January 1995

Implementation Notes to OMEGA Architecture

Klara Nahrstedt
University of Pennsylvania

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-95-28.

Implementation Notes to OMEGA Architecture

Abstract

The OMEGA architecture is an end-point architecture for provision of end-to-end QoS guarantees. The architecture principles, design, high-level implementation issues, and telerobotics experimental setup for validation of OMEGA concepts are described in my thesis (MS-CIS-95-31). In this document, I will concentrate on the structure of the software modules, their naming and content. The complete code is part of this document. There are approximately 10,000 lines of code.

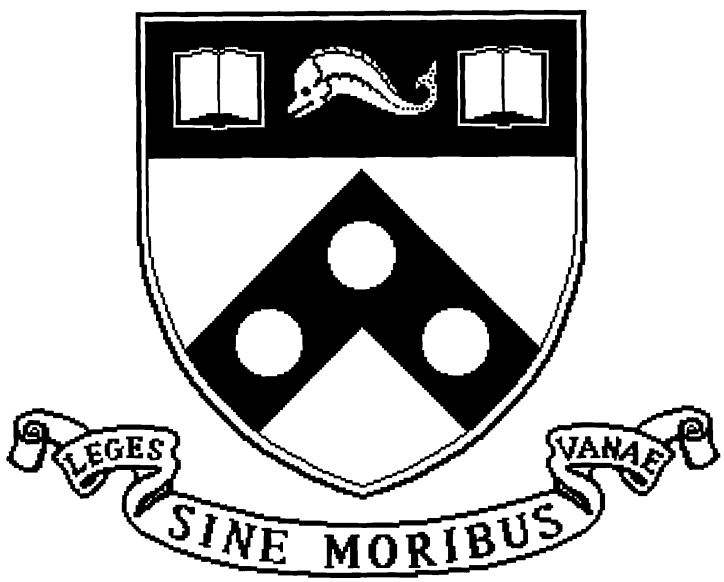
Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-95-28.

Implementation Notes to OMEGA Architecture

MS-CIS-95-28

Klara Nahrstedt [1]



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

1995

[1]This work was supported by the National Science Foundation and the Advanced Research Projects Agency under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives. Additional support was provided by Bell Communications Research under Project DAWN, by an IBM Faculty Development Award, and by Hewlett-Packard

Implementation Notes to OMEGA Architecture

Klara Nahrstedt*
University of Pennsylvania
e-mail: klara@aurora.cis.upenn.edu

1 OMEGA Experimental Platform

The OMEGA architecture is an end-point architecture for provision of end-to-end QoS guarantees. The architecture principles, design, high-level implementation issues, and telerobotics experimental setup for validation of OMEGA concepts are described in my thesis [1].

In this document, I will concentrate on the structure of the software modules, their naming and content. The complete code is part of this document. There are approximately 10,000 lines of code.

1.1 IBM RS/6000

The architecture was implemented on IBM RS/6000 machines, and is running on Models 530 (master side) and 360 (slave side). The implementation language is C. The implementation of OMEGA uses the real-time services of the AIX operating system [2], particularly it relies on

- the fine granularity of the timers through the functions *gettimeofday*, *usleep* and *ualarm*,
- possibility to set real-time priorities in the OS scheduler using *setpri* and use the OS fixed-priority scheduling policy. For OMEGA it means that the current processes of OMEGA run with the priority 0 (it must be a priority smaller than 16), therefore they are not pre-empted by the AIX scheduler and don't undergo the recalculation of priority by the OS scheduler (OS scheduler has priority 16).
- possibility to pin code and data in the user space *pincode, pinu*. This mean that the pinned code and data reside in the real memory, are not swapped out/in to/from the disc and don't underly the virtual memory mechanisms.

All these services help to provide more deterministic behavior when processing/moving real-time data.

*This work was supported by the National Science Foundation and the Advanced Research Projects Agency under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives. Additional support was provided by Bell Communications Research under Project DAWN, by an IBM Faculty Development Award, and by Hewlett-Packard.

1.2 ATM LAN

The two IBM RS/6000 are connected through a dedicated ATM link. I use ATM host interfaces which provide a bandwidth of 133 Mbps to the application through their ATM drivers. The driver provides an access to two types of connections: (1) the connections, where datagrams can be sent, use the AAL3/4 CS PDUs, and (2) the connections, where cells can be sent, use the null AAL layers, it means raw ATM layer. The size of the datagrams can be up to 64 KB, the size of the cells is 44 bytes. The driver interface includes functions to access ATM device services such as `open(/*parameters*/)` to open the ATM device, `read(/*parameters */)` to read PDUs (datagrams/cells) from the ATM device, `write(/*parameters*/)` to write PDUs. The different classes of transmission (datagram/cell) are distinguished through assignment of VCI numbers. The raw ATM layer is accessed if VCI, smaller that 0x4000, are allocates. The AAL3/4 layers is accessed if VCIs larger that 0x4000 are assigned. I divide the classes in OMEGA by using parameters `DATAGRAM-MODE`, or `CELL-MODE`. The ATM driver functions are hidden by using the RTNP(Real-Time Network Protocol) functions:

- *connect_s/connect_r* to open simplex ATM connections with the proper PDU. These functions use two routines: *init_send_atm* to open the ATM device and set ioctl parameters for simplex connection in sending direction, *init_recv_atm* to open the ATM device and set ioct parameters for simplex connection in receiving direction,
- *send_cell, recv_cell* to send/receive PDUs of size 44 bytes or less;
- *send_pkt, recv_pkt* to send/receive PDUs of size 64 KB or less.

These functions might be reimplemented when a new ATM card (e.g., commercial ATM Fore card) and driver replace our experimental ATM card.

2 Software Organization in Directories

The Figure 1 shows the tree structure of the directories where the OMEGA software resides. The software can be found in ftp/pub/dsl/Klara_Nahrstedt/tele.d directory. Figure 2 shows the OMEGA related files in the directories, the Figure 3 shows the application related files in the directories. I briefly describe the content of the most important files which are used in the implementation of OMEGA. For a more detailed description, the comments in the code should help.

- *QoS_management.d*

  This directory includes the QoS Broker protocols (`QoSBroker` routine), such as broker-buyer (`QoSBroker_Buyer` routine) and broker-seller (`QoSBroker_Seller`). The highest level of protocols is implemented in `QoSbroker.c`. `QoSkernel.c` includes all the services a broker needs to make end-to-end guarantees decisions. It means that there are the admission services (`admitAppQoS, admitNetQoS` routines), negotiation services (`negotiateAppQoS, negotiateNetQoS` routines), the QoS translator (`QoSTranslator`), and system routines for task management (`setTaskParam, getTaskParam` routines and others). In the current implementation, the tuning service `tuneQoS.c` is outside of the service kernel because it is used to translate between perceptual and application QoS and is embedded in GUI (Graphical User Interface) which resides above the broker.

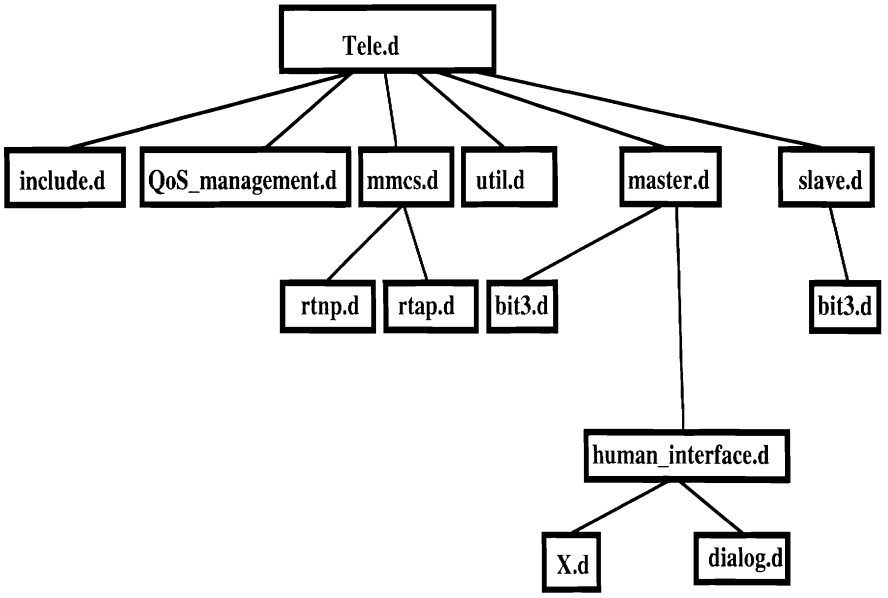


Figure 1: *Software organization in directories.*

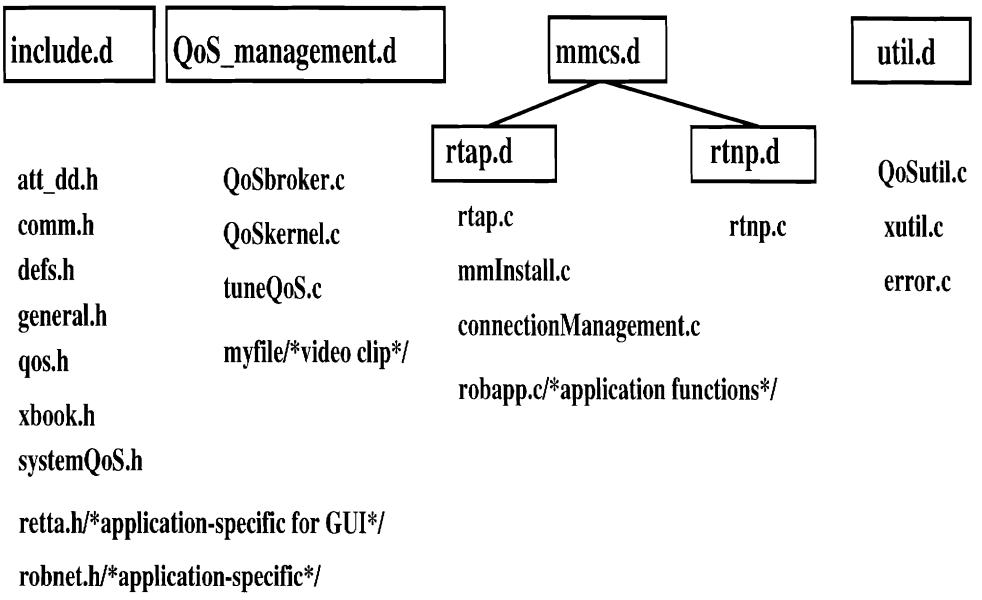


Figure 2: *Files of OMEGA.*

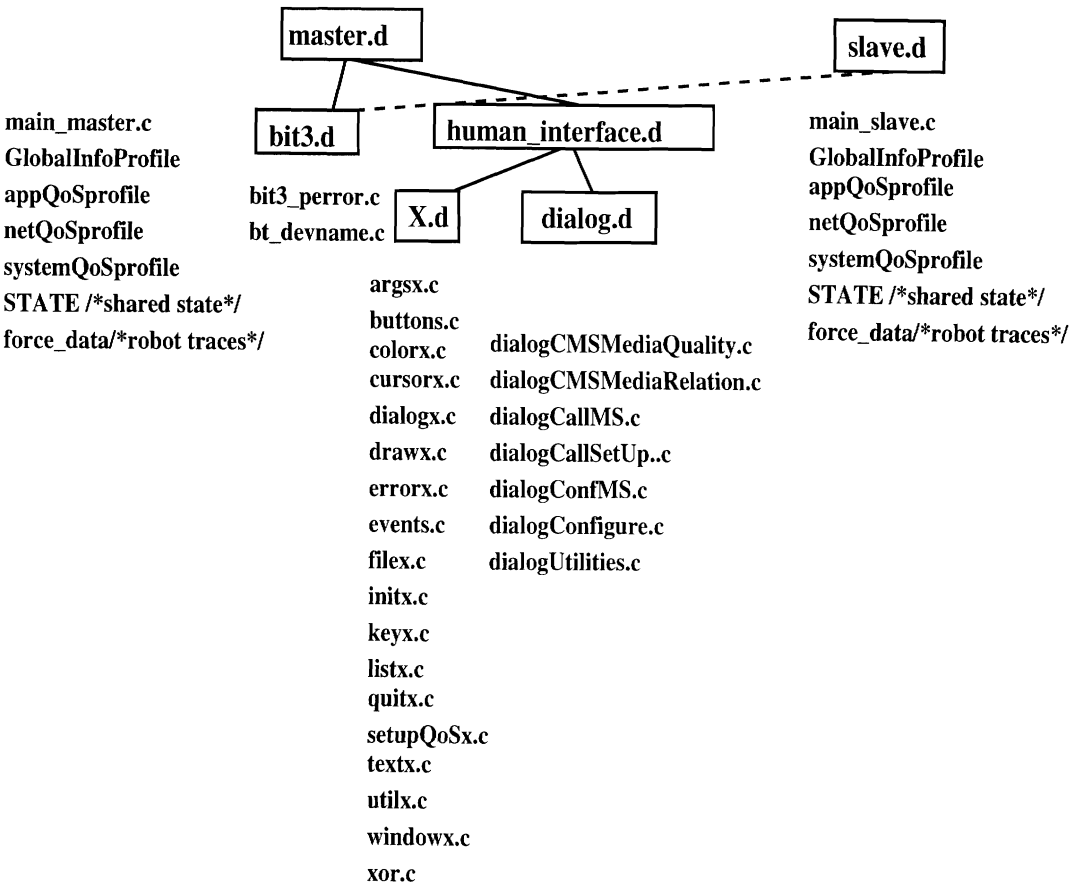


Figure 3: *Files related to telerobotics.*

- *include.d*

  This directory includes all `.h` files OMEGA and the telerobotics application uses. The include files are separated by semantics:

  - `qos.h` includes definitions of all application and network QoS parameter structures.
  - `systemQoS.h` includes definitions of the scheduler, tasks which characterize the system QoS and other system information related to tasks and scheduling.
  - `rtnp.h` includes the ATM/RTNP constants such as the supported cell size, datagram size, VCIs for negotiation, renegotiation and forward error correction.
  - `comm.h` includes all data structures/constants for communication, such as PDUs for negotiation response, signaling, network cell/datagram structure, video packet structure, forward error correction structure.

- *mmcs.d*

  This directory includes two other subdirectories: `rtnp.d` includes in `rtnp.c` the RTNP functions (`connect_s`, `connect_r`, `send_cell`, `send_pkt`. `recv_cell`, `recv_pkt`). The other directory `rtap.d` includes:

  - `connectionManagement.c` includes procedure (`connSetup`) for opening connections (using `connect_s and connect_r`) for one multimedia call based on VCI assignment in the network QoS profile from the broker,
  - `rtap.c` include the RTAP functions and RTAP/RTNP functions plugged into the scheduler. In a full-scaled implementation this file might be split, a more general set of RTAP functions developed and the scheduler needs to access the functions in a more general sense, similar to the OS scheduler. It means there is a need to find a more general application programming interface to plug in application tasks.
  - `robapp.c` includes functions to read robotics data from robotics traces.

- *util.d*

  `QoSutil.c` includes utility functions, such as processing of least common multiplier (`lcm`; translation between different representations: `transRate` for translation from application to network rate and vice versa (it is used in `QoSTranslator`), and `transInterDelay` for translating delay from samples/second in second range (application input) to millisecond range (network input); help functions such as `getThroughput` to compute aggregate bandwidth, `getproctime` to get a time duration ($t_2 - t_1$); and other functions to release resources for a medium and connection.

- *master.d*

  In this directory the profiles are stored where important shared information are stored (`appQoSprofile, netQoSprofile, GlobalInfoProfile,systemQoSprofile`). They are exchanged between the broker and the scheduler of RTAP/RTNP tasks. The `main_master.c` initializes all the profiles and calls the GUI.

  From the master directory the `human_interface.d` directory includes the X files in `X.d` and the GUI dialog files in `dialog.d`. The GUI is very much telerobotics oriented. This user

interface could lead to a more general interface to register (specify) stream QoS. This is future work. GUI is implemented in hierarchical layers. The main (highest control layer) menu control is implemented in `X.d/buttonx.c` file, routine `pressButton`. Here, the event loop controls the user interface buttons *EXIT, QoSConfig, CallSetUp, START, STOP, Help.* Under each button, different functions are called to provide the functionality. For example, under *QoSConfig* the routines in `dialog.d` directory are called to provide a GUI for the configuration of a multimedia call. Under *CallSetUp*, partially the GUI routines from `dialog.d` directory are called, but at some point of the menu hierarchy the broker from `QoS_management.d` directory is called to make resource admissions and allocations. The menu button *START* accesses the routines in `rtap.d` directory to start the RTAP/RTNP scheduler in `rtap.c` file. The *STOP* menu button accesses the routines in `rtap.d/rtap.c` file to stop the scheduler. It is important to stress that the menu events influence the events at the slave side as well, hence the information about the GUI events must be propagated to the slave, so that the slave side knows in which state the GUI (master side) is. For this task, the application negotiation connections are used.

- *slave.d*

  `slave.d` includes the `main_slave.c` file and the shared profiles (the same profile naming as in `master.d` directory) used in for exchange of information between the broker and the scheduler. They represent the contract between the broker and the scheduler, the same way as the master case. The `main_slave.c` file includes the high level menu control loop similar to the GUI in `buttonx.c` file to be consistent with the master side and follow the activity on the master side.

3 User Interfaces of OMEGA

Using the `makefile` in the master.d and slave.d directories, the executable code is `telemaster` in `master.d` directory for master side and `teleslave` in `slave.d` directory for the slave side. Telemaster provides a GUI for the human operator to enter the application QoS for the master robot side and the slave robot side. It means that the operator configures remotely the slave as well.

3.1 Influence of GUI Menu Control on other System Components

The GUI is menu driven. The operator must set up the QoS characteristics for the multimedia call in each direction sending and receiving (INPUT/OUTPUT) and parameterize application QoS for both sides sender and receiver (MASTER/SLAVE). The teleslave does not have a GUI. At the beginning (before any connection is established) it immediately calls the broker and waits for the application QoS on the application negotiation connection. On the second run, the main application program waits on the application negotiation connection for a menu control. The reason is that the operator (remote application) decides either to run only one stream and then immediately to start with transmission, or the establish another connection for a new stream. Hence, the application negotiation connection is not only used to exchange the application QoS parameters during the application negotiation, but also for exchange of menu control commands, so that the broker at the SLAVE side knows in which state the MASTER is. This menu control is propagated to the

scheduler state too. It means that the renegotiation task recognizes the menu state at the SLAVE side, and can react if the MASTER decides to quit, or just to stop the transmission.

3.2 Usage of GUI

The user/operator specifies through GUI the QoS for a multimedia call in one direction `QoSConfig`, and goes directly to `CallSetUp` menu. Here the broker is called to establish the resource deal between the BUYER and the SELLER. When setting up the QoS, both (MASTER and SLAVE) QoS must be configured. The broker uses both, master QoS to setup its own databases and the slave QoS to send to the SLAVE and configure the slave QoS databases. After one multimedia call is set up, then the user can go back to `QoSConfig` menu and setup QoS for the opposite multimedia call. After this is done, the *CallSetUp* menu is chosen and a brokerage performed on this stream. When all calls are registered with the broker, the *START* button is pushed and the transmission starts. The *STOP* button is pressed to stop the transmission, but not to release the resources. What it means is that the contract still exists and the user by pushing the *START* button can continue the transmission. Only the *EXIT* button will release the negotiated deal. During the transmission, the user can press the *CallSetUp* button and initiate renegotiation by pressing *MOD/Param* button in the establishment *CallSetUp* menu. Here, the possibility is to decrease the quality of a video rate, for example, from 5 frames/second to 1 frame/second. I don't support currently increase of a video rate.

4 Conclusion

OMEGA implementation is a prototype of the OMEGA architecture. The goal of the implementation was to prove a concept of OMEGA, i.e., to show that when application QoS are specified to the broker, the broker provides the right answer about the resource availability (using the services such as translation, admission and negotiation in an integrated fashion) and the scheduler provides the timing guarantees. A full-scaled implementation is future work.

During the prototype implementation, many limitations of the chosen platform and my own choices in the implementation became clear. Both types of limitations are discussed in my thesis. Hence, the implementor must be aware of these choices, he/she makes with respect to the chosen platform (workstations, multimedia devices, real-time OS support) as well as his/her own implementation choices (constants, data structures, system principles, underlying system services). Many issues, although designed in OMEGA architecture, need future work to map these design issues to a real system environment.

