Messages {
/*------------------------------------------------------------------------------------------*/
/* Receive data from RIGHT side and increase buffer size to processing. */
/*------------------------------------------------------------------------------------------*/
msg_rec = RX_RIGHT
action = {
    int i, msg_vec[3], toms_l_buffer[MAX_CL];
    get_msg_data (msg_vec);
    get_st (toms_l_buffer, "ToMS_L_Buffer[]");

    i = msg_vec[ID];
    toms_l_buffer[i] = toms_l_buffer[i] + msg_vec[DATA];

    set_st ("ToMS_L_Buffer[]", toms_l_buffer);
};

/*------------------------------------------------------------------------------------------*/
/* Receive data from LEFT side and increase buffer size to processing.   */
/*-------------------------------------------------------------------------------------------*/
msg_rec = RX_LEFT
action = {
    int i, msg_vec[3], toms_r_buffer[MAX_CL];
    get_msg_data (msg_vec);
    get_st (toms_r_buffer, "ToMS_R_Buffer[]");
    i = msg_vec[ID];
    toms_r_buffer[i] = toms_r_buffer[i] + msg_vec[DATA];

    set_st ("ToMS_R_Buffer[]", toms_r_buffer);
};
/*---------------------------------------------------------------------*/
}
Rewards {

}
)
```