References

[1] K. Nahrstedt, "An Architecture for End-to-End Quality of Service Provision and its Experimental Validation", *PhD Thesis*, July 1995

[2] IBM Corporation, "AIX Version 3.1: RISC System/6000 as a Real-Time System", IBM International Technical Support Center, Austin, March, 1991

```
/*************************************************************************/
/* Filename: QoSBroker.c                                                 */
/* Purpose : QoS management protocol for QoS set-up                      */
/* Author : Klara Nahrstedt                                              */
/* Update : 6/29/95                                                     */
/*************************************************************************/

#include "/home/klara/tele.d/include.d/defs.h"
#include "/home/klara/tele.d/include.d/retta.h"
#include "/home/klara/tele.d/include.d/comm.h"
#include "/home/klara/tele.d/include.d/systemQoS.h"

int QoSBroker(Param,Add_Param,Notification,side,inout,state)
APP_QOS *Param;
ADD_INFO *Add_Param;
NOTIFY *Notification;
int side; /* Spec of the initiator (BUYER/SELLER) */
int inout;
int state;
{
  struct timeval tv1,tv2;
  struct timezone tz;
  long clock;

  setAppQoS(Param,inout);
  switch(side)
    {
    case BUYER:
      gettimeofday(&tv1,&tz);
      if (QoSBroker_Buyer(Add_Param,Notification,inout) == BAD_VALUE)
        {
          return(BAD_VALUE);
        }
      if (Add_Param->info[GET_IMAGE].done == TRUE)
        {
          return(0);
        }
      gettimeofday(&tv2,&tz);
      getproctime(tv1,tv2,&clock);
      printf("CALL ESTABLISHMENT = %d microsecond \n", clock);
      break;
    case SELLER:
      if (QoSBroker_Seller(Add_Param,Notification,inout) == BAD_VALUE)
        {
          return(BAD_VALUE);
        }
      if (Add_Param->info[GET_IMAGE].done == TRUE)
        {
          return(0);
        }
      break;
    default:
      break;
    }
}
/*************************************************************************/
/* BROKER - BUYER PROTOCOL                                               */
/*************************************************************************/

int QoSBroker_Buyer(Add_Param, Notification,inout)
ADD_INFO *Add_Param;
NOTIFY *Notification;
int inout;
{
  int vci1;
  int conid1;
  FEC_FLAGS err;
  NEG_RESPONSE response;
  int result;
  APP_QOS AParam;
  NET_QOS_TABLE NParam ;
  struct timeval tv1,tv2;
  struct timezone tz;
  long runtime;
  GLOBAL_STATE SystemState;
  RATE_MONOTONIC_SCHEDULER rms;
  INFO_STATE WhatInfo;

/********************** GET APPLICATION QOS ***********************************/

  getAppQoS(&AParam,inout);

/******************* ADMIT APPLICATION QOS ********************************/

  if (Add_Param->info[GET_IMAGE].done == FALSE)
    {
      gettimeofday(&tv1,&tz);
      if ((result = AdmitAppQoS(&AParam, Notification,inout,BUYER)) == BAD_VALUE)
        {
          perror("AdmitAppQoS: not admitted ");
        }
      gettimeofday(&tv2,&tz);
      getproctime(tv1,tv2,&runtime);
      printf(" AdmitApp QoS runtime = %d usec \n",runtime);
    }
  else
    {
      Notification->note = NEG_SUCCESS;
    }
  switch(Notification->note)
    {
    case NEG_SUCCESS:
/********************** NEGOTIATE APPLICATION QoS WITH REMOTE SITE *************/
      gettimeofday(&tv1,&tz);
      negotiateAppQoS(&AParam,Add_Param,Notification,BUYER,&inout);
      if (Add_Param->info[GET_IMAGE].done == TRUE)
        {
          return(0);
        }
      gettimeofday(&tv2,&tz);
      getproctime(tv1,tv2,&runtime);
      printf(" Negotiate App QoS runtime = %d usec \n",runtime);
      break;
    case NEG_MODIFY:
    case NEG_REJECT:
      /* send to Seller a message about admission problems */
      WhatInfo.i_set[0] = SystemStateInfo;
      WhatInfo.i_set[1] = NOT_SPECIFIED;
      RetrieveGlobalState(&SystemState,&rms,WhatInfo);
      if (SystemState.net.Aneg_in.status == FREE)
        {
          vci1 = APP_SIGNAL1_VCI;
          connect_s(vci1,&conid1,DATAGRAM_MODE,sizeof(APP_QOS));
          SystemState.net.Aneg_in.status = TAKEN;
          SystemState.net.Aneg_in.id = conid1;
        }
      response.result=REJECT;
      response.reason = Notification->reason;
```

```
      err.err_flag = FALSE;
      if ((result = send_pkt(SystemState.net.Aneg_in.id,&response,
                             sizeof(NEG_RESPONSE),err))== WRONG_SIZE)
        {
          perror("negotiateAppQoS datagram too small for response transmission");
          exit(1);
        }

      /************ deallocate resources SystemState, Scheduler
       is done in the admission procedure******/
      return(BAD_VALUE);
      break;
    }


/********************** TRANSLATE APPLICATION QOS TO NETWORK QoS ***************/

  switch(Notification->note)
    {
    case NEG_SUCCESS:
      gettimeofday(&tv1,&tz);
      QoSTranslator(APP_TO_NET,BUYER,inout);
      gettimeofday(&tv2,&tz);
      getproctime(tv1,tv2,&runtime);
      printf("  QoS Translator  runtime = %d usec \n",runtime);
      break;
    case NEG_MODIFY:
    case NEG_REJECT:
      /************ deallocate resources
       is done in the negotiation procedure ******/
      return(BAD_VALUE);
      break;
    }
/********************** GET NETWORK QOS ***************************************/

  getNetQoS(&NParam,inout);
  printf("QoSBrokerBuyer: NParam.connection[0].load.end_to_end_delay=%f \n",
         NParam.connection[0].load.end_to_end_delay);
/********************** ADMIT NETWORK QOS *************************************/
  gettimeofday(&tv1,&tz);
  if ((result = AdmitNetQoS(&NParam, Notification,inout,BUYER)) == BAD_VALUE)
    {
      perror("AdmitNetQoS: not admitted ");
    }
  gettimeofday(&tv2,&tz);
  getproctime(tv1,tv2,&runtime);
  printf(" Admit Net QoS runtime = %d usec \n",runtime);
/********************** NEGOTIATE NETWORK QOS *********************************/
  switch(Notification->note)
    {
    case NEG_SUCCESS:
      gettimeofday(&tv1,&tz);
      negotiateNetQoS(&NParam,Notification,BUYER,inout);
      gettimeofday(&tv2,&tz);
      getproctime(tv1,tv2,&runtime);
      printf(" Negotiate Net  QoS runtime = %d usec \n",runtime);
      break;
    case NEG_MODIFY:
    case NEG_REJECT:
      /************ deallocate resources Scheduler
       is done in the admission procedure ******/
      return(BAD_VALUE);
      break;
    }
  switch(Notification->note)
    {
    case NEG_SUCCESS:
      QoSTranslator(NET_TO_APP,BUYER,inout);
      break;
    case NEG_MODIFY:
    case NEG_REJECT:
      /************ deallocate resources
       is done in the negotiation procedure ******/
      return(BAD_VALUE);
      break;
    }
}

int QoSBroker_Seller(Add_Param, Notification,inout)
ADD_INFO *Add_Param;
NOTIFY *Notification;
int inout;
{
  APP_QOS AParam;
  NET_QOS_TABLE NParam;

  struct timeval tv1,tv2;
  struct timezone tz;
  long clock;

  int Acid1,Acid2;
  int Ncid1,Ncid2;
/********************** NEGOTIATE APPLICATION QOS *****************************/

  negotiateAppQoS(&AParam,Add_Param,Notification,SELLER,&inout);
  printf("BROKER/SELLER: after negotiation \n");
  if (Add_Param->info[GET_IMAGE].done == TRUE)
    {
      return(0);
    }
/********************** NEGOTIATE NETWORK QOS *********************************/
  printf("BROKER/SELLER: before network negotiation \n");
  switch(Notification->note)
    {
    case NEG_SUCCESS:
      printf("BROKER/SELLER: negotiation success \n");
      gettimeofday(&tv1,&tz);
      negotiateNetQoS(&NParam,Notification,SELLER,inout);
      gettimeofday(&tv2,&tz);
      getproctime(tv1,tv2,&clock);
      printf("negotiateNetQoS = %d microsecond \n", clock);

      break;
    case NEG_MODIFY:
    case NEG_REJECT:
      /************ deallocate resources
       is done in the admission procedure ******/
      return(BAD_VALUE);
      break;
    }
  switch(Notification->note)
    {
    case NEG_SUCCESS:
      QoSTranslator(NET_TO_APP,SELLER,inout);
      break;
    case NEG_MODIFY:
    case NEG_REJECT:
      /************ deallocate resources
```

```
       is done in the negotiation procedure ******/
      return(BAD_VALUE);
      break;
    }

}
```

```
/*************************************************************************/
/* Filename : ServiceKernel.c                                            */
/* Purpose : Functions for QoS management, such as QoS translations,     */
/*           negotiation, mapping of QoS and resource                    */
/* Author : Klara Nahrstedt                                              */
/* Update : 7/06/95                                                     */
/*************************************************************************/

/*#include "/home/klara/tele.d/include.d/defs.h" */
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <math.h>
#include     <sys/time.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>


#include "/home/klara/tele.d/include.d/retta.h"
#include "/home/klara/tele.d/include.d/comm.h"
#include "/home/klara/tele.d/include.d/systemQoS.h"


/************************************************/
/* UMS includes                                 */
/************************************************/


#include <UMSObject.h>
#include <UMSClass.h>
#include <UMSVideoIO.h>
#include <UMSStrings.h>

/************************************************/
/* UMS variables                                */
/************************************************/

UMSVideoIO  video_obj;
Environment *ev;
long colormap_size;

/*************************************************************************
 * X variables
 *************************************************************************/

 Display         *dpy;
 Window          wid;
 int                    screen;
 GC                     gc;
 int                    depth;
 int                    pixel_pad;
 Visual          *my_visual;
 Status          result;
 XVisualInfo     info;
 XSetWindowAttributes wa;
 long            colormap_size;


/*************************************************************************/
/* Initiatlization of application QoS and multimedia devices to zeros   */
/*************************************************************************/

int init_app_qos(param)
APP_QOS *param;
{
  bzero((char *)(param),sizeof(APP_QOS));
}

int init_devices(dev)
MM_DEVICES *dev;
{
   bzero((char *)(dev),sizeof(MM_DEVICES));
}
/*************************************************************************/
/* openProfile opens the QoS  database  for application, system,network*/
/*************************************************************************/

openProfile(fd,name)
int *fd;
char name[10];
{
   if ((*fd=open(name,O_RDWR | O_CREAT)) == -1)
    {
      perror("Couldn't open profile database ");
      exit(-1);
    }
}

/*************************************************************************/
/* close Profile closes the QoS database                                 */
/*************************************************************************/

closeProfile(fd)
int  fd;
{
  if (close(fd) == -1)
    {
      perror("Couldn't close the profile database ");
      exit(-1);
    }
}
/*************************************************************************/
/* SetAppQoS procedure sets application QoS in Application QoS Profile */
/*************************************************************************/

int setAppQoS(Param,inout)
APP_QOS *Param;
int inout;
{
  int n_bytes;
  int fd;
  /*************************************************************************/
  /* The application QoS profile is organized as follows: first come the*/
  /* application Qos of the sending stream, then comes the output QoS   */
  /* i.e., application QoS of receiving stream                          */
  /*************************************************************************/

  openProfile(&fd, "appQoSprofile");
  if (inout == INPUT)
    {
      lseek(fd,0L,0); /*position at the beginning of the file */
      if ((n_bytes=write(fd,(char *)(Param),sizeof(APP_QOS)))== -1)
        {
          perror("setAppQoS: write QoS input to profile  failure");
          return(-1);
        }
      closeProfile(fd);
      return(0);
```

```
    }
  if (inout == OUTPUT)
    {
      /* position  after the input application QoS from 0 of the file */
      lseek(fd,sizeof(APP_QOS),0);
      if (write(fd,(char *)(Param),sizeof(APP_QOS))== -1)
        {
          perror("setAppQoS: write QoS output  to profile  failure");
          return(-1);
        }
      closeProfile(fd);
      return(0);
    }

}
/*************************************************************************/
/* GetAppQoS procedure gets application QoS from Application QoS Profile*/
/*************************************************************************/

int getAppQoS(Param,inout)
APP_QOS *Param;
int inout;
{
  int fd;
  openProfile(&fd, "appQoSprofile");
  if (inout == INPUT)
    {
      lseek(fd,0L,0); /*position at the beginning of the file */
      if (read(fd,(char *)(Param),sizeof(APP_QOS))== -1)
        {
          perror("getAppQoS: read QoS input to profile  failure");
          return(-1);
        }
      closeProfile(fd);
      return(0);
    }
  if (inout == OUTPUT)
    {
      /* position  after the input application QoS from 0 of the file */
      lseek(fd,sizeof(APP_QOS),0);
      if (read(fd,(char *)Param,sizeof(APP_QOS))== -1)
        {
          perror("getAppQoS: read QoS output  to profile  failure");
          return(-1);
        }
      closeProfile(fd);
      return(0);
    }
}
/*************************************************************************/
/* SetAppTaskParam sets OS parameters in system parameter profile      */
/*************************************************************************/

SetTaskParam(side)
int side;
{

  TASKS app;
  int fd;
  int n_bytes;

  bzero((char *)(&app),sizeof(TASKS));
  openProfile(&fd,"systemQoSprofile");

  switch(side)
    {
    case BUYER:
/*************************************************************************/
/* Initialization of tasks processing robotics data                      */
/*************************************************************************/

      app.medium = ROBOT;
      app.inout = INPUT;
      app.app[0].name = ReadRobotData;
      app.app[0].ordering = 1;
      app.app[0].duration = 337; /* in microsecond */

      if ((n_bytes=write(fd,(char *)(&app),sizeof(TASKS)))== -1)
        {
          perror("setAppTaskParam: write master QoS to profile  failure");
          return(-1);
        }

      bzero((char *)(&app),sizeof(TASKS));
      app.medium = ROBOT;
      app.inout = OUTPUT;
      app.app[0].name = WriteRobotData;
      app.app[0].ordering = 2;
      app.app[0].duration = 558; /* in microseconds */


      if ((n_bytes=write(fd,(char *)(&app),sizeof(TASKS)))== -1)
        {
          perror("setAppTaskParam: write master QoS to profile  failure");
          return(-1);
        }

/*************************************************************************/
/* Initialization of tasks processing video data                         */
/*************************************************************************/
      bzero((char *)(&app),sizeof(TASKS));

      app.medium = VIDEO;
      app.inout = INPUT;
      app.app[0].name = ReadVideoData;
      app.app[0].ordering = 3;
      app.app[0].duration = 4220; /* in microseconds */


      if ((n_bytes=write(fd,(char *)(&app),sizeof(TASKS)))== -1)
        {
          perror("setAppTaskParam: write master QoS to profile  failure");
          return(-1);
        }


      bzero((char *)(&app),sizeof(TASKS));

      app.medium = VIDEO;
      app.inout = OUTPUT;
      app.app[0].name = WriteVideoData;
      app.app[0].ordering = 4;
      app.app[0].duration = 35200;


      if ((n_bytes=write(fd,(char *)(&app),sizeof(TASKS)))== -1)
        {
          perror("setAppTaskParam: write master QoS to profile  failure");
```

```
        return(-1);
      }

/*************************************************************************/
/* Initialization of Network Tasks                                       */
/*************************************************************************/
      bzero((char *)(&app),sizeof(TASKS));

      app.medium = NETWORK;
      app.inout = INPUT;
      app.app[0].name = SendCell;
      app.app[0].ordering = 1;
      app.app[0].duration = 350; /* in microseconds */
      app.app[1].name = SendDatagram;
      app.app[1].ordering = 2;
      app.app[1].duration=479; /* 479 for robotics datagram */

      if ((n_bytes=write(fd,(char *)(&app),sizeof(TASKS)))== -1)
        {
          perror("setAppTaskParam: write master QoS to profile  failure");
          return(-1);
        }

      bzero((char *)(&app),sizeof(TASKS));

      app.medium = NETWORK;
      app.inout = OUTPUT;
      app.app[0].name = ReceiveCell;
      app.app[0].ordering = 1;
      app.app[0].duration = 390; /* in microseconds */
      app.app[1].name=ReceiveDatagram;
      app.app[1].ordering = 2;
      app.app[1].duration = 14850;

      if ((n_bytes=write(fd,(char *)(&app),sizeof(TASKS)))== -1)
        {
          perror("setAppTaskParam: write master QoS to profile  failure");
          return(-1);
        }

      break;
    case SELLER:

/*************************************************************************/
/* Initialization of tasks processing robotics data                      */
/*************************************************************************/

      app.medium = ROBOT;
      app.inout = INPUT;
      app.app[0].name = ReadRobotData;
      app.app[0].ordering = 1;
      app.app[0].duration = 155; /* in microsecond */

      if ((n_bytes=write(fd,(char *)(&app),sizeof(TASKS)))== -1)
        {
          perror("setAppTaskParam: write master QoS to profile  failure");
          return(-1);
        }

      bzero((char *)(&app),sizeof(TASKS));
      app.medium = ROBOT;
      app.inout = OUTPUT;
      app.app[0].name = WriteRobotData;
      app.app[0].ordering = 2;
      app.app[0].duration = 364; /* in microseconds */


      if ((n_bytes=write(fd,(char *)(&app),sizeof(TASKS)))== -1)
        {
          perror("setAppTaskParam: write master QoS to profile  failure");
          return(-1);
        }

/*************************************************************************/
/* Initialization of tasks processing video data                         */
/*************************************************************************/
      bzero((char *)(&app),sizeof(TASKS));

      app.medium = VIDEO;
      app.inout = INPUT;
      app.app[0].name = ReadVideoData;
      app.app[0].ordering = 3;
      app.app[0].duration = 4220; /* in microseconds */


      if ((n_bytes=write(fd,(char *)(&app),sizeof(TASKS)))== -1)
        {
          perror("setAppTaskParam: write master QoS to profile  failure");
          return(-1);
        }


      bzero((char *)(&app),sizeof(TASKS));

      app.medium = VIDEO;
      app.inout = OUTPUT;
      app.app[0].name = WriteVideoData;
      app.app[0].ordering = 0;
      app.app[0].duration = 35200;


      if ((n_bytes=write(fd,(char *)(&app),sizeof(TASKS)))== -1)
        {
          perror("setAppTaskParam: write master QoS to profile  failure");
          return(-1);
        }

/************** NETWORK TASKS ************************/

      bzero((char *)(&app),sizeof(TASKS));

      app.medium = NETWORK;
      app.inout = INPUT;
      app.app[0].name = SendCell;
      app.app[0].ordering = 1;
      app.app[0].duration = 155; /* in microseconds */
      app.app[1].name = SendDatagram;
      app.app[1].ordering = 2;
      app.app[1].duration= 3082;

      if ((n_bytes=write(fd,(char *)(&app),sizeof(TASKS)))== -1)
        {
          perror("setAppTaskParam: write master QoS to profile  failure");
          return(-1);
        }

      bzero((char *)(&app),sizeof(TASKS));
```

```
      app.medium = NETWORK;
      app.inout = OUTPUT;
      app.app[0].name = ReceiveCell;
      app.app[0].ordering = 1;
      app.app[0].duration = 155; /* in microseconds */
      app.app[1].name=ReceiveDatagram;
      app.app[1].ordering = 2;
      app.app[1].duration = 18302;

      if ((n_bytes=write(fd,(char *)(&app),sizeof(TASKS)))== -1)
        {
          perror("setAppTaskParam: write master QoS to profile  failure");
          return(-1);
        }

      break;
    }
  closeProfile(fd);
}

/*************************************************************************/
/* GetAppTaskParam retrieves OS parameters from system profile         */
/*************************************************************************/

GetTaskParam(task_i, medium,inout)
TASKS *task_i; /* return all values of a task specified by the keywords */
int medium; /* keyword to search for in system profile */
int inout;   /* keyword to search for in system profile */
{
  int fd;
  off_t offset;
  offset = 0L;
  openProfile(&fd,"systemQoSprofile");
  lseek(fd,0L,0); /*position at the beginning of the file */
  switch(medium)
    {
    case ROBOT:
      switch(inout)
        {
        case INPUT:
          lseek (fd,0L,0);
          if (read(fd,(char *)(task_i),sizeof(TASKS))== -1)
            {
              perror("getAppTaskParam: read task  profile  failure");
              return(-1);
            }
          break;
        case OUTPUT:
          offset =sizeof(TASKS);
          lseek(fd,offset,0);
          if (read(fd,(char *)(task_i),sizeof(TASKS))== -1)
            {
              perror("getAppTaskParam: read task  profile  failure");
              return(-1);
            }
          break;
        }
      break;
    case VIDEO:
      switch(inout)
        {
        case INPUT:
          offset = 2*sizeof(TASKS);
          lseek (fd,offset,0);
          if (read(fd,(char *)(task_i),sizeof(TASKS))== -1)
            {
              perror("getAppTaskParam: read task  profile  failure");
              return(-1);
            }
          break;
        case OUTPUT:
          offset = 3*sizeof(TASKS);
          lseek(fd,offset,0);
          if (read(fd,(char *)(task_i),sizeof(TASKS))== -1)
            {
              perror("getAppTaskParam: read task  profile  failure");
              return(-1);
            }
          break;
        }
      break;
    case NETWORK:
      switch(inout)
        {
        case INPUT:
          offset = 4*sizeof(TASKS);
          lseek (fd,offset,0);
          if (read(fd,(char *)(task_i),sizeof(TASKS))== -1)
            {
              perror("getAppTaskParam: read task  profile  failure");
              return(-1);
            }
          break;
        case OUTPUT:
          offset = 5*sizeof(TASKS);
          lseek(fd,offset,0);
          if (read(fd,(char *)(task_i),sizeof(TASKS))== -1)
            {
              perror("getAppTaskParam: read task  profile  failure");
              return(-1);
            }
          break;
        }

    default:
      return(NOT_SUPPORTED);
      break;
    }

  closeProfile(fd);
}
/*************************************************************************/
/* SetAppSysParam sets system parameter description of application in  */
/* system parameter profile                                            */
/*************************************************************************/

/*************************************************************************/
/* Management Operations over Global State                             */
/*************************************************************************/
/* Store Global State */
StoreGlobalState(SystemState,Schedule,WhatInfo)
GLOBAL_STATE SystemState;
RATE_MONOTONIC_SCHEDULER    Schedule;
INFO_STATE WhatInfo;
{
  int fd;
  int i;
  off_t offset;
```

```
  offset = 0L;
  openProfile(&fd,"GlobalInfoProfile");

  for (i=0; i<NUMBER_OF_STATE_INFO; i++)
    {
      switch(WhatInfo.i_set[i])
        {
        case SystemStateInfo:
          offset = 0L;
          lseek(fd,offset,0);
          if (write(fd,(char *)(&SystemState),sizeof(GLOBAL_STATE))== -1)
            {
              perror("storeGlobalState: write   to profile  failure");
              return(-1);
            }
          break;
        case ScheduleInfo:
          offset =sizeof(GLOBAL_STATE);
          lseek(fd, offset,0);
          if (write(fd,(char *)(&Schedule),sizeof(RATE_MONOTONIC_SCHEDULER))== -1)
            {
              perror("storeGlobalState: write   to profile  failure");
              return(-1);
            }
        default:
          break;
        }
    }
  closeProfile(fd);
}
/* retrieve Global State */
RetrieveGlobalState(SystemState,Schedule,WhatInfo)
GLOBAL_STATE *SystemState;
RATE_MONOTONIC_SCHEDULER *Schedule;
INFO_STATE WhatInfo;
{
  int fd;
  int i;
  off_t offset;
  openProfile(&fd,"GlobalInfoProfile");
  for (i=0; i<NUMBER_OF_STATE_INFO; i++)
    {
      switch(WhatInfo.i_set[i])
        {
        case SystemStateInfo:
          offset = 0L;
          lseek(fd, offset,0);
          if (read(fd,(char *)(SystemState),sizeof(GLOBAL_STATE))== -1)
            {
              perror("retrieveGlobalState: read  profile  failure");
              return(-1);
            }
          break;
        case ScheduleInfo:
          offset = sizeof(GLOBAL_STATE);
          lseek(fd, offset,0);
          if (read(fd,(char *)(Schedule),sizeof(RATE_MONOTONIC_SCHEDULER))== -1)
            {
              perror("retrieveGlobalState: read  profile  failure");
              return(-1);
            }
          break;
        default:
          break;
        }
    }

  closeProfile(fd);
}
/*************************************************************************/
/* GetAppSysParam retrieves system description of application from    */
/* system parameter profile                                          */
/*************************************************************************/

/*************************************************************************/
/* AdmitAppQoS  admits application QoS parameters                    */
/*************************************************************************/

int AdmitAppQoS(Param,Notification,inout,side)
APP_QOS *Param;
NOTIFY *Notification;
int inout;
int side;
{
  int i,j,k,m,t,s,mi;
  BOOLEAN first, preempt;
  BOOLEAN FirstRun;
  int ticksOld,minPeriodOld;
  TASKS task_i;
  TASKS preempted_tasks;
  TASK task;
  long taskProcTime,task_begin;
  GLOBAL_STATE SystemState;
  int reply, previousTasks,repeatTasks,newTasks;
  INFO_STATE info;
  long min_period, period;
  RATE_MONOTONIC_SCHEDULER rms;
  long x[2*MEDIA_NUMBER];

  FirstRun = TRUE;
  previousTasks=0;
  newTasks = 0; /* video tasks */
  repeatTasks = 0;

  for (i=1;i<MEDIA_NUMBER;i++)
    {
      Param->stream[i].scheduled = FALSE;
    }
  first = TRUE;
  bzero((char *)(&info),sizeof(INFO_STATE));
  min_period = 0;
  bzero((char *)(&preempted_tasks),sizeof(TASKS));

  info.i_set[0]=SystemStateInfo;
  info.i_set[1]=ScheduleInfo;
  RetrieveGlobalState(&SystemState,&rms,info);
  ticksOld = rms.number_of_ticks;
  minPeriodOld = rms.min_period;

  for (i=1; i < MEDIA_NUMBER; i++)
    {
      if (Param->stream[i].type != NOT_SPECIFIED)
        {
          reply=GetTaskParam(&task_i, Param->stream[i].type,inout);

          if (reply== NOT_SUPPORTED || (task_i.medium == VIDEO &&
                                        task_i.app[0].name == NOT_SUPPORTED))
            {
```

```
            Notification->note   = NEG_REJECT;
            if (Param->stream[i].type == VIDEO)
              {
                Notification->reason = VIDEO_NOT_SUPPORTED;
                /* release video description */
                releaseMedium(Param,VIDEO);
                setAppQoS(Param,inout);
              }
            return(BAD_VALUE);
          }
/*********************************************************************/
/* KEEP GLOBAL STATE                                                 */
/*********************************************************************/
/* store Global State what media are specified, and which
           direction was tested
 I store in the global state only medium which satisfied the
end-to-end delay condition , only for this medium it makes sence
to prove further schedulablity                                       */

         SystemState.app.sdirection[inout].media[i].medium = Param->stream[i].type;
         SystemState.app.sdirection[inout].media[i].rate = Param->stream[i].medium.app_
spec.sample_rate;
/*********************************************************************/
/* FIND MINIMAL PERIOD                                               */
/*********************************************************************/
         switch(Param->stream[i].type)
           {
           case ROBOT:
             if (first)
               {

                 min_period = 1000000/Param->stream[i].medium.app_spec.sample_rate;
                 first = FALSE;
               }

             period = 1000000/Param->stream[i].medium.app_spec.sample_rate;
             break;
           case VIDEO:
             if (first)
               {
                 min_period = 60000000/Param->stream[i].medium.app_spec.sample_rate;
                 first = FALSE;
               }
             period = 60000000/Param->stream[i].medium.app_spec.sample_rate;
             break;
           }
         if (min_period >  period)
           min_period = period;
       }
   }

  printf("AdmitAppQoS: Mininam Period = %d \n", min_period);

  for (m=1;m<MEDIA_NUMBER;m++)
    {
      printf("AdmitAppQoS: mediaType[%d]= %d, rate = %d ",
             m,
             SystemState.app.sdirection[INPUT].media[m].medium,
             SystemState.app.sdirection[INPUT].media[m].rate);
      printf("AdmitAppQoS: mediaType[%d]= %d , rate =%d",
             m,
             SystemState.app.sdirection[OUTPUT].media[m].medium,
             SystemState.app.sdirection[OUTPUT].media[m].rate);
      printf("Param->stream[m].type = %d,inout=%d, Param->rate=%d \n",
             Param->stream[m].type,
             inout,
             Param->stream[m].medium.app_spec.sample_rate);
    }


/*********************************************************************/
/* SCHEDULABILITY TEST IN THE APPLICATION SUBSYSTEM                  */
/*********************************************************************/
  if (rms.min_period == 0)
    {
      rms.min_period = min_period;
    }
  else
    {
      if (rms.min_period > min_period)
        {
          rms.min_period = min_period;
        }
      else
        {
          /* rms.min_period unchanged */
        }
    }

  /********** find lcm for spec of number of ticks ************************/

  k=0;
  for (i=1;i<MEDIA_NUMBER;i++)
    {
      if (SystemState.app.sdirection[inout].media[i].rate !=0)
        {
          switch(SystemState.app.sdirection[inout].media[i].medium)
            {
            case ROBOT:
              x[k] = 1000000/SystemState.app.sdirection[inout].media[i].rate;
              break;
            case VIDEO:
              x[k] = 60000000/SystemState.app.sdirection[inout].media[i].rate;
              break;
            }
          k++;
        }
    }

  rms.number_of_ticks =  lcm(x,k)/rms.min_period;
/*
  printf("AdmitAppQoS: number of ticks= %d, lcm = %d, k=%d, rms.min_period=%d  \n",rms.nu
mber_of_ticks,
          lcm(x,k),k,
          rms.min_period);
*/
/*  rms.number_of_ticks = 1; */
/*********************************************************************/
/* CHECH IF TASK DURATION FITS IN THE MINIMAL PERIOD                 */
/*********************************************************************/

  for(k=1; k< MEDIA_NUMBER;k++)
    {
      if (Param->stream[k].type != NOT_SPECIFIED)
        {
          taskProcTime = 0;

          GetTaskParam(&task_i,Param->stream[k].type,inout);
```

```
          for (j=0; j<NUMBER_OF_TASKS_PER_MEDIUM;j++)
            {
              taskProcTime += task_i.app[j].duration;
            }
          printf("AdmitAppQoS: <medium,taskProcTime,taskDuration,inout>=<%d,%d,%d,%d> \n
", Param->stream[k].type,taskProcTime, task_i.app[0].duration,inout);

          switch(Param->stream[k].type)
            {
            case ROBOT:
              period = 1000000/Param->stream[k].medium.app_spec.sample_rate;
              break;
            case VIDEO:
              period = 60000000/Param->stream[k].medium.app_spec.sample_rate;
              break;
            default:
              break;
            }
/*************************************************************************/
/* SCHEDULABILITY TEST: IF TASKS ARE SCHEDULABLE IN THEIR OWN PERIOD     */
/*************************************************************************/
          if (taskProcTime > period)
            {
              Notification->note = NEG_REJECT;
              Notification->reason = SCHEDULE_NOT_FEASIBLE;
              /* release medium resources */
              releaseMedium(Param,k);
              setAppQoS(Param,inout);
              rms.number_of_ticks = ticksOld;
              rms.min_period = minPeriodOld;
              SystemState.app.sdirection[inout].media[i].medium = 0;
              SystemState.app.sdirection[inout].media[i].rate =0;
              info.i_set[0] = SystemStateInfo;
              info.i_set[1] = ScheduleInfo;
              StoreGlobalState(SystemState,rms,info);
              return(BAD_VALUE);
            }
        }
    }
/*************************************************************************/
/* SCHEDULABILITY TEST: IF TASKS ARE SCHEDULABLE IN THE MINIMAL PERIOD   */
/*************************************************************************/

  preempt = FALSE;

  for (t = 0; t < rms.number_of_ticks;t++)
    {
/*************** FIND LAST ELEMENT SPECIFIED  IN SCHEDULER  MINIMAL PERIOD (M) ********/
      m=0; /* scheduler index */
      for (i=0;i<MEDIA_NUMBER*NUMBER_OF_TASKS_PER_MEDIUM;i++)
        {
          if (rms.sched[t].sched_queue[i].task_name == NOT_SPECIFIED)
            {
              if (t == 0 && FirstRun == TRUE)
                {
                  previousTasks = i;
                  FirstRun = FALSE;
                }
              else
                {
                  for (mi=0; mi < previousTasks+repeatTasks;mi++)
                    {
                      rms.sched[t].sched_queue[mi].task_name =
                        rms.sched[0].sched_queue[mi].task_name;
                      rms.sched[t].sched_queue[mi].inout =
                        rms.sched[0].sched_queue[mi].inout;
                      rms.sched[t].sched_queue[mi].medium =
                        rms.sched[0].sched_queue[mi].medium;
                      rms.sched[t].sched_queue[mi].task_duration =
                        rms.sched[0].sched_queue[mi].task_duration;
                      rms.sched[t].sched_queue[mi].time_begin =
                        rms.sched[t-1].sched_queue[mi].time_begin
                          + rms.min_period;
                      rms.sched[t].sched_queue[mi].time_deadline =
                        t*rms.sched[0].sched_queue[mi].time_deadline;

                      printf("AdmitAppQoS:medium,taskname,taskbegin,taskduration = %d,%d,
%d,%d \n",
                             rms.sched[t].sched_queue[mi].medium,
                             rms.sched[t].sched_queue[mi].task_name,
                             rms.sched[t].sched_queue[mi].time_begin,
                             rms.sched[t].sched_queue[mi].task_duration);


                    }
                  m = previousTasks+newTasks+repeatTasks;
                  i=MEDIA_NUMBER*NUMBER_OF_TASKS_PER_MEDIUM;
                }
            }
/************** GET TASK BEGIN FOR M-TH TASK IN PERIOD***************/
          task_begin = 0;
          if (m==0)
            task_begin = 0;
          else
            { task_begin=rms.sched[t].sched_queue[m-1].time_begin +
                rms.sched[t].sched_queue[m-1].task_duration;
            }
          for (i=1;i<MEDIA_NUMBER;i++)
            {
              if (Param->stream[i].type !=NOT_SPECIFIED &&
                  Param->stream[i].scheduled == FALSE)
                {
                  GetTaskParam(&task_i,Param->stream[i].type,inout);

                  for (j=0; j<NUMBER_OF_TASKS_PER_MEDIUM;j++)
                    {
/******************* TASK IS SPECIFIED **********************************/
                      if (task_i.app[j].name != NOT_SPECIFIED && (task_i.app[j].Scheduled ==
FALSE))
                        {
/******************* TASK IS SCHEDULABLE IN THE MINIMAL PERIOD ***********/
                          if (task_begin+task_i.app[j].duration < (t+1)*rms.min_period)
                            {
                              rms.sched[t].sched_queue[m].task_name = task_i.app[j].name;
                              rms.sched[t].sched_queue[m].inout = task_i.inout;
                              rms.sched[t].sched_queue[m].time_begin = task_begin;

/****************** CHECK END_TO_END DELAY ****************************/
                              if (rms.sched[t].sched_queue[m].time_begin +
                                  task_i.app[j].duration >  (Param->stream[i].medium.net_spec
.end_to_end_delay * 1000.0))
                                {
                                  Notification->note = NEG_REJECT;
                                  Notification->reason = END_TO_END_TEST_FAILURE;
                                  /* release medium description */
                                  releaseMedium(Param,i);
```

```
                        printf("released medium i=%d, type = %d \n",i,Param->strea
m[i].type);
                        setAppQoS(Param,inout);
                        /* release resources ***********************/
                        rms.number_of_ticks = ticksOld;
                        rms.min_period = minPeriodOld;
                        freeSchedResources(&rms,t,m);
                        SystemState.app.sdirection[inout].media[i].medium = 0;
                        SystemState.app.sdirection[inout].media[i].rate =0;
                        info.i_set[0] = SystemStateInfo;
                        info.i_set[1] = ScheduleInfo;
                        StoreGlobalState(SystemState,rms,info);
                        return(BAD_VALUE);
                      }

                    switch(Param->stream[i].type)
                      {
                      case ROBOT:
                        period = 1000000/Param->stream[i].medium.app_spec.sample_r
ate;
                        break;
                      case VIDEO:
                        period = 60000000/Param->stream[i].medium.app_spec.sample_
rate;
                        break;
                      default:
                        break;
                      }
                    rms.sched[t].sched_queue[m].time_deadline =
                      task_begin + period;
                    rms.sched[t].sched_queue[m].medium = Param->stream[i].type;
                    rms.sched[t].sched_queue[m].task_duration =
                      task_i.app[j].duration;
/*
                    printf("AdmitAppQoS: sched=t,m,inout,taskname,taskbegin,medium
,deadline,duration (%d,%d,%d,%d,%d,%d,%d, %d)\n",t,m,
                           rms.sched[t].sched_queue[m].inout,
                           rms.sched[t].sched_queue[m].task_name,
                           rms.sched[t].sched_queue[m].time_begin,
                           rms.sched[t].sched_queue[m].medium,
                           rms.sched[t].sched_queue[m].time_deadline,
                           rms.sched[t].sched_queue[m].task_duration);
*/
                    task_begin = task_begin + task_i.app[j].duration;
                    switch(Param->stream[i].type)
                      {
                      case ROBOT:
                        repeatTasks = j+1;
                        break;
                      case VIDEO:
                        newTasks = j+1;
                        break;
                      }

                    m++;
                    task_i.app[j].Scheduled = TRUE;
                  }
                else
/********************* TASK IS NOT SCHEDULABLE IN THE MINIMAL PERIOD *******/
                  {
                    if (task_i.app[j].duration > rms.min_period)
                      {
                        Notification->note = NEG_REJECT;
                        Notification->reason = SCHEDULE_NOT_FEASIBLE;
                        /* release medium description */
                        releaseMedium(Param,i);
                        setAppQoS(Param,inout);
                        /* release resources ***********************/
                        rms.number_of_ticks = ticksOld;
                        rms.min_period = minPeriodOld;
                        freeSchedResources(&rms,t,m);
                        SystemState.app.sdirection[inout].media[i].medium = 0;
                        SystemState.app.sdirection[inout].media[i].rate =0;
                        info.i_set[0] = SystemStateInfo;
                        info.i_set[1] = ScheduleInfo;
                        StoreGlobalState(SystemState,rms,info);
                        return(BAD_VALUE);
                      }
/********************* CHECK END_TO_END_DELAY ******************************/
                    if (task_begin+task_i.app[j].duration >  (Param->stream[i].medi
um.net_spec.end_to_end_delay * 1000.0))
                      {
                        Notification->note = NEG_REJECT;
                        Notification->reason = END_TO_END_TEST_FAILURE;
                        /* release medium description */
                        releaseMedium(Param,i);
                        setAppQoS(Param,inout);
                        /* release resources ***********************/
                        rms.number_of_ticks = ticksOld;
                        rms.min_period = minPeriodOld;
                        freeSchedResources(&rms,t,m);
                        SystemState.app.sdirection[inout].media[i].medium = 0;
                        SystemState.app.sdirection[inout].media[i].rate =0;
                        info.i_set[0] = SystemStateInfo;
                        info.i_set[1] = ScheduleInfo;
                        StoreGlobalState(SystemState,rms,info);
                        return(BAD_VALUE);
                      }
                    preempt = TRUE;
                    /* preempt tasks to  the next period */
                    preempted_tasks.medium =Param->stream[i].type;
                    preempted_tasks.inout = inout;
                    switch(Param->stream[i].type)
                      {
                      case ROBOT:
                        period = 1000000/Param->stream[i].medium.app_spec.sample_ra
te;
                        break;
                      case VIDEO:
                        period = 60000000/Param->stream[i].medium.app_spec.sample_r
ate;
                        break;
                      default:
                        break;
                      }
                    preempted_tasks.app[j].period = period;
                    preempted_tasks.app[j].name = task_i.app[j].name;
                    preempted_tasks.app[j].duration = task_i.app[j].duration;
                    j=NUMBER_OF_TASKS_PER_MEDIUM;
                    i=MEDIA_NUMBER;
                  }
              }
            else
/********************* TASK IS NOT SPECIFIED (END OF SCHEDULER) *************/
              {
                if (task_i.app[j].name == NOT_SPECIFIED )
                  {
```

```
                        task_begin =rms.sched[t].sched_queue[m-1].time_begin + task_i.
app[j-1].duration;
                        if (task_begin < (t+1)*rms.min_period && preempt == TRUE)
                          {
                            s =0;
                            while (task_begin < (t+1)*rms.min_period &&
                                   (preempted_tasks.app[s].name!=NOT_SPECIFIED))
                              {
                                rms.sched[t].sched_queue[m].task_name =
                                  preempted_tasks.app[s].name;
                                rms.sched[t].sched_queue[m].time_begin =
                                  task_begin;
                                rms.sched[t].sched_queue[m].time_deadline =
                                  task_begin + preempted_tasks.app[s].period;
                                rms.sched[t].sched_queue[m].task_duration =
                                  preempted_tasks.app[s].duration;
                                rms.sched[t].sched_queue[m].medium = preempted_tasks.m
edium;

                                m++;
                                task_begin +=preempted_tasks.app[s].duration;
                                bzero((char *)(&preempted_tasks.app[s]), sizeof(TASK))
;

                                s++;
                              }
                            if (preempted_tasks.app[s].name == NOT_SPECIFIED)
                              preempt = FALSE;
                            else
                              preempt = TRUE;
                          }
                      }
                  }
              }
            Param->stream[i].scheduled = TRUE;
          }
      }
  }



  for (s=0;s<10;s++)
       printf("AdmitAppQoS: scheduled tasks %d \n", rms.sched[0].sched_queue[s].task_na
me);



/*********************************************************************/

  info.i_set[0] = SystemStateInfo;
  info.i_set[1] = ScheduleInfo;
  StoreGlobalState(SystemState,rms,info);
  Notification->note = NEG_SUCCESS;
  return(NEG_SUCCESS);

}
/*********************************************************************/
/* Negotiate AppQoS negotiates application QoS between application     */
/* sender and receiver                                               */
/*********************************************************************/

negotiateAppQoS(param,add_info,notification,side,direction)
APP_QOS *param;
ADD_INFO *add_info;
NOTIFY *notification;
int side;
int *direction;
{

/********************* video variables **********************/

  unsigned char   *imagedata;
  unsigned char *data;

  XImage                  *image;
  long                    width;
  long                    height;
  int                     number,end;
  int                             num_frames;
  char                    ch;
  long                    error_code;
  int rc;
  int image_size;
  double frame_size;
  double frame_rate;
  long each_buffer_size;
  long number_of_elements;
  _IDL_SEQUENCE_UMSVideoIO_RingBufferElement *ring_buffer;
  UMSVideoIO_RingBufferElement *rbuffer1;
  long index;
  long flag;
  unsigned char  * Address[5];


  int vci1,vci2;
  int result;
  int conid1;
  int conid2;
  int i,inout;
  FEC_FLAGS err;
  NEG_RESPONSE response;
  GLOBAL_STATE SystemState;
  RATE_MONOTONIC_SCHEDULER rms;
  INFO_STATE WhatInfo;
  struct timeval tv1n,tv2n;
  struct timeval tv1,tv2;
  struct timezone tz;
  long clock;
  inout = *direction;
  WhatInfo.i_set[0] = SystemStateInfo;
  WhatInfo.i_set[1] = NOT_SPECIFIED;
  RetrieveGlobalState(&SystemState,&rms,WhatInfo);

  ev = somGetGlobalEnvironment();
  frame_rate = 15.00;
  colormap_size = 128;
  width = 240;
  height = 160;
  depth = 8;
  image_size = width*height;
  if ((imagedata=(unsigned char *)malloc(width*height)) == NULL)
    {
      exit(-1);
    }


  switch(side)
    {
    case BUYER:
```

```
/******************** send application QoS to the remote site ************/
      if (SystemState.net.Aneg_in.status == FREE)
        {
          vci1 = APP_SIGNAL1_VCI;
          connect_s(vci1,&conid1,DATAGRAM_MODE,sizeof(APP_QOS));
          SystemState.net.Aneg_in.status = TAKEN;
          SystemState.net.Aneg_in.id = conid1;
        }
      response.result = ACCEPT;

      err.err_flag = FALSE;
      if ((result = send_pkt(SystemState.net.Aneg_in.id,&response,sizeof(NEG_RESPONSE),e
rr))== WRONG_SIZE)
        {
          perror("negotiateAppQoS datagram too small for response transmission");
          exit(1);
        }
      if ((result = send_pkt(SystemState.net.Aneg_in.id,param,sizeof(APP_QOS),err))== WR
ONG_SIZE)
        {
          perror("negotiateAppQoS datagram too small for app QoS transmission");
          exit(1);
        }

      if ((result = send_pkt(SystemState.net.Aneg_in.id,add_info,sizeof(ADD_INFO),err))=
= WRONG_SIZE)
        {
          perror("negotiateAppQoS datagram too small for add_info transmission");
          exit(1);
        }
/******************** wait for response from the remote side ***************/
      if (SystemState.net.Aneg_out.status == FREE)
        {
          vci2 = APP_SIGNAL2_VCI;
          connect_r(vci2,&conid2,DATAGRAM_MODE);
          SystemState.net.Aneg_out.status = TAKEN;
          SystemState.net.Aneg_out.id = conid2;
        }
      printf("BUYER: sent add_info[GET_IMAGE] = %d \n",
                  add_info->info[GET_IMAGE].done);

      if (add_info->info[GET_IMAGE].done == TRUE)
        {
          /***************** create window and display video **************/

         /* Create the video IO object */

          video_obj = UMSVideoIONew();

          /* Create the display window */

          rc = create_window(width, height);
          if (rc == 1)
            {
              error_code = 1;
              return(error_code);
            }

          err.err_flag = FALSE;

          recv_pkt(SystemState.net.Aneg_out.id,imagedata,image_size,err);

          image = XCreateImage(dpy, my_visual, depth, ZPixmap, 0,
                               imagedata,
                               width, height, pixel_pad, 0);
          XPutImage(dpy, wid, gc, image, 0, 0, 0, 0, width, height);
          XSync(dpy,FALSE);
          XFlush(dpy);

          sleep(10);
          XFreeGC(dpy,gc);
          XDestroyWindow(dpy,wid);
          StoreGlobalState(SystemState,rms,WhatInfo);
          notification->note = NEG_SUCCESS;
          notification->reason = GET_IMAGE_SUCCESS;
          return(0);
        }

      recv_pkt(SystemState.net.Aneg_out.id,&response,sizeof(NEG_RESPONSE),err);
      switch(response.result)
        {
        case ACCEPT:
          notification->note = NEG_SUCCESS;
          break;
        case MODIFY:
          notification->note = NEG_MODIFY;
          notification->reason = response.reason;
          bcopy((char *)(&response.stream_spec),(char *)(param),sizeof(APP_QOS));
          break;
        case REJECT:
          notification->note = NEG_REJECT;
          notification->reason = response.reason;
          bzero((char *)(param),sizeof(APP_QOS));
          setAppQoS(param, inout);
          break;
        }
      break;
    case SELLER:
      if (SystemState.net.Aneg_out.status == FREE)
        {
          vci1 = APP_SIGNAL1_VCI;
          connect_r(vci1,&conid1,DATAGRAM_MODE);
          SystemState.net.Aneg_out.status = TAKEN;
          SystemState.net.Aneg_out.id = conid1;
        }
      err.err_flag = FALSE;
      recv_pkt(SystemState.net.Aneg_out.id,&response,sizeof(NEG_RESPONSE),err);
      if (response.result == REJECT)
        {
          notification->note = NEG_REJECT;
          notification->reason = response.reason;
;
          return(BAD_VALUE);
        }
      recv_pkt(SystemState.net.Aneg_out.id,param,sizeof(APP_QOS),err);


      for (i=1;i < MEDIA_NUMBER; i++)
        {
          if (param->stream[i].type != NOT_SPECIFIED)
            {
              inout = param->stream[i].direction;
/*
              printf("negotiateAppQoS: Parameter(type)=%d\n",param->stream[i].type);
              printf("QoS Broker Seller: Parameter(sample_rate)=%d\n",
                     param->stream[i].medium.app_spec.sample_rate);

              printf("received direction = %d and inout = %d \n",
```

```
                    param->stream[i].direction,inout);
*/

           }

        }
      if (inout == INPUT)
        {
          inout = OUTPUT;
        }
      else
        {
        inout = INPUT;
      }
/*
      printf("received direction = %d and actual direction = %d \n",
                  param->stream[i].direction,inout);
*/

/*************** receive additional information ******************    *******/

      recv_pkt(SystemState.net.Aneg_out.id,add_info,sizeof(ADD_INFO),err);
      if (SystemState.net.Aneg_in.status == FREE)
        {
          vci2 = APP_SIGNAL2_VCI;
          connect_s(vci2,&conid2,DATAGRAM_MODE,image_size);
          SystemState.net.Aneg_in.status = TAKEN;
          SystemState.net.Aneg_in.id = conid2;
        }
      printf("SELLER: received add_info[GET_IMAGE] = %d \n",
                  add_info->info[GET_IMAGE].done);

      if (add_info->info[GET_IMAGE].done == TRUE)
        {
          /****************** setup video device ************************\
*/

          /************* Create the video IO objcet ***********/

          video_obj = UMSVideoIONew();

          /************ Open the video device ***************/

          rc = UMSVideoIO_open(video_obj,ev,"/dev/sr0");
          if (rc != UMSVideoIO_Success)
            {
              printf("Cannot open video device. rc = %d\n",rc);
              error_code = 1;
              goto Error;
            }

          /********* Set object parameters ****************/

          /*********** Set analog input format for capture card */

          rc = UMSVideoIO_set_analog_video_format(video_obj,ev,"NTSC");
          if (rc != UMSVideoIO_Success)
            {
              printf("Cannot set analog video format. rc = %d\n", rc);
              error_code = 1;
              goto Error;
            }
          /*********** Set output digital video format ***********/
          if (depth == 24 )
            rc = UMSVideoIO_set_output_image_format(video_obj,ev,"RGB24");
          else
            rc = UMSVideoIO_set_output_image_format(video_obj,ev,"RGB8Dither");

          if (rc != UMSVideoIO_Success)
            {
              printf("Cannot set output image format. rc =%d \n",rc);
              error_code = 1;
              goto Error;
            }

          /*********** Capture digital video as uncompressed frames ***/
          /********** Here we set the dimensions of the frames ********/

          rc = UMSVideoIO_set_uncompressed_image_size(video_obj,ev,&width,
                                                      &height);
          if (rc != UMSVideoIO_Success)
            {
              printf("Cannot set uncompressed image size. rc = %d \n",rc);
              error_code =1;
              goto Error;
            }
          /*********** Set up the ring of buffers to receive the frames ***/

          /* Compute the size of one buffer ****************************/

          if (depth == 8)
            each_buffer_size = width*height;
          else
            each_buffer_size = 4*width*height;

          /* number of buffers *****************************************/
          number_of_elements = 4;

          /************ SOM sequence containing an array of ring-buffer
            structures */

          ring_buffer = (_IDL_SEQUENCE_UMSVideoIO_RingBufferElement*)
            malloc(sizeof(_IDL_SEQUENCE_UMSVideoIO_RingBufferElement));

          if (ring_buffer < 0)
            {
              printf("Cannot malloc ring_buffer \n");
              error_code = 1;
              goto Error;
            }
          ring_buffer->_length  = number_of_elements;
          ring_buffer->_maximum = number_of_elements;

          ring_buffer->_buffer = (struct UMSVideoIO_RingBufferElement *)
            malloc(sizeof(struct UMSVideoIO_RingBufferElement) * number_of_eleme\
nts);
          if (ring_buffer->_buffer <=0)
            {
              printf("Cannot malloc ring_buffer->_buffer\n");
              error_code = 1;
              goto Error;

            }


          /************* Set up each ring buffer ************************/
```

```
        rbuffer1 = ring_buffer->_buffer;

        for (i=0; i < number_of_elements; i++)
          {
            Address[i] = (unsigned char *) malloc(each_buffer_size + 4096);
            if (Address[i] <= 0)
              {
                printf("Cannot malloc ring_buffer %d \n",i);
                error_code = 1;
                goto Error;
              }
            rbuffer1->Address = (((long) Address[i]) + 4096
                                 - (((long) Address[i]) % 4096));
            rbuffer1->AfterHeader = rbuffer1->Address;
            rbuffer1->SizeOfBuffer = each_buffer_size;
            rbuffer1->SizeOfDataInBuffer = 0;
            rbuffer1->InUseByCaller = 0;
            rbuffer1++;       /* point to the next ring buffer structure */

          }


        rc = UMSVideoIO_setup_uncompressed_capture_buffers(video_obj,ev,
                                                           ring_buffer);
        if (rc != UMSVideoIO_Success)
          {
            printf("Cannot setup uncompressed_capture_buffers. rc=%d\n",rc);
            error_code =1;
            goto Error;
          }

        /************ Set capture mode to uncompressed ****************/

        rc = UMSVideoIO_set_uncompression(video_obj,ev,UMSVideoIO_On);
        if (rc != UMSVideoIO_Success)
          {
            printf("Cannot set_uncompression. rc = %d \n", rc);
            error_code =1;
            goto Error;
          }
        rc = UMSVideoIO_set_capture_rate(video_obj,ev,&frame_rate);
        if (rc != UMSVideoIO_Success)
          {
            printf("Cannot get capture rate  rc = %d \n", rc);
            error_code =1;
            goto Error;
          }

        /************** GET IMAGE and SEND IT *******************/
        index = 0;
        flag = 1;
        rc = UMSVideoIO_get_uncompressed_frame(video_obj,ev,&index,flag);
        if(rc != UMSVideoIO_Success)
          {
            printf("Cannot get_uncompressed_frame. rc = %d \n",rc);
            error_code =1;
            goto Error;
          }

        imagedata = (unsigned char *) ring_buffer->_buffer[index].Address;

        ring_buffer->_buffer[index].InUseByCaller = 0;
        /*************** SEND IMAGE **********************************/
            err.err_flag = FALSE;
            send_pkt(SystemState.net.Aneg_in.id,imagedata,image_size,err);

            rc = UMSVideoIO_close(video_obj,ev);
            if (rc !=UMSVideoIO_Success)
              {
                printf("Cannot close video device. rc = % \n", rc);
                error_code = 1;
              }
            for (i=0; i < number_of_elements; i++)
              {
                if (Address[i] != NULL)
                  free(Address[i]);
              }
            if (ring_buffer->_buffer != NULL)
              free(ring_buffer->_buffer);
            if (ring_buffer != NULL)
              free(ring_buffer);

            if (video_obj != NULL)
              _somFree(video_obj);



            StoreGlobalState(SystemState,rms,WhatInfo);
            notification->note = NEG_SUCCESS;
            notification->reason = GET_IMAGE_SUCCESS;
            return(0);
          }
        gettimeofday(&tv1n,&tz);

    /************* admission part of the negotiation process by seller *********/
        setAppQoS(param,inout);

        gettimeofday(&tv1,&tz);
        AdmitAppQoS(param,notification,inout,side);
        gettimeofday(&tv2,&tz);
        getproctime(tv1,tv2,&clock);


        printf("AdmitAppQoS = %d microsecond \n", clock);

        switch(notification->note)
          {
          case NEG_SUCCESS:
            response.result=ACCEPT;
            break;
          case NEG_MODIFY:
            response.result = MODIFY;
            response.reason = notification->reason;
            bcopy((char *)param,(char *)(&response.stream_spec),sizeof(APP_QOS));
            break;
          case NEG_REJECT:
            response.result = REJECT;
            response.reason = notification->reason;
            bzero((char *)(param),sizeof(APP_QOS));
            setAppQoS(param, inout);
            break;
          }

        err.err_flag = FALSE;
        send_pkt(SystemState.net.Aneg_in.id,&response,sizeof(NEG_RESPONSE),err);
        gettimeofday(&tv2n,&tz);
        getproctime(tv1n,tv2n,&clock);
```

```
      printf("NegotiateAppQoS = %d microsecond \n", clock);
      break;
    }
  *direction = inout;
  StoreGlobalState(SystemState,rms,WhatInfo);
  free(imagedata);
  printf("negotiateAppQoS: after free imagedata \n");
  return(0);
 Error:

  for (i=0; i < number_of_elements; i++)
    {
      if (Address[i] != NULL)
        free(Address[i]);
    }
  if (ring_buffer->_buffer != NULL)
    free(ring_buffer->_buffer);
  if (ring_buffer != NULL)
    free(ring_buffer);

  if (video_obj != NULL)
    _somFree(video_obj);
  return(error_code);


}/* negotiateAppQoS */
/*************************************************************************/
/* QoS Translator translates between application QoS and Network QoS   */
/* as well as communicates between application and transport layers    */
/*************************************************************************/
QoSTranslator(trans_direct, side,inout)
int trans_direct;
int side;   /* Byuer or Seller */
int inout;
{

  APP_QOS AParam;
  NET_QOS_TABLE NTable;
  INFO_STATE WhatInfo;
  GLOBAL_STATE SystemState;
  RATE_MONOTONIC_SCHEDULER Schedule;
  TASK task;
  int prio1,prio2,prio3,prio4;
  int i,j,k,l,m;
  double NumberOfFragments, transRate(),transInterDelay();

/*************************************************************************/
/*******************  Initialize Variables   ************************/
/*************************************************************************/
  j=0; /* first scheduler interval for intra = FALSE */
  m=0; /* first scheduler interval for intra = TRUE */
  k=0;
  prio1 = 0;
  prio2 = 5;
  prio3 = 10;
  prio4 = 20;

  switch(trans_direct)
    {
    case APP_TO_NET:
      getAppQoS(&AParam,inout);
      getNetQoS(&NTable,inout);
/*************************************************************************/
/*  compute net QoS for each medium from app QoS                        */
/*************************************************************************/
      for (i=1; i<MEDIA_NUMBER;i++)
        {
          if (AParam.stream[i].type != NOT_SPECIFIED &&
              NTable.status[k] == FREE)
            {


              printf("QoStranslator: number of connections k = %d\n",k);

/*********************************************************************/
/*              NO INTEGRATION/DISINTEGRATION REQUIRED               */
/*********************************************************************/
              if (AParam.stream[i].intra == FALSE)
                {
/*************************************************************************/
/************** 1. decide which network packet size M will be used ********/
/*************************************************************************/
                  if (AParam.stream[i].medium.app_spec.sample_size <= CELL_SIZE)
                    {
                      NTable.connection[k].load.size = CELL_SIZE;
                      NTable.connection[k].load.id = CELL_MODE;
                      switch(inout)
                        {
/********* ALLOW 20 CONNECTIONS FOR INPUT AND 20 COONECTIONS FOR OUTPUT ****/
                        case INPUT:
                          NTable.vci[k] = 0x0020 + k;
                          break;
                        case OUTPUT:
                          NTable.vci[k] = 0x0040 + k;
                          break;
                        }
                    }
                  else
                    {
                      NTable.connection[k].load.id = DATAGRAM_MODE;
                      switch(inout)
                        {
/********* ALLOW 20 CONNECTIONS FOR INPUT AND 20 COONECTIONS FOR OUTPUT ****/
                        case INPUT:
                          NTable.vci[k] = 0x4000 + k;
                          break;
                        case OUTPUT:
                          NTable.vci[k] = 0x4020 + k;
                          break;
                        }

                      if (AParam.stream[i].medium.app_spec.sample_size <= DATAGRAM_SIZE)
                        NTable.connection[k].load.size = DATAGRAM_SIZE;
/*************************************************************************/
/************** if fragmentation is required ****************************/
/*************************************************************************/
                      else
                        {
                          AParam.relations.fragmentation[i].medium = i;
                          AParam.relations.fragmentation[i].frag = TRUE;
                          NTable.connection[k].load.size = DATAGRAM_SIZE;

                        }
                    }
                  NTable.status[k] = TAKEN;
/*************************************************************************/
/****************************** Rate Translation *************************/
/*************************************************************************/
```

```
/*** Comment - need to adjust the video rate (from frames/min to frames/sec)*/

                if (AParam.stream[i].type == VIDEO)
                  {
                    AParam.stream[i].medium.app_spec.sample_rate =
                      AParam.stream[i].medium.app_spec.sample_rate/60;
                  }
                NTable.connection[k].load.rate=transRate(AParam.stream[i].medium.app_s
pec.sample_size,
                                                         AParam.stream[i].medium.app_s
pec.sample_rate,
                                                         NOT_SPECIFIED,APP_TO_NET,
                                                         NTable.connection[k].load.id,
                                                         &NumberOfFragments);
                AParam.relations.fragmentation[i].number = NumberOfFragments;
/*************************************************************************/
/* Interarrival time in milisecond - assumption is rate in samples/second    */
/*************************************************************************/
                NTable.connection[k].load.intermediate_delay = transInterDelay(AParam.
stream[i].medium.app_spec.sample_size,
                                                AParam.stream[i].medium.app_spec.sampl
e_rate,
                                                APP_TO_NET,
                                                NTable.connection[k].load.id);
/*************************************************************************/
/**** End-to-End Delay Translation (in microseconds )       ****************/
/*************************************************************************/
                NTable.connection[k].load.end_to_end_delay =
                  1000.0*((double) AParam.stream[i].medium.net_spec.end_to_end_delay);
                WhatInfo.i_set[0] = ScheduleInfo;
                WhatInfo.i_set[1] = NOT_SPECIFIED;
                WhatInfo.i_set[2] = NOT_SPECIFIED;

                printf("QoStranslator: netRate: %f , netDelay: %f, fragments: %f, conn
ection k = %d, end-to-end delay=%f \n",
                       NTable.connection[k].load.rate,
                       NTable.connection[k].load.intermediate_delay,
                       NumberOfFragments,k,
                       NTable.connection[k].load.end_to_end_delay);


                RetrieveGlobalState(&SystemState,&Schedule,WhatInfo);
                for (l=0; l<MEDIA_NUMBER*NUMBER_OF_TASKS_PER_MEDIUM;l++)
                  {
                    if (Schedule.sched[j].sched_queue[l].task_name !=
                        NOT_SPECIFIED  &&
                        (Schedule.sched[j].sched_queue[l].medium == i))
                      {
                        switch(Schedule.sched[j].sched_queue[l].task_name)
                          {
                          case ReadRobotData:
                            getOneTaskParam(&task,ReadRobotData,
                                            ROBOT,INPUT);

                            printf("QoStranslator:task duration %d \n",task.duration);

                            break;
                          case WriteRobotData:
                            getOneTaskParam(&task,WriteRobotData,
                                            ROBOT,OUTPUT);
                            break;
                          case CopyRobotData:
                            getOneTaskParam(&task,CopyRobotData,ROBOT,inout);
                            break;
                          case PointerManagementRobotData:
                            getOneTaskParam(&task,PointerManagementRobotData,ROBOT,inou
t);
                            break;
                          case ReadVideoData:
                            getOneTaskParam(&task,ReadVideoData,
                                            VIDEO,INPUT);
                            break;
                          case WriteVideoData:
                            getOneTaskParam(&task,WriteVideoData,
                                            VIDEO,OUTPUT);
                            break;
                          case CopyVideoData:
                            getOneTaskParam(&task,CopyVideoData,VIDEO,inout);
                            break;
                          case PointerManagementVideoData:
                            getOneTaskParam(&task,PointerManagementVideoData,VIDEO,inou
t);
                            break;
                          case SendCell:
                            getOneTaskParam(&task,SendCell,
                                            NETWORK,INPUT);
                            break;
                          case SendDatagram:
                            getOneTaskParam(&task,SendDatagram,
                                            NETWORK,INPUT);
                            break;
                          case ReceiveCell:
                            getOneTaskParam(&task,ReceiveCell,
                                            NETWORK,OUTPUT);
                            break;
                          case ReceiveDatagram:
                            getOneTaskParam(&task,ReceiveDatagram,
                                            NETWORK,OUTPUT);
                            break;
                          }
                        NTable.connection[k].load.end_to_end_delay =
                          NTable.connection[k].load.end_to_end_delay
                            - ((double) task.duration);

                        printf("QoStranslator: net delay %f \n",
                               NTable.connection[k].load.end_to_end_delay);

                      }
                  }


/*************************************************************************/
/******************   Loss Rate Translation   ****************************/
/*************************************************************************/
                if (AParam.relations.fragmentation[i].frag == FALSE)
                  {
                    NTable.connection[k].load.loss.loss_rate =
                      AParam.stream[i].medium.net_spec.loss_rate;
                    /* Loss of two consecutive packets not allowed */
                    NTable.connection[k].load.loss.loss_cons_pkt = FALSE;
                  }
                else
                  {
                    NTable.connection[k].load.loss.loss_rate =
                      AParam.stream[i].medium.net_spec.loss_rate*AParam.relations.fragm
entation[i].number;
                    NTable.connection[k].load.loss.loss_cons_pkt = TRUE;
                  }
```

```
/*************************************************************************/
/***************** Importance Translation    ******************************/
/*************************************************************************/
                  switch(AParam.stream[i].medium.net_spec.importance)
                    {
                    case HIGH_IMPORTANCE: /* prio: 1-5 */
                      NTable.connection[k].load.priority = prio1+1;
                      prio1++;
                      NTable.connection[k].perform.error_alg = FEC;
                      break;
                    case MEDIUM_IMPORTANCE: /* prio: 6-10 */
                      NTable.connection[k].load.priority = prio2+1;
                      prio2++;
                      NTable.connection[k].perform.error_alg = NONE;
                      break;
                    case LOW_IMPORTANCE: /*prio: 11-20 */
                      NTable.connection[k].load.priority = prio3+1;
                      prio3++;
                      NTable.connection[k].perform.error_alg = NONE;
                      break;
                    default: /* best effort, importance not specified prio>20*/
                      NTable.connection[k].load.priority = prio4+1;
                      prio4++;
                      NTable.connection[k].perform.error_alg = NONE;
                      break;
                    }
/*************************************************************************/
/***************** Other Parameter Translation ***************************/
/*************************************************************************/
/***************** throughput in bits/second *****************************/
                  NTable.connection[k].load.throughput =
                    NTable.connection[k].load.rate*
                      ((double) (NTable.connection[k].load.size*8));
                  NTable.connection[k].perform.rate_alg = FIFO;
                  NTable.connection[k].perform.ordering = TRUE;
                  NTable.connection[k].perform.communication =
                    AParam.relations.communication;
                  NTable.connection[k].perform.cost = NOT_SPECIFIED;
                  NTable.medium[k] = AParam.stream[i].type;

                  printf("QoStranslator: net QoS=(con_id,importance,loss, throughput)=(%
d,%d,%d,%f) \n",
                         k,NTable.connection[k].load.priority,
                         NTable.connection[k].load.loss.loss_rate,
                         NTable.connection[k].load.throughput);

                  k++;
                }/*one-on-one translation*/
              else
                {
/*
                  printf("INTRA IS TRUE \n");
*/
                  for (j=0; j<COMPONENT_NUMBER; j++)
                    {
/*
                      printf("QoStranslator: connection k=%d, AParam.stream[i].component
_spec[j].name = %d,%d,%d \n",
                             k,i,j,AParam.stream[i].component_spec[j].name);
*/
                      if (NTable.status[k] == FREE && AParam.stream[i].component_spec[j]
.name == j)
                        {
/*
                             printf("AParam.stream[i].component_spec[j].size = %d \n", APara
m.stream[i].component_spec[j].size);
*/
/********** DECIDE WHICH PACKET SIZE WILL BE USED FOR A COMPONENT *****/
                          if (AParam.stream[i].component_spec[j].size <= CELL_SIZE)
                            {
                              NTable.connection[k].load.size =CELL_SIZE;
                              NTable.connection[k].load.id =CELL_MODE;
/***** ASSIGN PER COMPONENT ONE NETWORK CONNECTION IF IMPORTANCE DIFFERENT ***/
                              switch(inout)
                                {
                                case INPUT:
                                  NTable.vci[k] = 0x0020 +k;
                                  printf("QoSTranslator/INTRA:<NTable.vci[k],k>=<%d,%d> \
n",
                                         NTable.vci[k],k);
                                  break;
                                case OUTPUT:
                                  NTable.vci[k] = 0x0040 +k;
                                  break;
                                }
                            }
                          else
                            {
                              NTable.connection[k].load.id = DATAGRAM_MODE;
                              switch(inout)
                                {
                                case INPUT:
                                  NTable.vci[k] = 0x4000 +k;
                                  break;
                                case OUTPUT:
                                  NTable.vci[k] = 0x4020 +k;
                                  break;
                                }
                              if ( AParam.stream[i].component_spec[j].size <
                                  DATAGRAM_SIZE)
                                NTable.connection[k].load.size = DATAGRAM_SIZE;
/**** IF COMPONENT IS BIGGER THE DATAGRAM SIZE THEN IT NEEDS TO BE FRGAMENTED ***/
                              else
                                {
                                  AParam.relations.fragmentation[i].medium = i;
                                  AParam.relations.fragmentation[i].component[j].name
                                    = AParam.stream[i].component_spec[j].name;
                                  AParam.relations.fragmentation[i].component[j].frag = T
RUE;
                                  AParam.relations.fragmentation[i].frag = TRUE;
                                  NTable.connection[k].load.size = DATAGRAM_SIZE;
                                }
                            }
                          NTable.status[k] = TAKEN;
/************** RATE TRANSLATION ************************************************/
                          NTable.connection[k].load.rate =
                            transRate(AParam.stream[i].component_spec[j].size,
                                      AParam.stream[i].component_spec[j].rate,
                                      NOT_SPECIFIED, APP_TO_NET,
                                      NTable.connection[k].load.id,
                                      &NumberOfFragments);
                          AParam.relations.fragmentation[i].component[j].number = NumberO
fFragments;
/*************************************************************************/
/* Interarrival time in milisecond - assumption is rate in samples/second    */
/*************************************************************************/
                          NTable.connection[k].load.intermediate_delay =
```

```
                    transInterDelay(AParam.stream[i].component_spec[j].size,
                                    AParam.stream[i].component_spec[j].rate,
                                    APP_TO_NET,
                                    NTable.connection[k].load.id);
/*************************************************************************/
/**** End-to-End Delay Translation (in microseconds )       ***************/
/*************************************************************************/
                  NTable.connection[k].load.end_to_end_delay =
                    1000.0*((double) AParam.stream[i].medium.net_spec.end_to_end
_delay);
                  WhatInfo.i_set[0] = ScheduleInfo;
                  WhatInfo.i_set[1] = NOT_SPECIFIED;
                  WhatInfo.i_set[2] = NOT_SPECIFIED;
/*
                  printf("QoStranslator/INTRA: netRate: %f , netDelay: %f, fragm
ents: %f, connection k = %d, end-to-end delay=%f \n",
                         NTable.connection[k].load.rate,
                         NTable.connection[k].load.intermediate_delay,
                         NumberOfFragments,k,
                         NTable.connection[k].load.end_to_end_delay);
*/
                  RetrieveGlobalState(&SystemState,&Schedule,WhatInfo);
                  for (l=0; l<MEDIA_NUMBER*NUMBER_OF_TASKS_PER_MEDIUM;l++)
                    {
                      if (Schedule.sched[m].sched_queue[l].task_name !=
                          NOT_SPECIFIED  &&
                          (Schedule.sched[m].sched_queue[l].medium == i))
                        {
                          switch(Schedule.sched[m].sched_queue[l].task_name)
                            {
                            case ReadRobotData:
                              getOneTaskParam(&task,ReadRobotData,ROBOT,INPUT);
/*
                              printf("ReadRobotData:QoStranslator:task duration
%d \n",task.duration);
*/
                              break;
                            case WriteRobotData:
                              getOneTaskParam(&task,
                                              WriteRobotData,
                                              ROBOT,OUTPUT);
/*
                              printf("WriteRobotData:QoStranslator:task duration
 %d \n",task.duration);
*/
                              break;
                            case CopyRobotData:
                              getOneTaskParam(&task,CopyRobotData,ROBOT,inout);
                              break;
                            case PointerManagementRobotData:
                              getOneTaskParam(&task,PointerManagementRobotData,R
OBOT,inout);
                              break;
                            case ReadVideoData:
                              getOneTaskParam(&task,
                                              ReadVideoData,
                                              VIDEO,INPUT);
                              break;
                            case WriteVideoData:
                              getOneTaskParam(&task,
                                              WriteVideoData,
                                              VIDEO,OUTPUT);
                              break;
                            case CopyVideoData:
                              getOneTaskParam(&task,CopyVideoData,VIDEO,inout);
                              break;
                            case PointerManagementVideoData:
                              getOneTaskParam(&task,PointerManagementVideoData,VI
DEO,inout);
                              break;
                            case SendCell:
                              getOneTaskParam(&task,SendCell,
                                              NETWORK,INPUT);
/*
                              printf("SendCell:QoStranslator:task duration %d \n"
,task.duration);
*/
                              break;
                            case SendDatagram:
                              getOneTaskParam(&task,SendDatagram,
                                              NETWORK,INPUT);
                              break;
                            case ReceiveCell:
                              getOneTaskParam(&task,ReceiveCell,
                                              NETWORK,OUTPUT);
/*
                              printf("ReceiveCell:QoStranslator:task duration %d
\n",task.duration);
*/
                              break;
                            case ReceiveDatagram:
                              getOneTaskParam(&task,
                                              ReceiveDatagram,
                                              NETWORK,OUTPUT);
                              break;
                            }
                          NTable.connection[k].load.end_to_end_delay =
                            NTable.connection[k].load.end_to_end_delay
                              - ((double) task.duration);
/*
                          printf("QoStranslator:  net delay %f \n",
                                 NTable.connection[k].load.end_to_end_delay);
*/
                        }
                    }
/*************************************************************************/
/*******************   Loss Rate Translation   ***************************/
/*************************************************************************/
                  if (AParam.relations.fragmentation[i].component[j].frag == FALS
E)
                    {
                      NTable.connection[k].load.loss.loss_rate =
                        AParam.stream[i].component_spec[j].loss;
                      /* Loss of two consecutive packets not allowed */
                      NTable.connection[k].load.loss.loss_cons_pkt = FALSE;
                    }
                  else
                    {
                      NTable.connection[k].load.loss.loss_rate =
                        AParam.stream[i].component_spec[j].loss*AParam.relations.
fragmentation[i].number;
                      NTable.connection[k].load.loss.loss_cons_pkt = TRUE;
                    }
/*
                  printf(" Importance = %d \n",AParam.stream[i].component_spec[j]
.importance);
```

```
*/
/*************************************************************************/
/***************** Importance Translation    ****************************/
/*************************************************************************/
                        switch(AParam.stream[i].component_spec[j].importance)
                          {
                          case HIGH_IMPORTANCE: /* prio: 1-5 */
                            NTable.connection[k].load.priority = prio1+1;
                            prio1++;
                            NTable.connection[k].perform.error_alg = FEC;
                            break;
                          case MEDIUM_IMPORTANCE: /* prio: 6-10 */
                            NTable.connection[k].load.priority = prio2+1;
                            prio2++;
                            NTable.connection[k].perform.error_alg = NONE;
                            break;
                          case LOW_IMPORTANCE: /*prio: 11-20 */
                            NTable.connection[k].load.priority = prio3+1;
                            prio3++;
                            NTable.connection[k].perform.error_alg = NONE;
                            break;
                          default: /* best effort, importance not specified prio>20*/
                            NTable.connection[k].load.priority = prio4+1;
                            prio4++;
                            NTable.connection[k].perform.error_alg = NONE;
                            break;
                          }
/*************************************************************************/
/***************** Other Parameter Translation ***************************/
/*************************************************************************/
/***************** throughput in bits/second *****************************/
                        NTable.connection[k].load.throughput =
                          NTable.connection[k].load.rate*
                            ((double) (NTable.connection[k].load.size*8));
                        NTable.connection[k].perform.rate_alg = FIFO;
                        NTable.connection[k].perform.ordering = TRUE;
                        NTable.connection[k].perform.communication =
                          AParam.relations.communication;
                        NTable.connection[k].perform.cost = NOT_SPECIFIED;
                        NTable.medium[k] = AParam.stream[i].type;
/*
                        printf("QoStranslator/INTRA: net QoS=(con_id,importance,loss,
throughput)=(%d,%d,%d,%f) \n",
                               k,NTable.connection[k].load.priority,
                               NTable.connection[k].load.loss.loss_rate,
                               NTable.connection[k].load.throughput);
*/

                        k++;
                      }
                    }
                  }/*one-to-many translation*/
              }/* if medium defined */
            else
              {
                if (NTable.status[k] == TAKEN)
                  {
                    k++;
                  }
              }
          }/*for i */
      setNetQoS(&NTable,inout);
      case NET_TO_APP:
        break;
      }

}
/*************************************************************************/
/* SetNetQoS procedure sets network qos in Network QoS Profile           */
/*************************************************************************/

int setNetQoS(Param,inout)
NET_QOS_TABLE *Param;
int inout;
{
  int fd,n_bytes;
  openProfile(&fd,"netQoSprofile");
  switch(inout)
    {
    case INPUT:
      lseek(fd,0L,0);
      if ((n_bytes=write(fd,(char *)(Param),sizeof(NET_QOS_TABLE)))== -1)
        {
          perror("setAppQoS: write QoS input to profile  failure");
          return(-1);
        }
      closeProfile(fd);
      return(0);
      break;
    case OUTPUT:
      lseek(fd,sizeof(NET_QOS_TABLE),0);
      if ((n_bytes=write(fd,(char *)(Param),sizeof(NET_QOS_TABLE)))== -1)
        {
          perror("setAppQoS: write QoS input to profile  failure");
          return(-1);
        }
      closeProfile(fd);
      return(0);
      break;
    }
}
/*************************************************************************/
/* GetNetQoS procedure gets network QoS from network QoS profile         */
/*************************************************************************/

int  getNetQoS(Param,inout)
NET_QOS_TABLE *Param;
int inout;
{
  int fd;
  openProfile(&fd, "netQoSprofile");
  if (inout == INPUT)
    {
      lseek(fd,0L,0); /*position at the beginning of the file */
      if (read(fd,(char *)(Param),sizeof(NET_QOS_TABLE))== -1)
        {
          perror("getNetQoS: read QoS input to profile  failure");
          return(-1);
        }
      closeProfile(fd);
      return(0);
    }
  if (inout == OUTPUT)
    {
      /* position  after the input application QoS from 0 of the file */
      lseek(fd,sizeof(NET_QOS_TABLE),0);
```

```
      if (read(fd,(char *)Param,sizeof(NET_QOS_TABLE))== -1)
        {
          perror("getNetQoS: read QoS output  to profile  failure");
          return(-1);
        }
      closeProfile(fd);
      return(0);
    }
}

/*************************************************************************/
/* AdmitNetQoS - admits network QoS - and computes global schedule       */
/*************************************************************************/

int AdmitNetQoS(Param,Notification,inout,side)
NET_QOS_TABLE *Param; /* net QoS for the direction 'inout' */
NOTIFY  *Notification;
int inout;
int side;{
  NET_QOS_TABLE oParam; /* net QoS for the opposite 'inout' */
  GLOBAL_STATE SystemState;
  RATE_MONOTONIC_SCHEDULER Scheduler;
  INFO_STATE WhatInfo;
  int i,j, k,jj, NumberOfMedia,mediumType;
  int PreviousTasks;
  TASK task;
  double through1,through2;
  BOOLEAN Scheduled,Preempted,FirstRun,RENEG;
  APP_QOS AParam;

/*************************************************************************/
/* SCHEDULABILITY TEST                                                   */
/*************************************************************************/
  NumberOfMedia=0;
  PreviousTasks = 0;
  Scheduled = FALSE;
  Preempted = FALSE;
  FirstRun = TRUE;
  RENEG = FALSE;
  WhatInfo.i_set[0]=SystemStateInfo;
  WhatInfo.i_set[1]=ScheduleInfo;
  RetrieveGlobalState(&SystemState,&Scheduler,WhatInfo);
  getAppQoS(&AParam,inout);
/*
  printf("AdmitNetQoS: Param->connection[0].load.end_to_end_delay= %f \n",
         Param->connection[0].load.end_to_end_delay);
*/
  k=0;/* count of network connections */

  for (i=1;i<MEDIA_NUMBER;i++)
    {
      if (AParam.stream[i].type == i)
        {
          NumberOfMedia = NumberOfMedia + 1;
        }
    }

  printf("Number of Media = %d \n", NumberOfMedia);


  for (i=0; i<Scheduler.number_of_ticks;i++)
    {
      for (j=1;j<MEDIA_NUMBER*NUMBER_OF_TASKS_PER_MEDIUM;j++)
        {
          if (Scheduled == TRUE && i>0 )
            {
              for (jj =0; jj < PreviousTasks; jj++)
                {
                  Scheduler.sched[i].sched_queue[jj].medium =
                    Scheduler.sched[0].sched_queue[jj].medium;
                  Scheduler.sched[i].sched_queue[jj].inout =
                    Scheduler.sched[0].sched_queue[jj].inout;
                  Scheduler.sched[i].sched_queue[jj].task_name =
                    Scheduler.sched[0].sched_queue[jj].task_name;
                  Scheduler.sched[i].sched_queue[jj].task_duration =
                    Scheduler.sched[0].sched_queue[jj].task_duration;
                  Scheduler.sched[i].sched_queue[jj].time_begin =
                    Scheduler.sched[i-1].sched_queue[jj].time_begin
                      + Scheduler.min_period;
                  Scheduler.sched[i].sched_queue[jj].time_deadline =
                    i*Scheduler.sched[0].sched_queue[jj].time_deadline;
/*
                  printf("AdmitNetQoS: sched=i,jj,inout,taskname,taskbegin,medium,deadlin
e,duration (%d,%d,%d,%d,%d,%d,%d, %d)\n",i,jj,
                                     Scheduler.sched[i].sched_queue[jj].inout,
                                     Scheduler.sched[i].sched_queue[jj].task_name,
                                     Scheduler.sched[i].sched_queue[jj].time_begin,
                                     Scheduler.sched[i].sched_queue[jj].medium,
                                     Scheduler.sched[i].sched_queue[jj].time_deadline,
                                     Scheduler.sched[i].sched_queue[jj].task_duration);
*/
                }
              j=MEDIA_NUMBER*NUMBER_OF_TASKS_PER_MEDIUM;
            }
          if (Scheduler.sched[i].sched_queue[j].task_name == NOT_SPECIFIED
              && Scheduled==FALSE)
            {
              if (Scheduler.sched[i].sched_queue[0].medium == ROBOT &&
                  FirstRun == TRUE)
                {
                  PreviousTasks = j-1;
                  FirstRun = FALSE;
/*
                  printf("PreviousTasks = %d \n",PreviousTasks);
*/
                }
              /********************** CONNECTION IS ALLOCATED ************************
*/
              if (Param->status[k] == TAKEN)
                {
                  mediumType = Param->medium[k];
                  AParam.stream[mediumType].medium.net_spec.count_con = k+1;
                  printf("medium = %d, con k=%d AdmitNetQoS: number of connections =%d \n
",
                         Param->medium[k],k,AParam.stream[mediumType].medium.net_spec.cou
nt_con);
                  /********************** CELL_MODE **********************************
*****/
                  if (Param->connection[k].load.id ==CELL_MODE)
                    {
                      switch(inout)
                        {
                        case INPUT:
                          Scheduler.sched[i].sched_queue[j].medium =
                            Scheduler.sched[i].sched_queue[j-1].medium;
                          Scheduler.sched[i].sched_queue[j].task_name =
                            SendCell;
                          Scheduler.sched[i].sched_queue[j].time_begin =
```

```
  Scheduler.sched[i].sched_queue[j-1].time_begin +
    Scheduler.sched[i].sched_queue[j-1].task_duration;
Scheduler.sched[i].sched_queue[j].inout = INPUT;

/*************************************************************
*************/

/* Check schedulability

*/

/*************************************************************
*************/

getOneTaskParam(&task,
                Scheduler.sched[i].sched_queue[j].task_name,
                NETWORK,
                Scheduler.sched[i].sched_queue[j].inout);
/*****************************TASK NOT SCHEDULABLE IN THE PER
IOD-MUST PREEMPT**/

if (Scheduler.sched[i].sched_queue[j].time_begin +
    task.duration > (i+1)*Scheduler.min_period)
  {
    /****************************** CHECK END-TO-END DELAY **
******************/

    switch(side)
      {
      case BUYER:
        if (Scheduler.sched[i].sched_queue[j].time_begin +
            task.duration > ((long) Param->connection[k].load.
end_to_end_delay))
          {
            Notification->note = NEG_REJECT;
            Notification->reason = END_TO_END_TEST_FAILURE;
            /* Connection is released */
            releaseConnection(Param,k);
            freeSchedResources(&Scheduler,i,j);
            for (jj=0;jj<j; jj++)
              {
                if (Scheduler.sched[i].sched_queue[jj].medium
== Param->medium[k])
                  {
                    freeSchedResources(&Scheduler,i,jj);
                  }
              }
            Param->medium[k] = 0;
            setNetQoS(Param,inout);
            WhatInfo.i_set[0] = ScheduleInfo;
            WhatInfo.i_set[1] = NOT_SPECIFIED;
            StoreGlobalState(SystemState,Scheduler,WhatInfo);
            return(BAD_VALUE);
          }
        break;
      case SELLER:
        if (Scheduler.sched[i].sched_queue[j].time_begin +
            task.duration + NET_CELL_DELAY >
            ((long) Param->connection[k].load.end_to_end_delay
))
          {
            Notification->note = NEG_REJECT;
            Notification->reason = END_TO_END_TEST_FAILURE;
            /* Connection is released */
            releaseConnection(Param,k);
            freeSchedResources(&Scheduler,i,j);
            for (jj=0;jj<j; jj++)
              {
                if (Scheduler.sched[i].sched_queue[jj].medium
== Param->medium[k])
                  {
                    freeSchedResources(&Scheduler,i,jj);
                  }
              }
            Param->medium[k] = 0;
            setNetQoS(Param,inout);
            WhatInfo.i_set[0] = ScheduleInfo;
            WhatInfo.i_set[1] = NOT_SPECIFIED;
            StoreGlobalState(SystemState,Scheduler,WhatInfo);
            return(BAD_VALUE);
          }
        break;
      }
    /**** eliminate the task from this interval,
      preempt the task  ***/
  }
else
  /**********************TASK SCHEDULABLE IN THE PERIOD *******
************/

  {
    /*************************************************************
***************/

    /* Check end-to-end delay

    /*************************************************************
***************/

    switch(side)
      {
      case BUYER:
        if (Scheduler.sched[i].sched_queue[j].time_begin +
            task.duration <
            ((long) Param->connection[k].load.end_to_end_delay)
)
          {
            Scheduler.sched[i].sched_queue[j].task_duration = t
ask.duration;

            /**** test is positive,
              go to next connection ****/
            k++;
            printf("AdmitNetQoS(BUYER): increased k\n");

          }
        else
          {
            Notification->note = NEG_REJECT;
            Notification->reason = END_TO_END_TEST_FAILURE;
            /* Connection is released */
            releaseConnection(Param,k);
            freeSchedResources(&Scheduler,i,j);

            for (jj=0;jj<j; jj++)
              {
                if (Scheduler.sched[i].sched_queue[jj].medium =
= Param->medium[k])
                  {
                    freeSchedResources(&Scheduler,i,jj);
                  }
              }
            Param->medium[k] = 0;
            setNetQoS(Param,inout);
            WhatInfo.i_set[0] = ScheduleInfo;
            WhatInfo.i_set[1] = NOT_SPECIFIED;
            StoreGlobalState(SystemState,Scheduler,WhatInfo);
            return(BAD_VALUE);
```

```
                    }
                  break;
                case SELLER:
                  if (Scheduler.sched[i].sched_queue[j].time_begin +
                      task.duration + NET_CELL_DELAY <
                      ((long) Param->connection[k].load.end_to_end_delay
))
                    {
                      Scheduler.sched[i].sched_queue[j].task_duration =
task.duration;

                      /**** test is positive,
                        go to next connection ****/
                      k++;
/*
                      printf("AdmitNetQoS(SELLER): increased k\n");
*/
                    }
                  else
                    {
                      Notification->note = NEG_REJECT;
                      Notification->reason = END_TO_END_TEST_FAILURE;
                      /* Connection is released */
                      releaseConnection(Param,k);
                      freeSchedResources(&Scheduler,i,j);

                      for (jj=0;jj<j; jj++)
                        {
                          if (Scheduler.sched[i].sched_queue[jj].medium
== Param->medium[k])
                            {
                              freeSchedResources(&Scheduler,i,jj);
                            }
                        }
                      Param->medium[k] = 0;
                      setNetQoS(Param,inout);
                      WhatInfo.i_set[0] = ScheduleInfo;
                      WhatInfo.i_set[1] = NOT_SPECIFIED;
                      StoreGlobalState(SystemState,Scheduler,WhatInfo);
                      return(BAD_VALUE);
                    }
                  break;
                }
            }
          break;
        case OUTPUT:
          /******************** switch application task with network ta
sks *************/
          Scheduler.sched[i].sched_queue[j].medium =
            Scheduler.sched[i].sched_queue[j-1].medium;
          Scheduler.sched[i].sched_queue[j].task_name =
            Scheduler.sched[i].sched_queue[j-1].task_name;
          Scheduler.sched[i].sched_queue[j-1].task_name =
            ReceiveCell;
          Scheduler.sched[i].sched_queue[j].inout =
            Scheduler.sched[i].sched_queue[j-1].inout;
          Scheduler.sched[i].sched_queue[j-1].inout = OUTPUT;
          Scheduler.sched[i].sched_queue[j].task_duration =
            Scheduler.sched[i].sched_queue[j-1].task_duration;
          /************************************************************
*************/
          /* Check schedulability
*/
          /************************************************************
*************/
          getOneTaskParam(&task,
                          Scheduler.sched[i].sched_queue[j-1].task_name,
                          NETWORK,
                          Scheduler.sched[i].sched_queue[j-1].inout);
          Scheduler.sched[i].sched_queue[j].time_begin =
            Scheduler.sched[i].sched_queue[j-1].time_begin +
              task.duration;
          Scheduler.sched[i].sched_queue[j-1].task_duration =
            task.duration;
          /******************** TASK NOT SCHEDULABLE IN THE MIN PERIOD-M
UST PREEMPT ***/
          if (Scheduler.sched[i].sched_queue[j].time_begin +
              Scheduler.sched[i].sched_queue[j].task_duration > (i+1)*Sch
eduler.min_period)
            {
              /****************************** CHECK END-TO-END DELAY ***
****************/
              switch(side)
                {
                case BUYER:
                  if (Scheduler.sched[i].sched_queue[j].time_begin +
                      Scheduler.sched[i].sched_queue[j].task_duration > (
(long) Param->connection[k].load.end_to_end_delay))
                    {
                      Notification->note = NEG_REJECT;
                      Notification->reason = END_TO_END_TEST_FAILURE;
                      /* Connection is released */
                      releaseConnection(Param,k);
                      freeSchedResources(&Scheduler,i,j);

                      for (jj=0;jj<j; jj++)
                        {
                          if (Scheduler.sched[i].sched_queue[jj].medium =
= Param->medium[k])
                            {
                              freeSchedResources(&Scheduler,i,jj);
                            }
                        }
                      Param->medium[k] = 0;
                      setNetQoS(Param,inout);
                      WhatInfo.i_set[0] = ScheduleInfo;
                      WhatInfo.i_set[1] = NOT_SPECIFIED;
                      StoreGlobalState(SystemState,Scheduler,WhatInfo);
                      return(BAD_VALUE);
                    }
                  break;
                case SELLER:
                  if (Scheduler.sched[i].sched_queue[j].time_begin +
                      Scheduler.sched[i].sched_queue[j].task_duration + N
ET_CELL_DELAY > ((long) Param->connection[k].load.end_to_end_delay))
                    {
                      Notification->note = NEG_REJECT;
                      Notification->reason = END_TO_END_TEST_FAILURE;
                      /* Connection is released */
                      releaseConnection(Param,k);
                      freeSchedResources(&Scheduler,i,j);

                      for (jj=0;jj<j; jj++)
                        {
                          if (Scheduler.sched[i].sched_queue[jj].medium =
= Param->medium[k])
                            {
                              freeSchedResources(&Scheduler,i,jj);
                            }
```

```
                    }
                  Param->medium[k] = 0;
                  setNetQoS(Param,inout);
                  WhatInfo.i_set[0] = ScheduleInfo;
                  WhatInfo.i_set[1] = NOT_SPECIFIED;
                  StoreGlobalState(SystemState,Scheduler,WhatInfo);
                  return(BAD_VALUE);
                }
              break;
            }
          /**** eliminate the task j  from this interval,
            preempt the task ***/
        }
      else
        /****************** TASK SCHEDULABLE IN MIN PERIOD *********
*************/
        {
          /*********************************************************
****************/
          /* Check end-to-end delay
                */
          /*********************************************************
****************/
          switch(side)
            {
            case BUYER:
              if (Scheduler.sched[i].sched_queue[j].time_begin +
                  Scheduler.sched[i].sched_queue[j].task_duration <
                  ((long) Param->connection[k].load.end_to_end_delay
))
                {

                  /**** test is positive,
                    go to next connection ****/
                  k++;

                  printf("AdmitNetQoS(BUYER): increased k\n");

                }
              else
                {
                  Notification->note = NEG_REJECT;
                  Notification->reason = END_TO_END_TEST_FAILURE;
                  /* Connection is released */
                  releaseConnection(Param,k);
                  freeSchedResources(&Scheduler,i,j);

                  for (jj=0;jj<j; jj++)
                    {
                      if (Scheduler.sched[i].sched_queue[jj].medium
== Param->medium[k])
                        {
                          freeSchedResources(&Scheduler,i,jj);
                        }
                    }
                  Param->medium[k] = 0;
                  setNetQoS(Param,inout);
                  WhatInfo.i_set[0] = ScheduleInfo;
                  WhatInfo.i_set[1] = NOT_SPECIFIED;
                  StoreGlobalState(SystemState,Scheduler,WhatInfo);
                  return(BAD_VALUE);
                }
              break;
            case SELLER:
```

```
                                                                    )

                                                                    /*

                                                                    */

                                                                    = Param->medium[k])

                                                                    **********/

                                                                    ************/

                                                                    */

                                                                    ************/
              if (Scheduler.sched[i].sched_queue[j].time_begin +
                  task.duration + NET_CELL_DELAY <
                  ((long) Param->connection[k].load.end_to_end_delay)

                {
                  /**** test is positive,
                    go to next connection ****/
                  k++;

                  printf("AdmitNetQoS(SELLER): increased k\n");

                }
              else
                {
                  Notification->note = NEG_REJECT;
                  Notification->reason = END_TO_END_TEST_FAILURE;
                  /* Connection is released */
                  releaseConnection(Param,k);
                  freeSchedResources(&Scheduler,i,j);

                  for (jj=0;jj<j; jj++)
                    {
                      if (Scheduler.sched[i].sched_queue[jj].medium =

                        {
                          freeSchedResources(&Scheduler,i,jj);
                        }
                    }
                  Param->medium[k] = 0;
                  setNetQoS(Param,inout);
                  WhatInfo.i_set[0] = ScheduleInfo;
                  WhatInfo.i_set[1] = NOT_SPECIFIED;
                  StoreGlobalState(SystemState,Scheduler,WhatInfo);
                  return(BAD_VALUE);
                }
              break;
            }
        }
      else
        {
          /********************** DATAGRAM_MODE ************************
          switch(inout)
            {
            case INPUT:
              Scheduler.sched[i].sched_queue[j].medium =
                Scheduler.sched[i].sched_queue[j-1].medium;
              Scheduler.sched[i].sched_queue[j].task_name =
                SendDatagram;
              Scheduler.sched[i].sched_queue[j].time_begin =
                Scheduler.sched[i].sched_queue[j-1].time_begin +
                  Scheduler.sched[i].sched_queue[j-1].task_duration;;
              Scheduler.sched[i].sched_queue[j].inout = INPUT;
              /*************************************************************

              /* Check schedulability

              /*************************************************************

              getOneTaskParam(&task,
                              Scheduler.sched[i].sched_queue[j].task_name,
```

```
                                NETWORK,
                                Scheduler.sched[i].sched_queue[j].inout);


                    printf("AdmitNetQoS(DATAGRAM,INPUT): Scheduler.sched[i].sch  = Param->medium[k])
ed_queue[j].time_begin = %d,task.duration=%d, Param->connection[k].load.end_to_end_delay
 = %f,<i,j,k>=<%d,%d,%d> \n",
                            Scheduler.sched[i].sched_queue[j].time_begin,
                            task.duration,
                            Param->connection[k].load.end_to_end_delay,i,j,k);



                  /*************** TASK NOT SCHEDULABLE IN MIN PERIOD - MUST PRE
EMPT *********/
                  if (Scheduler.sched[i].sched_queue[j].time_begin +
                      task.duration > (i+1)*Scheduler.min_period)
                    {
                      printf("Task not schedulable in min period \n");
                      /****************************** CHECK END-TO-END DELAY **
******************/
                      switch(side)
                        {
                        case BUYER:
                          if (Scheduler.sched[i].sched_queue[j].time_begin +
                              task.duration > ((long) Param->connection[k].load.
end_to_end_delay))
                            {
                              Notification->note = NEG_REJECT;
                              Notification->reason = END_TO_END_TEST_FAILURE;
                              /* Connection is released */
                              releaseConnection(Param,k);
                              freeSchedResources(&Scheduler,i,j);

                              for (jj=0;jj<j; jj++)
                                {
                                  if (Scheduler.sched[i].sched_queue[jj].medium
== Param->medium[k])
                                    {
                                      freeSchedResources(&Scheduler,i,jj);
                                    }
                                }
                              Param->medium[k] = 0;
                              setNetQoS(Param,inout);
                              WhatInfo.i_set[0] = ScheduleInfo;
                              WhatInfo.i_set[1] = NOT_SPECIFIED;
                              StoreGlobalState(SystemState,Scheduler,WhatInfo);
                              return(BAD_VALUE);
                            }
                          /**** eliminate the task from this interval,
                            preempt the task ***/
                          break;
                        case SELLER:
                          if (Scheduler.sched[i].sched_queue[j].time_begin +
                              task.duration + NET_DATAGRAM_DELAY >
                              ((long) Param->connection[k].load.end_to_end_delay
))
                            {
                              Notification->note = NEG_REJECT;
                              Notification->reason = END_TO_END_TEST_FAILURE;

                              /* Connection is released */
                              releaseConnection(Param,k);
                              freeSchedResources(&Scheduler,i,j);

                              for (jj=0;jj<j; jj++)
                                {
                                  if (Scheduler.sched[i].sched_queue[jj].medium =
                                    {
                                      freeSchedResources(&Scheduler,i,jj);
                                    }
                                }
                              Param->medium[k] = 0;
                              setNetQoS(Param,inout);
                              WhatInfo.i_set[0] = ScheduleInfo;
                              WhatInfo.i_set[1] = NOT_SPECIFIED;
                              StoreGlobalState(SystemState,Scheduler,WhatInfo);
                              return(BAD_VALUE);
                            }
                          /**** eliminate the task from this interval,
                            preempt the task ***/
                          break;
                        }
                    }
                  else
                    /************************** TASK SCHEDULABLE IN MIN PERIOD
************/
                    {
                      printf("Task schedulable in min.period \n");
                      /*************************************************************
***************/
                      /* Check end-to-end delay
                                                                  */
                      /*************************************************************
***************/
                      switch(side)
                        {
                        case BUYER:
                          if (Scheduler.sched[i].sched_queue[j].time_begin +
                              task.duration <
                              ((long) Param->connection[k].load.end_to_end_delay)
)
                            {
                              Scheduler.sched[i].sched_queue[j].task_duration = t
ask.duration;
                              /**** test is positive, go to next task ****/
                              k++;

                              printf("AdmitNetQoS(BUYER/INPUT): task schedulable
in min period, EED positive, increased k\n");

                            }
                          else
                            {
                              printf("Buyer/Input: Task schedulable in min period
 but EED not satisfied \n");
                              Notification->note = NEG_REJECT;
                              Notification->reason = END_TO_END_TEST_FAILURE;
                              /* Connection is released */
                              releaseConnection(Param,k);
                              freeSchedResources(&Scheduler,i,j);

                              for (jj=0;jj<j; jj++)
                                {
                                  if (Scheduler.sched[i].sched_queue[jj].medium =
= Param->medium[k])
                                    {
```

```
                        freeSchedResources(&Scheduler,i,jj);
                      }
                    }
                  Param->medium[k] = 0;
                  setNetQoS(Param,inout);
                  WhatInfo.i_set[0] = ScheduleInfo;
                  WhatInfo.i_set[1] = NOT_SPECIFIED;
                  StoreGlobalState(SystemState,Scheduler,WhatInfo);
                  return(BAD_VALUE);
                }
              break;
            case SELLER:
              if (Scheduler.sched[i].sched_queue[j].time_begin +
                  task.duration + NET_DATAGRAM_DELAY <
                  ((long) Param->connection[k].load.end_to_end_delay
))
                {
                  Scheduler.sched[i].sched_queue[j].task_duration =
task.duration;

                  /**** test is positive, go to next task ****/
                  k++;

/*
                  printf("AdmitNetQoS(SELLER): increased k\n");
*/
                }
              else
                {
                  Notification->note = NEG_REJECT;
                  Notification->reason = END_TO_END_TEST_FAILURE;
                  /* Connection is released */
                  releaseConnection(Param,k);
                  freeSchedResources(&Scheduler,i,j);

                  for (jj=0;jj<j;jj++)
                    {
                      if (Scheduler.sched[i].sched_queue[jj].medium
== Param->medium[k])
                        {
                          freeSchedResources(&Scheduler,i,jj);
                        }
                    }
                  Param->medium[k] = 0;
                  setNetQoS(Param,inout);
                  WhatInfo.i_set[0] = ScheduleInfo;
                  WhatInfo.i_set[1] = NOT_SPECIFIED;
                  StoreGlobalState(SystemState,Scheduler,WhatInfo);
                  return(BAD_VALUE);
                }
              break;
            }
        }

      break; /* INPUT*/
    case OUTPUT:
      /******************** switch application task with network ta
sks *************/

      Scheduler.sched[i].sched_queue[j].medium =
        Scheduler.sched[i].sched_queue[j-1].medium;
      Scheduler.sched[i].sched_queue[j].task_name =
        Scheduler.sched[i].sched_queue[j-1].task_name;
      Scheduler.sched[i].sched_queue[j-1].task_name =
        ReceiveDatagram;
      Scheduler.sched[i].sched_queue[j].inout =
        Scheduler.sched[i].sched_queue[j-1].inout;
      Scheduler.sched[i].sched_queue[j-1].inout = OUTPUT;
      Scheduler.sched[i].sched_queue[j].task_duration =
        Scheduler.sched[i].sched_queue[j-1].task_duration;

      /*****************************************************************
************/
      /* Check schedulability
*/
      /*****************************************************************
************/
      getOneTaskParam(&task,
                      Scheduler.sched[i].sched_queue[j-1].task_name,
                      NETWORK,
                      Scheduler.sched[i].sched_queue[j-1].inout);

      printf("AdmitNetQoS(DATAGRAM,OUTPUT): Scheduler.sched[i].sched_
queue[j].time_begin = %d,task.duration=%d, Param->connection[k].load.end_to_end_delay = %
f,i=%d,j=%d,k = %d \n",
              Scheduler.sched[i].sched_queue[j].time_begin,
              task.duration,
              Param->connection[k].load.end_to_end_delay,i,j,k);

      Scheduler.sched[i].sched_queue[j].time_begin =
        Scheduler.sched[i].sched_queue[j-1].time_begin +
          task.duration;
      Scheduler.sched[i].sched_queue[j-1].task_duration =
        task.duration;
      /***************** TASK NOT SCHEDULABLE IN MIN PERIOD - MUST PR
EEMPT ******/
      if (Scheduler.sched[i].sched_queue[j].time_begin +
          Scheduler.sched[i].sched_queue[j].task_duration > (i+1)*Sch
eduler.min_period)
        {
          printf("OUTPUT: task not schedulable in min period \n");
          /****************************** CHECK END-TO-END DELAY ***
*****************/

          switch(side)
            {
            case BUYER:
              if (Scheduler.sched[i].sched_queue[j].time_begin +
                  Scheduler.sched[i].sched_queue[j].task_duration > (
(long) Param->connection[k].load.end_to_end_delay))
                {
                  printf("BUYER/OUTPUT: EED not satisfied \n");
                  Notification->note = NEG_REJECT;
                  Notification->reason = END_TO_END_TEST_FAILURE;
                  /* Connection is released */
                  releaseConnection(Param,k);
                  freeSchedResources(&Scheduler,i,j);

                  for (jj=0;jj<j;jj++)
                    {
                      if (Scheduler.sched[i].sched_queue[jj].medium =
= Param->medium[k])
                        {
                          freeSchedResources(&Scheduler,i,jj);
                        }
                    }
                  Param->medium[k] = 0;
                  setNetQoS(Param,inout);
                  WhatInfo.i_set[0] = ScheduleInfo;
                  WhatInfo.i_set[1] = NOT_SPECIFIED;
```

```
                    StoreGlobalState(SystemState,Scheduler,WhatInfo);
                    return(BAD_VALUE);
                  }
                 /**** eliminate the task from this interval,
                   preempt the task***/
                 break;
               case SELLER:
                 if (Scheduler.sched[i].sched_queue[j].time_begin +
                     Scheduler.sched[i].sched_queue[j].task_duration +
NET_DATAGRAM_DELAY > ((long) Param->connection[k].load.end_to_end_delay))
                   {
                     Notification->note = NEG_REJECT;
                     Notification->reason = END_TO_END_TEST_FAILURE;
                     /* Connection is released */
                     releaseConnection(Param,k);
                     freeSchedResources(&Scheduler,i,j);
                     for (jj=0;jj<j; jj++)
                       {
                         if (Scheduler.sched[i].sched_queue[jj].medium
== Param->medium[k])
                           {
                             freeSchedResources(&Scheduler,i,jj);
                           }
                       }
                     Param->medium[k] = 0;
                     setNetQoS(Param,inout);
                     WhatInfo.i_set[0] = ScheduleInfo;
                     WhatInfo.i_set[1] = NOT_SPECIFIED;
                     StoreGlobalState(SystemState,Scheduler,WhatInfo);
                     return(BAD_VALUE);
                   }
                 /**** eliminate the task from this interval,
                   preempt the task***/
                 break;
               }
             Notification->note = NEG_REJECT;
             Notification->reason = SCHEDULE_NOT_FEASIBLE;
             /* Connection is released */
             releaseConnection(Param,k);
             freeSchedResources(&Scheduler,i,j);
             for (jj=0;jj<j; jj++)
               {
                 if (Scheduler.sched[i].sched_queue[jj].medium == Param
->medium[k])
                   {
                     freeSchedResources(&Scheduler,i,jj);
                   }
               }
             Param->medium[k] = 0;
             setNetQoS(Param,inout);
             WhatInfo.i_set[0] = ScheduleInfo;
             WhatInfo.i_set[1] = NOT_SPECIFIED;
             StoreGlobalState(SystemState,Scheduler,WhatInfo);
             return(BAD_VALUE);
           }
         else
           /**************** TASK SCHEDULABLE IN MIN PERIOD ***********
**************/
           {
             printf("OUTPUT: task schedulable in min period \n");
             /*********************************************************
****************/
             /* Check end-to-end delay
                */
```

```
**************/

                )

= Param->medium[k])

ET_DATAGRAM_DELAY <

)
```

```
/*********************************************************

switch(side)
  {
  case BUYER:
    if (Scheduler.sched[i].sched_queue[j].time_begin +
        Scheduler.sched[i].sched_queue[j].task_duration <
        ((long) Param->connection[k].load.end_to_end_delay)

      {
        printf("BUYER/OUTPUT: EED satisfied \n");
        /**** test is positive, go to next
          connection ****/
        k++;

        printf("AdmitNetQoS(BUYER): increased k\n");

      }
    else
      {
        printf("BUYER/OUTPUT: EED not satisfied \n");
        Notification->note = NEG_REJECT;
        Notification->reason = END_TO_END_TEST_FAILURE;
        /* Connection is released */
        releaseConnection(Param,k);
        freeSchedResources(&Scheduler,i,j);

        for (jj=0;jj<j; jj++)
          {
            if (Scheduler.sched[i].sched_queue[jj].medium =

              {
                freeSchedResources(&Scheduler,i,jj);
              }
          }
        Param->medium[k] = 0;
        setNetQoS(Param,inout);
        WhatInfo.i_set[0] = ScheduleInfo;
        WhatInfo.i_set[1] = NOT_SPECIFIED;
        StoreGlobalState(SystemState,Scheduler,WhatInfo);
        return(BAD_VALUE);
      }
    break;
  case SELLER:
    if (Scheduler.sched[i].sched_queue[j].time_begin +
        Scheduler.sched[i].sched_queue[j].task_duration + N

        ((long) Param->connection[k].load.end_to_end_delay)

      {
        /**** test is positive, go to next
          connection  ****/
        k++;

        printf("AdmitNetQoS(SELLER): increased k\n");

      }
    else
      {
        Notification->note = NEG_REJECT;
        Notification->reason = END_TO_END_TEST_FAILURE;
        /* Connection is released */
        releaseConnection(Param,k);
        freeSchedResources(&Scheduler,i,j);
```

```
                                    for (jj=0;jj<j; jj++)
                                      {
                                        if (Scheduler.sched[i].sched_queue[jj].medium
== Param->medium[k])
                                          {
                                            freeSchedResources(&Scheduler,i,jj);
                                          }
                                      }
                                    Param->medium[k] = 0;
                                    setNetQoS(Param,inout);
                                    WhatInfo.i_set[0] = ScheduleInfo;
                                    WhatInfo.i_set[1] = NOT_SPECIFIED;
                                    StoreGlobalState(SystemState,Scheduler,WhatInfo);
                                    return(BAD_VALUE);
                                  }
                                break;
                              }
                            }
                          break;
                        }
                      } /*else*/
                  } /*if connection taken*/
                else
                  /********************** CONNECTION IS FREE ***************************
*/
                  {

                    printf("AppNetQoS:Finished with  i=%d interval\n",i);

                    if (Preempted == FALSE)
                      {
                        Scheduled = TRUE;
                      }
                    k=0;
                    j=MEDIA_NUMBER*NUMBER_OF_TASKS_PER_MEDIUM;
                  }
              } /* if sched slot free */
          }/*for j */
      }/*for i*/

    for (i=0;i<Scheduler.number_of_ticks;i++)
      {
        for (j=0;j<MEDIA_NUMBER*NUMBER_OF_TASKS_PER_MEDIUM;j++)
          {
            if (Scheduler.sched[i].sched_queue[j].task_name!=NOT_SPECIFIED)
              {
                printf("AdmitNetQoS:medium,taskname, taskbegin, task duration= %d,%d,%d, %
d \n",
                       Scheduler.sched[i].sched_queue[j].medium,
                       Scheduler.sched[i].sched_queue[j].task_name,
                       Scheduler.sched[i].sched_queue[j].time_begin,
                       Scheduler.sched[i].sched_queue[j].task_duration);
              }
            else
              {
                if (RENEG == FALSE)
                  {
                    Scheduler.sched[i].sched_queue[j].medium = RENEG_INFO;
                    Scheduler.sched[i].sched_queue[j].task_name = Renegotiate;
                    Scheduler.sched[i].sched_queue[j].time_begin =
                      Scheduler.sched[i].sched_queue[j-1].time_begin +
                        Scheduler.sched[i].sched_queue[j-1].task_duration;
                    Scheduler.sched[i].sched_queue[j].task_duration = 5;
                    printf("AdmitNetQoS:medium,taskname, taskbegin, task duration= %d,%d,%d
, %d \n",
                           Scheduler.sched[i].sched_queue[j].medium,
                           Scheduler.sched[i].sched_queue[j].task_name,
                           Scheduler.sched[i].sched_queue[j].time_begin,
                           Scheduler.sched[i].sched_queue[j].task_duration);
                    RENEG = TRUE;
                  }
              }
          }
      }
    WhatInfo.i_set[0] = ScheduleInfo;
    WhatInfo.i_set[1] = NOT_SPECIFIED;
    StoreGlobalState(SystemState,Scheduler,WhatInfo);
    setAppQoS(&AParam,inout);
  /*************************************************************************/
  /*  THROUGPUT TEST (BANDWIDTH TEST)                                      */
  /*************************************************************************/

  /************************ GET OPPOSITE DIRECTION NET QOS ***************/
    if (inout == INPUT)
        getNetQoS(&oParam,OUTPUT);
    else
        getNetQoS(&oParam,INPUT);
  /********************** GET THROUGHPUT FOR ONE DIRECTION ***************/
    through1 = 0.0;
    through2 = 0.0;
    getThroughput(Param,&through1);
    getThroughput(&oParam,&through2);
  /********************** TEST AGAINST 130 000 000 bits/second ************/

    if (through1+through2 < 130000000.0)
      {
        Notification->note = NEG_SUCCESS;
      }
    else
      {
  /*
        printf("throughput test failed \n");
  */
        Notification->note = NEG_REJECT;
        Notification->reason = THROUGHPUT_TEST_FAILURE;
        bzero((char *)(Param), sizeof(NET_QOS_TABLE));
        setNetQoS(Param,inout);
        return(BAD_VALUE);
      }

  }

  /*************************************************************************/
  /* Negotiation of Network QoS                                            */
  /*************************************************************************/

  int negotiateNetQoS(param,notification,side,inout)
  NET_QOS_TABLE *param;
  NOTIFY *notification;
  int side;
  int inout;
  {
    int vci1,vci2;
    int result;
    int conid1;
    int conid2;
    struct timeval tv1,tv2;
```

```
  struct timezone tz;
  long clock;

  FEC_FLAGS err;
  NEG_RESPONSE response;
  GLOBAL_STATE SystemState;
  RATE_MONOTONIC_SCHEDULER rms;
  INFO_STATE WhatInfo;

  WhatInfo.i_set[0] = SystemStateInfo;
  WhatInfo.i_set[1] = NOT_SPECIFIED;

  RetrieveGlobalState(&SystemState,&rms,WhatInfo);

  printf("negotiateNetQoS: enter \n");
  switch(side)
    {
    case BUYER:
/******************** send application QoS to the remote site ************/
      if (SystemState.net.Nneg_in.status == FREE)
        {
          vci1 = NET_SIGNAL1_VCI;
          connect_s(vci1,&conid1,DATAGRAM_MODE,sizeof(NET_QOS_TABLE));
          SystemState.net.Nneg_in.status = TAKEN;
          SystemState.net.Nneg_in.id = conid1;
        }
      err.err_flag = FALSE;
      if ((result = send_pkt(SystemState.net.Nneg_in.id,param,sizeof(NET_QOS_TABLE),err)
)== WRONG_SIZE)
        {
          perror("negotiateNetQoS datagram too small for app_qos transmission");
          exit(1);
        }
/******************** wait for response from the remote side ***************/
      if (SystemState.net.Nneg_out.status == FREE)
        {
          vci2 = NET_SIGNAL2_VCI;
          connect_r(vci2,&conid2,DATAGRAM_MODE);
          SystemState.net.Nneg_out.status = TAKEN;
          SystemState.net.Nneg_out.id = conid2;
        }

      recv_pkt(SystemState.net.Nneg_out.id,&response,sizeof(NEG_RESPONSE),err);
      switch(response.result)
        {
        case ACCEPT:
          notification->note = NEG_SUCCESS;
          break;
        case MODIFY:
          notification->note = NEG_MODIFY;
          notification->reason = response.reason;
          bcopy((char *)(&response.stream_spec),(char *)(param),sizeof(NET_QOS_TABLE));
          break;
        case REJECT:
          notification->note = NEG_REJECT;
          notification->reason = response.reason;
          bzero((char *)(param),sizeof(NET_QOS_TABLE));
          setNetQoS(param,inout);
          break;
        }
      break;
    case SELLER:

      printf("negotiateNetQoS: System state for net neg = %d \n",
                SystemState.net.Nneg_out.status);

      if (SystemState.net.Nneg_out.status == FREE)
        {

          vci1 = NET_SIGNAL1_VCI;
          connect_r(vci1,&conid1,DATAGRAM_MODE);
          SystemState.net.Nneg_out.status = TAKEN;
          SystemState.net.Nneg_out.id = conid1;
        }
      err.err_flag = FALSE;
      printf("negotiateNetQoS: wait for recv NetQoS \n");
      recv_pkt(SystemState.net.Nneg_out.id,param,sizeof(NET_QOS_TABLE),err);

      printf("negotiateNetQoS: Parameter(size)=%d\n",param->connection[0].load.size);
      printf("negotiateNetQoS: Parameter(rate)=%f\n",
             param->connection[0].load.rate);

/************* STORE RECEIVED NETWORK QoS  *****************************/
      setNetQoS(param,inout);
/************* admission part of the negotiation process by seller *********/
      gettimeofday(&tv1,&tz);

      AdmitNetQoS(param,notification,inout,side);
      gettimeofday(&tv2,&tz);
      getproctime(tv1,tv2,&clock);
      printf("AdmitNetQoS = %d microsecond \n", clock);

      switch(notification->note)
        {
        case NEG_SUCCESS:
          response.result=ACCEPT;
          break;
        case NEG_MODIFY:
          response.result = MODIFY;
          response.reason = notification->reason;
          bcopy((char *)param,(char *)(&response.stream_spec),sizeof(NET_QOS_TABLE));
          break;
        case NEG_REJECT:
          response.result = REJECT;
          response.reason = notification->reason;
          bzero((char *)(param),sizeof(NET_QOS_TABLE));
          setNetQoS(param,inout);
          break;
        }
      if (SystemState.net.Nneg_in.status == FREE)
        {
          vci2 = NET_SIGNAL2_VCI;
          connect_s(vci2,&conid2,DATAGRAM_MODE,sizeof(NEG_RESPONSE));
          SystemState.net.Nneg_in.status = TAKEN;
          SystemState.net.Nneg_in.id = conid2;
        }
      err.err_flag = FALSE;
      send_pkt(SystemState.net.Nneg_in.id,&response,sizeof(NEG_RESPONSE),err);
      break;
    }
  StoreGlobalState(SystemState,rms,WhatInfo);
}
```

```
/*********************************************/
/* Filename : tuneQoS.c                      */
/* Purpose  : Display Video  from Ultimedia device */
/* Author   : Klara Nahrstedt                */
/* Update   : 05/4/95                        */
/*********************************************/


/*****************************************************************************
 * General includes
 *****************************************************************************/

#include <stdio.h>
#include <sys/stat.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <fcntl.h>
#include <math.h>
/*****************************************************************************
 * UMS includes
 *****************************************************************************/

#include <UMSObject.h>
#include <UMSClass.h>
#include <UMSVideoIO.h>
#include <UMSStrings.h>
#include "/home/klara/tele.d/include.d/comm.h"

/*****************************************************************************
 * X variables
 *****************************************************************************/

Display         *dpy;
Window          wid;
int                     screen;
GC                      gc;
int                     depth;
int                     pixel_pad;
Visual          *my_visual;
Status          result;
XVisualInfo     info;
XSetWindowAttributes wa;
long            colormap_size;


/*****************************************************************************
 * UMS variables
 *****************************************************************************/

        UMSVideoIO  video_obj;
        Environment *ev;
        long            colormap_size;

int stopTuneVideo()
{

  int fd;
  int command;

  command = STOP; /* STOP */

  fd = open("/home/klara/tele.d/QoS_management.d/STOPfile",O_RDWR);

  write(fd,(char *) (&command),sizeof(command));

  close(fd);

}


int  startTuneVideo(x,y,side,inout,rate)
int x,y;
int side,inout;
int rate;
{
  XEvent event;
  XImage                        *image;
  unsigned char *imagedata;
  long                  width;
  long                  height;
  int                   number,i,j,end;
  int                           num_frames;
  char                  ch;
  long                  error_code;
  int rc;
  int image_size;
  int inter_delay;
  int fd;
  int fd_com;
  int command;

  ev = somGetGlobalEnvironment();

/********** defaults ************************************************/

  num_frames = 30;
  colormap_size = 128;
  width = 320;
  height = 240;
  depth = 8;

  /* Create the video IO object */

  video_obj = UMSVideoIONew();

  image_size = width*height;

  if ((imagedata=(unsigned char *)malloc(width*height)) == NULL)

    {
      exit(-1);
    }


  /* Create the display window */

  rc = create_window(width, height);
  if (rc == 1)
    {
      error_code = 1;
      goto Error;
    }

  fd = open("/home/klara/tele.d/QoS_management.d/myfile",O_RDWR);
  fd_com = open("/home/klara/tele.d/QoS_management.d/STOPfile",O_RDWR);
  command = START;

  write(fd_com,(char *)(&command), sizeof(command));
```

```
  /* The main loop */

  lseek(fd,0L,SEEK_SET);

  if (rate!=0)
    {
      inter_delay = 60000000/rate; /* in microseconds */
    }
  for (end=0; end < num_frames; end++)
    {

      read(fd,imagedata,image_size);
      lseek(fd,0L,SEEK_CUR);

      printf("number of recv frames = %d \n",end);
      image = XCreateImage(dpy, my_visual, depth, ZPixmap, 0, imagedata,
                           width, height, pixel_pad, 0);
      XPutImage(dpy, wid, gc, image, 0, 0, 0, 0, width, height);

      XSync(dpy,False);
      XFlush(dpy);
/*
      XNextEvent(dpy,&event);
      switch(event.type)
        {
        case ButtonPress:
          if (event.xbutton.window == wid)
            {
              printf("I hit stop \n");
              end = num_frames;
              inter_delay = 0;
            }
          break;
        case ButtonRelease:
          break;
        }
*/
      usleep(inter_delay);
    }

  error_code = 0;


  /* Close the video IO object */


  XFreeGC(dpy,gc);
  XDestroyWindow(dpy,wid);
  free(imagedata);
  rc = UMSVideoIO_close(video_obj,ev);
  if (rc !=UMSVideoIO_Success)
    {
      printf("Cannot close video device. rc = % \n", rc);
      error_code = 1;
    }

  /* Free buffers, and destroy the video object */

 Error:

  return(error_code);
}

/*****************************************************************************
 * create_window
 *
 * Creates the X display window for the captured frames.
 *****************************************************************************/

long create_window(width, height)
long width;
long height;
{
        int rc;

        dpy = XOpenDisplay(NULL);
        if (dpy == NULL)
                {
                fprintf(stderr, "Cannot open display\n");
                return(1);
                }
        XSynchronize(dpy);
        screen = DefaultScreen(dpy);

        if (depth == 8)
                result = XMatchVisualInfo(dpy, screen, 8, PseudoColor, &info);
        else if (depth == 24)
                result = XMatchVisualInfo(dpy, screen, 24, TrueColor, &info);
        if (result)
                my_visual = info.visual;
        else
                {
                fprintf(stderr, "%d-bit visual type not supported\n", depth);
                return(1);
                }

        if (depth == 8)
                pixel_pad = 8;
        else
                pixel_pad = 32;

        rc = set_colormap();
        if  (rc == 1)
                return(1);

        wid = XCreateWindow(dpy, DefaultRootWindow(dpy), 0, 0, width, height,
                                                0, depth, InputOutput, my_visual,
                                                CWColormap | CWBorderPixel, &wa);
        XSync(dpy, False);

        gc = XCreateGC(dpy, wid, 0, NULL);

        XMapWindow(dpy, wid);
        XSelectInput(dpy, wid, ExposureMask | ButtonPressMask);
        return(0);
}

/*****************************************************************************
 * set_colormap
 *
 * Creates the colormap for the X display window
 *****************************************************************************/

long set_colormap()
{
        Colormap        stdmap, cmap;
        XColor          colval[256];
        int                     i;
```

```
        int                     rc;

        if (depth == 8)                                 /* 8-bit display */
                {
                /* Query the colors in the default window's colormap */

                stdmap = XDefaultColormap(dpy, screen);
                for (i = 0; i < 256; i++)
                        colval[i].pixel = i;
                XQueryColors(dpy, stdmap, colval, 256);

           /*
                * Compute the colormap entries to display the video.
                *
                * The function starts with the default colormap, and loads the
                * new values at the top of the colormap.  This helps to avoid
                * colormap flashing when the cursor moves between different
                * X windows.
                */

                rc = compute_colormap(colval);
                if (rc == 1)
                        return(1);

                /* Load the X colormap */

                cmap=XCreateColormap(dpy, DefaultRootWindow(dpy), my_visual, AllocAll);
                XStoreColors(dpy, cmap, colval, 256);
                XInstallColormap(dpy, cmap);
                }
        else                                                    /* 24-bit display */
                {
                cmap = XCreateColormap(dpy, DefaultRootWindow(dpy), my_visual,
                                                        AllocNone);
                }
        wa.colormap = cmap;
        wa.border_pixel = 0;

        return(0);
}
/*****************************************************************************
 * display_image
 *
 * Displays a captured image in the window.  As we have multiple buffers,
 * we need to change the XImage structure to point to the current buffer.
 *****************************************************************************/

/*****************************************************************************
 * compute_colormap
 *
 * Computes the colormap for the X display window
 *****************************************************************************/

long compute_colormap(colval)
XColor colval[];

{        int i, j, n;
        char *colormap;
        unsigned char *colormap_red;
        unsigned char *colormap_green;
        unsigned char *colormap_blue;
        long colormap_index;
        long rc;
        _IDL_SEQUENCE_octet cmap;

/*
 * Compute the colormap index to put the video's colormap at the
 * top of the window's colormap.
 */

colormap_index = 256 - colormap_size;

/* Set the colormap index and size. */

rc = UMSVideoIO_set_colormap_index(video_obj, ev, &colormap_index);
if ((rc != UMSVideoIO_Success) && (rc != UMSVideoIO_ValueChanged))
        {
        fprintf(stderr, "set_colormap_index failed;. rc = %d\n", rc);
        return(1);
        }

rc = UMSVideoIO_set_colormap_size(video_obj, ev, &colormap_size);
if ((rc != UMSVideoIO_Success) &&(rc != UMSVideoIO_ValueChanged))
        {
        fprintf(stderr, "set_colormap_size failed; rc = %d\n", rc);
        return(1);
        }

/*
 * If the object does not support the exact index size used
 * in the set methods, it will set its colormap parameters to
 * "close" values.  We use the get methods to query these
 * values.
 */

rc = UMSVideoIO_get_colormap_index(video_obj, ev, &colormap_index);
if (rc != UMSVideoIO_Success)
        {
        fprintf(stderr, "Cannot get_colormap_index. rc = %d\n", rc);
        return(1);
        }

printf("get_colormap_index: colormap_index = %d\n", colormap_index);

rc = UMSVideoIO_get_colormap_size(video_obj, ev, &colormap_size);
if (rc != UMSVideoIO_Success)
        {
        fprintf(stderr, "get_colormap_size failed; rc = %d\n", rc);
        return(1);
        }

printf("get_colormap_size: colormap_size = %d\n", colormap_size);

/* Set up the SOM sequence structure to receive the colormap */

cmap._length  = 4 * colormap_size;
cmap._maximum = cmap._length;
cmap._buffer  = malloc(4 * colormap_size);
if (cmap._buffer == NULL)
        {
        fprintf(stderr, "Cannot malloc space for colormap\n");
        return(1);
        }

/* Get the colormap */

rc = UMSVideoIO_get_colormap(video_obj, ev, &cmap);
if (rc != UMSVideoIO_Success)
```

```
                {
                fprintf(stderr, "get_colormap failed; rc = %d\n", rc);
                return(1);
                }

        /* Reformat the colormap for X */

        n = colormap_index;
        colormap = cmap._buffer;

        /* The object's colormap values are in 0RGB format.     Hence... */

        colormap_red   = colormap + 1;
        colormap_green = colormap + 2;
        colormap_blue  = colormap + 3;

   /*
        * Scale the component values to the 16-bit X format, and write
        * them out.
        */

        for (i = 0; i < colormap_size; i++)
                {
                j = 4 * i;
                colval[n].pixel = n;
                colval[n].flags = DoRed | DoGreen | DoBlue;
                colval[n].red   = ((int) colormap_red[j]   * 65535) / 255;
                colval[n].green = ((int) colormap_green[j] * 65535) / 255;
                colval[n].blue  = ((int) colormap_blue[j]  * 65535) / 255;
                n++;
                }

        return(0);


}
```

```
/*****************************************************************/
/* Filename: comm.h                                              */
/* Purpose : Communication connected  structures                 */
/* Author  : Klara Nahrstedt                                     */
/* Update: 12/15/94                                              */
/*****************************************************************/
#include "/home/klara/tele.d/include.d/general.h"
#include "/home/klara/tele.d/include.d/rtnp.h"
#include "/home/klara/tele.d/include.d/qos.h"

/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
/* Application QoS Negotiation/renegotiation, set up AQoS        */
/*****************************************************************/


typedef struct neg_resp
{
  int  result;
  int reason;
  APP_QOS stream_spec; /* if response = MODIFY then modified app_qos are sent*/
}NEG_RESPONSE;


/******************** communication states *******************/

/* state variables  between master and slave at the application level*/

#define QUIT     0
#define START    1
#define STOP     2
#define CALL_SET_UP 3
#define RENEGOTIATE 4
#define NEGOTIATE   5

/* state variables of negotiation situation between master and slave */
/* appl. receiver accepts the sensory environment */

#define ACCEPT         20

/* appl. receiver  has to modify the sensory env. */

#define MODIFY         21

/* appl. receiver can't communicate         */

#define REJECT  22

/****************** application data units ***************/

/****************** data packet format ******************/

typedef struct video_data
{
  int type;
  int fragment_nr;
  char *videodata;
} VIDEO_TYPE;


/*****************************************************************/
/*****************************************************************/
/*****************************************************************/
```

```
/* Network Communication Structures and Constants                */
/*****************************************************************/

/*****************************************************************/
/* Connect structure for atm connection set up used in      */
/* mastermain.c  and slavemain.c for unidirectional connetions   */
/*****************************************************************/

typedef struct single_connection
{
  int flag;
  int medium;
  int priority;
}CONNECT;



/***************** communication states *****************/

/* negotiation between master and slave about net QoS   */

#define NET_CON_ACCEPT 23
#define NET_CON_REJECT 24
#define NET_CON_MODIFY 25

/***************** network packet ***********************/
#define COPY 1
#define ORIGIN 0
typedef struct network_cell
{
  int seq;
  int copy;
  char *data;
}NET_CELL;

typedef struct network_pkt
{
  int seq;
  char *data;
}NET_DATAGRAM;

/************** FEC *********************************/

typedef struct fec_info
{
  int err_flag;
  int connid;   /* connection id for additional connection thorugh which the
                   additional information is sent */
}FEC_FLAGS;
```

```
/*****************************************************************/
/* Filename: qos.h                                               */
/* Purpose: Description of qos parameters                        */
/* Author: Klara Nahrstedt                                       */
/* Update: 5/19/95                                               */
/*****************************************************************/

/*****************************************************************/
/* Description of Application Media specified Quality of Services */
/*****************************************************************/

#define INPUT 0
#define OUTPUT 1

/*****************************************************************/
/* side   specification in QoS broker - Buyer or Seller     */
/*****************************************************************/

#define BUYER  20  /* Sender */
#define SELLER 21  /* Receiver */

/*****************************************************************/
/* Direction Specification in QoS translator                     */
/* Two  possibilities: From AppToNet, or NetToApp                */
/*****************************************************************/

#define APP_TO_NET    30 /* Translation from Application QoS to Network QoS */
#define NET_TO_APP    31 /* Translation from Network QoS to Application QoS */

/*****************************************************************/
/* Parameter Specification Errors                                */
/*****************************************************************/

#define NOT_SPECIFIED    0
#define OTHER           10
#define NOT_SUPPORTED   10
#define BAD_MAXBOUND     9
#define BAD_MINBOUND     8
#define BAD_VALUE        7
#define NONE             0

#define COMPONENT_NUMBER 5  /* number of  components in a medium stream */
#define MEDIA_NUMBER     5 /* number of media streams in mm stream */
#define MEDIA_PAIR       2
#define CONNECTION_NUMBER 32 /* actually 256, but we don't need so many */
/*****************************************************************/
/* Media Relations                                               */
/*****************************************************************/

typedef struct integration
{
  int link_id;
  int media[MEDIA_NUMBER];
}INTEGRATE;

typedef struct media_convert
{
  int from_media;
  int to_media;
}CONVERT;

typedef struct comp_fr
{
  int name;
  BOOLEAN frag;
  double number;
}COMP_FRAG;

typedef struct frag_spec
{
  int medium;
  COMP_FRAG component[COMPONENT_NUMBER];
  BOOLEAN frag;
  double number;  /* number of fragments*/
}FRAGMENT;

typedef struct synchronization
{

  int sync_skew;
  int between[MEDIA_PAIR];
}SYNC;

typedef struct media_rel
{
  BOOLEAN done[4]; /* 0-comp=R_SYNCH,1-comp=R_INTEG,2-comp=R_CONV,..*/
  SYNC synch_spec;
  INTEGRATE over_connection[MEDIA_NUMBER]; /* integration structure */
  CONVERT  conversion[MEDIA_NUMBER];       /* conversion structure  */
  FRAGMENT fragmentation[MEDIA_NUMBER];
  int communication;                       /*Communication relation */
}MEDIA_RELATIONS;

/*****************************************/
/* Communication relation                */
/*****************************************/

#define UNICAST     1
#define MULTICAST   2

/*****************************************/
/* IMPORTANCE for Media                  */
/*****************************************/

#define HIGH_IMPORTANCE  3   /* hard-real-time */
#define MEDIUM_IMPORTANCE 4  /* soft-real-time */
#define LOW_IMPORTANCE   5   /* non-real-time  */


/*****************************************/
/* Transmission Characteristics          */
/*****************************************/

typedef struct appl_net_qos
{                              /* delay between the sender and receiver */
  long     end_to_end_delay;     /* in milisecond if delay sensitive appl */
  long     loss_rate;           /* number of lost packets per second     */
  BOOLEAN  security;
  long     con_estab_delay;    /* important service setup time if failure*/
                               /* in the communication link occur too often */
  int importance;
  int cost;
  int count_con;   /* number of connection ids occupied */
  int conid[CONNECTION_NUMBER];
}TRANS_CHAR; /* APP_NET_QOS */

/*****************************************/
/* Compression Specification             */
```

```
/*********************************************/

typedef struct compress
{
  int name; /* compression mechanism JPEG, MPEG, etc. */
  int ratio; /* compression ratio 5:1 */
}COMPRESSION;

/*********************************************/
/* Medium Characteristics                    */
/*********************************************/

typedef struct app_qos
{
  int quality; /* Telephone, CD,... */
  int sample_size;
  int sample_rate;
  COMPRESSION comp_spec;
}MEDIA_CHAR;

/*********************************************/
/* Intraframe Specification                  */
/*********************************************/

typedef struct intra_spec
{
  int name;  /* name of the component (B-frame,I-frame,P-frame, a,n,t,.. */
  int size;  /* size of the component in bytes */
  int rate;  /* rate of the component */
  int importance;
  long loss; /* loss rate in packets per time */
}INTRAFRAME;

/*********************************************/
/* Interframe Specification                  */
/*********************************************/


typedef struct inter_spec
{
  MEDIA_CHAR app_spec;
  TRANS_CHAR net_spec;
}INTERFRAME;

/*********************************************/
/* Medium Quality Specification              */
/*********************************************/

typedef struct medium_spec
{
  BOOLEAN scheduled; /* medium is scheduled or not */
  int type;  /* medium type - audio,video, sensory data */
  BOOLEAN intra;  /* if intrafrem spec exists */
  int direction; /* INPUT or OUTPUT */
  INTRAFRAME component_spec[COMPONENT_NUMBER];
  INTERFRAME medium;
}MEDIA_QUALITY;


/*********************************************/
/* Video Quality Characteristics             */
/*********************************************/

#define STILL_IMAGE      0
#define MOTION_VIDEO     1


#define JPEG_COMP        1
#define MPEG             2


/*********************************************/
/* Audio Quality Characteristics             */
/*********************************************/

#define TELEPHONE_QUALITY  0
#define CD_LIKE_QUALITY    1
#define CD_QUALITY         2

/*********************************************/
/* Robotics Quality                          */
/*********************************************/
#define LOW     0   /* rate between 50 and 100  */
#define MEDIUM  1   /* rate between 100 and 300 */
#define HIGH    2   /* rate between 300 and 500 */

/*********************************/
/* Type of data                  */
/*********************************/

#define  ROBOT  1
#define  NETWORK 2
#define  TEXT   3
#define  VIDEO  4
#define  AUDIO  5
#define  RENEG_INFO 6

/*************************************************/
/* Application QoS Specification                 */
/* Unidirectional multimedia stream specification */
/*************************************************/

typedef struct env_descr
{
  MEDIA_QUALITY    stream[MEDIA_NUMBER];
  MEDIA_RELATIONS  relations;
}APP_QOS;

/* in dialogCMSMediaQuality.c I need parameter spec for setParamValue */
/* Media Quality Parameter Flags for Interframe Spec*/


#define S_INTRA   47
#define S_COMPRESS 48
#define S_TYPE    49
#define QUALITY   50
#define S_SIZE    51 /* sample size */
#define S_RATE    52
#define S_LOSS    53
#define S_DELAY   54  /* end-to-end delay */
#define S_PRIO    55  /* sample importance */

/* Media Quality Parameter Flags for Intraframe Spec */
#define C_NAME    56
#define C_SIZE    57
#define C_RATE    58
#define C_PRIO    59
#define C_LOSS    60
```

```
/* Medium Component Specification */
#define N_COMP    0
#define O_COMP    1
#define A_COMP    2
#define P_COMP    3


/* in dialogCMSMediaRelation.c I need parameter spec for setRelationValue */
/* Media Relation Parameter Flags */

#define R_SYNCH  0
#define R_INTEG  1
#define R_CONV   2
#define R_COMM   3

/*************************************************************************/
/* Additional Information Structure sent during the QoS negotiation    */
/*************************************************************************/

typedef struct OneAddInfo
{
  int side;      /* master slave */
  int direction; /* request, response */
  int param;      /* get image, set position */
  int done;     /* yes/no */
  int value;    /* positions */
}ONE_INFO;

/******************** directions ***********************************/

#define REQUEST       6
#define RESPONSE      7

/******************** parameters ***********************************/

#define GET_IMAGE     2
#define SET_POSITION  3

/****************** positions ***********************************/

#define PARK          4
#define READY         5

typedef struct ArrayOfOneAddInfo
{
  APP_QOS other_qos;
  ONE_INFO  info[10];
}ADD_INFO;

/*********************************************************************/
/* Device Support Description for Input/Output  in MM Application  */
/*********************************************************************/

/*******************************/
/* Device structure             */
/*******************************/

typedef struct dev_descr
{
 int       type_of_data;    /*  Data type the device supports  */
 int       device_descr;    /*  Device Description             */
  BOOLEAN   support;         /*  Device is/is not supported at the end-point */
}DEVICE;
```

```
/*******************************/
/* Device Description           */
/*******************************/

#define  SCREEN             20   /* in/out device            */
#define  CAMERA             21   /* Video Input              */
#define  MICROPHONE         22   /* audio device input       */
#define  SPEAKER            23   /* audio device output      */
#define  ROBOT_HAND         24   /* in/out device            */
#define  ROBOT_SIMULATOR    25   /* output device            */

/*****************************************************/
/* Description of multimedia devices supported at the*/
/* end-point for input and output separately         */
/*****************************************************/

typedef struct appl_devices
{
   DEVICE       dev_support[MEDIA_NUMBER];
   MEDIA_CHAR   spec[MEDIA_NUMBER];
 }MM_DEVICES;

/*****************************************************************************/
/* ------------------------------NETWORK ------------------------------------*/
/*****************************************************************************/
/* Network Quality Specification                                             */
/*****************************************************************************/


typedef struct reliable
{
   long loss_rate;
   BOOLEAN loss_cons_pkt;   /* loss of two consecutive pkt possible */
 } RELIABILITY;

/*********************************************************************/
/* Network Connection Quality Specification                          */
/*********************************************************************/

typedef struct l_spec
{
  int id;     /* connection identifier */
  int size;  /* packet size */
  RELIABILITY loss;  /* packet error rate */
  double rate;  /* packet rate */
  double throughput; /* bytes per second */
  double end_to_end_delay;
  double intermediate_delay;
  int priority;
}CONN_LOAD;

/************************************************/
/*  Error Algorithms                            */
/************************************************/

#define FEC  11
#define RETRANSMISSION 12
#define PRIORITY_CODING 13

/************************************************/
/* Rate-Control Service Disciplines             */
/************************************************/
```

```
#define FIFO 20
#define RATE_BASED 21

typedef struct p_spec
{
  int error_alg; /* FEC, Retranmsission , etc. */
  int rate_alg;  /* FIFO, RATE Monotonic */
  BOOLEAN ordering;
  int communication; /* Unicast, multicast, broadcast */
  int cost;
}CONN_PERFORMACE;

typedef struct connection_qos
{
  CONN_LOAD load;
  CONN_PERFORMACE perform;
}NET_QOS;

typedef struct net_table_spec
{
  int filler_byte;  /* doesn't mean anything, need for open connection
                       size of NET_QOS_TABLE  cannot be (n*44)+4; */
  int medium[CONNECTION_NUMBER];
  BOOLEAN status[CONNECTION_NUMBER]; /* the connection is free or taken */
  NET_QOS connection[CONNECTION_NUMBER];
  int vci[CONNECTION_NUMBER];
}NET_QOS_TABLE;


typedef struct reneg_spec
{
  int menu_state;
  int changed_rate;
}RENEGOT_INFO;
```

```
/*********************************************************************/
/* Filename: defs.h                                                  */
/* Purpose : Definition of system h files                            */
/* Author  : Klara nahrstedt                                         */
/* Update  : 02/28/95                                                */
/*********************************************************************/



#include     <stdio.h>
#include     <math.h>
#include     <ctype.h>
#include     <sys/types.h>
#include     <signal.h>
#include     <sys/socket.h>
#include     <sys/socketvar.h>
#include     <netinet/in.h>
#include     <sys/errno.h>
#include     <limits.h>
#include     <fcntl.h>
#include     <unistd.h>
#include     <stdlib.h>
#include     <sys/ioctl.h>
#include     <string.h>
#include     <arpa/inet.h>
#include     <signal.h>
#include     <netdb.h>
#include     <sys/time.h>
#include     <sys/mdio.h>
#include     <sys/file.h>
#include     <sys/uio.h>
#include     <sys/stat.h>

/*************** bit 3 need following h-files ************************/

#include     <sys/mman.h>
#include     <sys/btio.h>
#include     "/pkg/bit3/921/v1.6/sys/btio.h"
#include     "/pkg/bit3/921/v1.6/sys/btlio.h"

/*************** atm needs following   h-files ***********************/

#include    <sys/mode.h>
#include    <sys/errno.h>
#include    <sys/devinfo.h>
#include    <sys/comio.h>
#include    <sys/xmem.h>

/*
#include     <sys/timer.h>
#include     <sys/trchkid.h>
#include     <sys/trcmacros.h>
*/
#include     <sys/resource.h>
#include     <sys/param.h>
#include     <varargs.h>
#include     <sys/wait.h>
#include     "/home/klara/tele.d/include.d/general.h"

extern int    errno;
extern int    sys_nerr;
extern char  *sys_errlist[];

#define AURORA   "aurora.cis.upenn.edu"
#define GODZILLA "godzilla.cis.upenn.edu"
#define FORBIN   "forbin.cis.upenn.edu"
#define BEEBO    "beebo.cis.upenn.edu"
#define BINGLE   "bingle.cis.upenn.edu"
#define UPWARDS  "upwards.cis.upenn.edu"

/* #define SERV_HOST_ADDR "158.130.6.16"  FORBIN */
#define SERV_HOST_ADDR3 "158.130.6.3" /* AURORA */
#define SERV_HOST_ADDR "158.130.10.47" /*UPWARDS*/
/* #define SERV_HOST_ADDR  "158.130.6.17" Godzilla */

#define SERV_HOST_ADDR2 "158.130.10.51" /*BEEBO*/
```

```
/*************************************************************/
/* Filename: general.h                                       */
/* Purpose: General Definitions of Constants                 */
/* Author:  Klara Nahrstedt                                  */
/* Update: 06/16/94                                          */
/*************************************************************/

#define TRUE     1
#define FALSE    0
#define BOOLEAN  int

#ifndef NULL
#define NULL  ((void *) 0)
#endif

#define MILLION 1000000
```

```
/*************************************************************************/
/* Filename: retta.h (REal-Time-Teleoperation-Application                 */
/* Purpose : Structures and Constants tied to the teleoperation app       */
/* Author  : Klara Nahrstedt                                              */
/* Update  : 12/06/94                                                     */
/*************************************************************************/

#define    TEXT1_FONT       1
#define    TEXT2_FONT       2
#define    MAX_TEXT_LENGTH  120

#define   RES_NAME   "teleapp"
#define   RES_CLASS  "realtime"
#define   DEFAULT_FONT1   "variable"
#define   DEFAULT_FONT2   "9x15"
#define   DEFAULT_TITLE   "TeleManufacturing Application"


#define NORMAL_STATE   0
#define ICONIC_STATE   1

#define NORMAL_WINDOW   0
#define POP_UP_WINDOW   1

#define IN_PALETTE      1
#define NOT_IN_PALETTE  2

#define MAX_CHAR     10

/* Button Sizes */

#define BUTTON_LEVEL1_HEIGHT 25
#define BUTTON_LEVEL1_WIDTH  100

#define BUTTON_LEVEL2_HEIGHT 20
#define BUTTON_LEVEL2_WIDTH  60

#define BUTTON_LEVEL3_HEIGHT 15
#define BUTTON_LEVEL3_WIDTH  15


#define DISTANCE 20
#define LINE_DISTANCE 25

#define MAIN_WINDOW_WIDTH 900
#define MAIN_WINDOW_HEIGHT 900

/* specification of master or slave side */

#define MASTER  1
#define SLAVE   0

#define CONFIGURE 4
#define CHANGE_SHOW 5

#define TEXT_WINDOW_HEIGHT   15
#define TEXT_WINDOW_WIDTH    50

#define ERROR_WINDOW_HEIGHT  20
#define ERROR_WINDOW_WIDTH   300
typedef struct MediaQualityFlags
{
  int descr;
  int audio;
  int video;
  int robot;
  int changeRobotRate;
  int initRobotIntra;
  int selectTune;
  int freeListTuneDone;
  int freeErrorWindow;
  int selectCompression;
  int freeListCompressionDone;
  int send;
  int change_show;
  } MQ_FLAG;

typedef struct MediaRelationFlags
{
  int selectSync;
  int errorWindow;
}MR_FLAG;

/****************** flags for dialogCallMS.c *************************/

typedef struct AddInfoFlags
{
  int AddInfoWindows; /* GetImage and SetPositions*/
  int SetPositionWindows; /* ReadyPostion and ParkPosition */
}ADD_FLAG;

/***************** notification responses at the user interface ******/

#define NEG_SUCCESS 6
#define NEG_REJECT  7
#define NEG_MODIFY  8

/**************** reasons for NEG_REJECT, MODIFY *******************/

#define VIDEO_NOT_SUPPORTED 9
#define END_TO_END_TEST_FAILURE 10
#define SCHEDULE_NOT_FEASIBLE 11
#define THROUGHPUT_TEST_FAILURE 12

/**************** reasons for NEG_SUCCESS ****************/
#define GET_IMAGE_SUCCESS   20

typedef struct note_spec
{
  int note;
  int reason;
}NOTIFY;
```

```
/*********************************************************/
/*  Filename :     robnet.h                              */
/*  Description :  shared h file between JIFFE and RS/6000 */
/*  Author :       Klara Nahrstedt,                      */
/*                 DSL-Lab                               */
/*  Update:        05/01/93                              */
/*********************************************************/

/********************* General Constants ******************/

#define NUMPOINTS              12 /* number of robotics data */

/********************* Shared States - kind ***************/

/*#define FIRST      0 *//* first action for initialization of ports */
/*#define READY      1 *//* Ready state/Init done from jiffe side */
/*#define BEGIN      2 *//* Ready state/Init done from network side */
/*#define SEND       3 *//* Busy state/Jiffe writes data into VME/MCA bus */
/*#define RECEIVE    4 *//* Busy state/Network writes data into VME/MCA bus */
/*#define DONE       5 *//* Done state/Task done  from jiffe side */
/*#define END        6 *//* Done state/Task done from network side */
/*#define RBT_ERROR  7 *//* Error state/Problems from jiffe side */
/*#define NET_ERROR  8 *//* Error state/Problems from network side */

#define R_FIRST       0
#define R_READY       1
#define R_BEGIN       2
#define R_SEND        3
#define R_RECEIVE     4
#define R_DONE        5
#define R_END         6
#define R_RBT_ERROR   7
#define R_NET_ERROR   8

#define R_LAST        -1
/************ Specification of buffers on VME/MCA dual port *************/

#define BUF_IN        100    /* take portion for sending robot data */
#define BUF_OUT       101    /* take portion for receiving robot data */

typedef struct robot_io_data
{
  int     kind;
  int     seq;
  float   data[NUMPOINTS];
}ROBOT_IO;
```

```
/*************************************************************************/
/* Filename : rtnp.h                                                     */
/* Purpose : All constants  on the rtnp protocol stack    */
/* Author : Klara Nahrstedt                                              */
/* Update : 06/19/95                                                     */
/*************************************************************************/

#ifndef DEFAULT_MID
#define DEFAULT_MID 1
#define DEFAULT_OTHER 0
#endif

#define PAGESIZE 4096
#define DEFAULT_DEVICE "/dev/atm"

#ifndef MOBY_SIZE
#define MOBY_SIZE 65536
#endif

#define CELL_SIZE 48
#define DATAGRAM_SIZE MOBY_SIZE

#define CELL_MODE      2
#define DATAGRAM_MODE  3


/****************** connection status **********************************/

#define FREE  0
#define TAKEN 1

/****************** error flags for rtnp protocol **********************/

#define OK                  0
#define WRONG_SIZE          1
#define ERROR_DATAGRAM      2 /* notification recv-datagram is errorful */
#define ERROR_CELL          3 /* notification recv-cell is errorful */



/***************** fixed vci's for signalling of application negotiation,
network negotiation and forward error correction
*/

#define APP_SIGNAL1_VCI 0x4060  /*caller(operator) -> callee(slave) */
#define APP_SIGNAL2_VCI 0x4061  /*callee(slave) -> caller(operator) */
#define NET_SIGNAL1_VCI 0x4050  /*caller(operator) -> callee(slave) */
#define NET_SIGNAL2_VCI 0x4051  /*callee(slave) -> caller(operator) */
#define FEC_CELL1_VCI   0x0010  /*caller(operator) -> callee(slave) */
#define FEC_CELL2_VCI   0x0011  /*callee(slave) -> caller(operator) */
#define FEC_DATAGR1_VCI 0x4040  /*caller(operator) -> callee(slave) */
#define FEC_DATAGR2_VCI 0x4041  /*callee(slave) -> caller(operator) */
#define RENEG_SIGNAL_VCI 0x0012 /* from buyer to seller - signal change */

/************ NETWORK END-TO-END DELAY ******************************/
/************* in microseconds ************************************/
#define NET_CELL_DELAY    5
#define NET_DATAGRAM_DELAY  3000
```

```
/*************************************************************************/
/* Filename: systemQoS.h                                                 */
/* Puspose: Description of system QoS parameters                         */
/* Author : Klara Nahrstedt                                              */
/* Update : 06/09/95                                                     */
/*************************************************************************/


#define NUMBER_OF_TASKS_PER_MEDIUM 10
/* Number of ticks is number of periods for which we need to compute
schedule, it goes from 0 to the gcm of the media                        */
#define NUMBER_OF_TICKS  10

/*************************************************************************/
/* Task per Medium Spec                                                  */
/*************************************************************************/

typedef struct task_spec
{
  BOOLEAN Scheduled;
  int name;
  int ordering;
  long period;
  long duration;
}TASK;

typedef struct tasks_spec
{
  int medium;
  int inout;
  TASK app[NUMBER_OF_TASKS_PER_MEDIUM];
}TASKS;


/*************************************************************************/
/* Scheduler Specification                                               */
/*************************************************************************/

typedef struct sched_element_spec
{
  int medium;
  int inout;
  int task_name;
  int task_duration;
  long time_begin;
  long time_deadline;
}SCHED_ELEMENT;

typedef struct sched_per_minperiod
{
  SCHED_ELEMENT sched_queue[MEDIA_NUMBER*NUMBER_OF_TASKS_PER_MEDIUM];
}SCHEDULER_PERIOD;

typedef struct sched_spec
{
  int number_of_ticks; /*gcm*/
  long min_period;
  SCHEDULER_PERIOD sched[NUMBER_OF_TICKS];
}RATE_MONOTONIC_SCHEDULER;
typedef struct buff_spec
{
  int size;
}BUFFSIZE;
```

```
/*************************************************************************/
/*  System Resources                                                     */
/*************************************************************************/

typedef struct resource_spec
{
  TASKS  ss_tasks;
  BUFFSIZE ss_buffer;
}RESOURCES;

/*************************************************************************/
/* Task Names                                                            */
/*************************************************************************/

#define ReadRobotData 1
#define CopyRobotData 2
#define WriteRobotData 3
#define PointerManagementRobotData 4

#define SendCell 11
#define ReceiveCell 12
#define SendDatagram 13
#define ReceiveDatagram 14

#define Renegotiate 20

#define ReadVideoData 100
#define CopyVideoData 101
#define WriteVideoData 102  /* Display*/
#define PointerManagementVideoData 103



/*************************************************************************/
/* Global State for admission  tests                                     */
/*************************************************************************/

typedef struct me_info
{
  int medium;
  int rate;
}MEDIUM_INFO;

typedef struct st_info
{
  MEDIUM_INFO media[MEDIA_NUMBER];
}STREAM_INFO;

typedef struct sd_info
{
  STREAM_INFO sdirection[2];
}STREAM_DIRECTION_INFO;

typedef struct co_info
{
  int id;
  int status;
}CONNECTION_INFO;

typedef struct ca_info
{
  CONNECTION_INFO Aneg_in;
  CONNECTION_INFO Aneg_out;
  CONNECTION_INFO Nneg_in;
```

```
  CONNECTION_INFO Nneg_out;
}CALL_NEG_INFO;

typedef struct state_spec
{
  STREAM_DIRECTION_INFO app;
  CALL_NEG_INFO net;
}GLOBAL_STATE;

/*********************************************************************/
/* Specification of What Global State Information should be stored */
/* or retrieved                                                      */
/*********************************************************************/

#define NUMBER_OF_STATE_INFO 2

#define SystemStateInfo  1
#define AppScheduleInfo  2
#define ScheduleInfo     3

typedef struct state_info
{
  int i_set[NUMBER_OF_STATE_INFO];
}INFO_STATE;
```

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/cursorfont.h>
#include <X11/keysym.h>
#include <X11/keysymdef.h>
#include <X11/Xatom.h>

#ifndef MAXHOSTNAME
#define MAXHOSTNAME 80
#endif
#ifndef BUFSIZE
#define BUFSIZE 256
#endif


#define DEFAULT_CURSOR XC_left_ptr

#define BUTTON_WIDTH  60
#define BUTTON_HEIGHT 24
#define ButtonText(display, window, gc, text, length) \
       (XDrawImageString(display, window, gc, 5, 15, text, length))
#define WANTS_EXPOSES  1
#define NOWANT_EXPOSES 0
#define FIXED_POSITION    USPosition
#define USERSET_POSITION  PPosition


typedef struct BUTTON {
  Display *display;
  Window w;
  Window parent;
  GC gc;
  unsigned long fore,back;
  char name[80];
  KeySym hotkey;               /* KeySym associated with name of button */
                               /* e.g., Quit (name) and XK_q (hotkey)   */
  int length;                  /* strlen of text                        */
  int (*function) ();          /* function to call when button pressed  */
  int active;                  /* Can user press the button (on/off)?   */
  unsigned long foregroundoff; /* foreground when button is off         */
  int instr;                   /* Does the button refer to an instruction */
                               /* This is used for message passing      */
  struct BUTTON *next;
} Button;

typedef struct BUTTONLIST {
  int count;
  Button *buttons;             /* pointer to list of buttons */
  Button *parent_window;       /* this is where all of the info on the parent
                                  can be kept */
} ButtonList;

/* This structure keeps track of all info on a process' main window
   There will probably be only 1 main window for each process.

   This information can be used when a window is closed to free
   up all the X resources associated with the window and all of its
   children (buttons).
*/
typedef struct MAINXWINDOW {  /* Top-level window for a process */
  Display *display;           /* the window's display */
  Window window;              /* the main window      */
  GC gc;                      /* Its graphics context */
  XFontStruct *font_struct;   /* The font associated with the window */
  int screen;
  ButtonList *buttonlist;     /* Buttons associated with this window */
} MainXWindow;


typedef struct ACCFRAME
{ char username[MAXHOSTNAME];
  char hostname[MAXHOSTNAME];
  int width;
  int height;
  char *image;
} Accframe;

typedef struct PHOTOIDWINDOW {
  Accframe    *photoinfo;
  MainXWindow *windowinfo;
} PhotoIDWindow;
```

```
/*
** argsx.c
** Functions to handle commend-line arguments to an X program
*/

#include <stdio.h>

#include "/home/klara/tele.d/include.d/retta.h"

#define DEFAULT_DISPLAY    NULL
#define DEFAULT_GEOMETRY   NULL

/*
** getArguments sets up a set of test strings with
** either default values or the values entered in by the user on the
** command line
*/


getArguments(argc, argv, displayName, geometry,font1Name,
             font2Name, title)
int argc;
char *argv[];
char displayName[];
char geometry[];
char font1Name[];
char font2Name[];
char title[];

{
  int argCounter;
  int iconicState;

  displayName[0] = '\0';
  geometry[0] = '\0';
  strcpy(font1Name, DEFAULT_FONT1);
  strcpy(font2Name, DEFAULT_FONT2);
  strcpy(title, DEFAULT_TITLE);

  iconicState = NORMAL_STATE;


  for (argCounter = 0; argCounter <argc; argCounter++)
    {
      if (strncmp(argv[argCounter], "-h", 2) == 0)
        {
          printhelpMessage();
          exit(1);
        }

      if (strncmp(argv[argCounter], "-display",8) == 0)
        {
          argCounter++;
          if (argCounter < argc)
            {
              strcpy(displayName,argv[argCounter]);
            }
          else
            {
              fprintf(stderr,
                      "ERROR: the -display option should be %s \n",
                      " f0llowed by a display name.");
            }
        }

      if (strncmp( argv[argCounter], "-geom", 5) == 0)
        {
          argCounter++;
          if (argCounter < argc)
            {
              strcpy(geometry, argv[argCounter]);
            }
          else
            {
              fprintf(stderr,
                      "ERROR: the -geometry option should be %s\n",
                      "followed by a geometry spec,");
              fprintf(stderr,
                      "e.g. 100x100+200+200 for \n%s\n%s\n",
                      "location 100,100",
                      "size 200 by 200.");
            }
        }

      if ((strncmp(argv[argCounter],"-title",6) == 0) ||
          (strncmp(argv[argCounter],"-name",5) == 0))
        {
          argCounter++;
          if (argCounter < argc)
            {
              strcpy(title,argv[argCounter]);
            }
          else
            {
              fprintf(stderr,
                      "ERROR: the -title option should be %s\n",
                      "be fllowed by a window title");
            }
        }

      if (strncmp(argv[argCounter],"-iconic",7) == 0)
        {
          iconicState =ICONIC_STATE;
        }
      if (( strncmp(argv[argCounter], "-font", 5) ==0) ||
          (strncmp(argv[argCounter],"-fn",3) == 0))
        {
          argCounter++;
          if(argCounter < argc)
            {
              strcpy(font2Name,argv[argCounter]);
            }
          else
            {
              fprintf(stderr,
                      "ERROR: the -font option should be %s\n",
                      "be followed by a font name.");
            }
        }
    }

if (strlen(displayName) <1)
  displayName = NULL;

if (strlen(geometry) <1)
  geometry = NULL;

return(iconicState);
```

```
  }/* -- function getArguments */

/*
** printhelpMessage
*/

printhelpMessage()
{
   fprintf(stderr, "The allowable command line option are:\n");
   fprintf(stderr, "\t-display displayname \n");
   fprintf(stderr, "\tUse a different display for output\n");
   fprintf(stderr, "\t-geometry geometrspec \n");
   fprintf(stderr, "\tSpecify window location and size \n");
   fprintf(stderr, "\t-font fontname \n");
   fprintf(stderr, "\tUse the given font name for text\n");
   fprintf(stderr, "\t-title windowtitle\n");
   fprintf(stderr, "\tUse the given name for the window title\n");
   fprintf(stderr, "\t-iconic\n");
   fprintf(stderr, "\tStart with the window in iconic state \n");
 } /* -- function printhelpMessage */
```

```
/*************************************************************************/
/** Filename: buttonx.c                                                 **/
/** Purpose : Routines for processing ButtonPress events - main buttons
     (EXIT,START,STOP) */
/** Author  : Klara Nahrstedt                                           **/
/** Update  : 07/01/95                                                  **/
/*************************************************************************/


#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "/home/klara/tele.d/include.d/defs.h"
#include "/home/klara/tele.d/include.d/retta.h"
#include "/home/klara/tele.d/include.d/comm.h"
#include "/home/klara/tele.d/include.d/systemQoS.h"

extern *display;

extern Window theTeleroboticsWindow;
extern Window theExitWindow;
extern Window theConfigurationWindow;
extern Window theCallSetUpWindow;
extern Window theStartWindow;
extern Window theStopWindow;
extern Window theHelpWindow;

extern GC theTeleroboticsGC;
int pid; /* child process id */

/*
** processButton
*/


processButton(event,MenuStateFD)
XButtonPressedEvent   *event;
int MenuStateFD;
{
  int status =1;
  int choice;
  int x,y;
  GLOBAL_STATE SystemState;
  RATE_MONOTONIC_SCHEDULER Scheduler;
  INFO_STATE WhatInfo;
  int MenuControl;
  FEC_FLAGS err;
  int onalarm();
  long clock;
  struct timeval t1neg,t2neg;
  struct timezone tz;
  XExposeEvent theExposeEvent;

  err.err_flag = FALSE;

  if (event->window == theExitWindow)
    {
      highlightChoice(event->window,"grey",
                      BUTTON_LEVEL1_WIDTH,BUTTON_LEVEL1_HEIGHT);
      XFlush(display);
      WhatInfo.i_set[0]=SystemStateInfo;
      WhatInfo.i_set[1]=NOT_SPECIFIED;
      RetrieveGlobalState(&SystemState,&Scheduler,WhatInfo);
       if (SystemState.net.Aneg_in.status == TAKEN)
        {
          printf("Send to slave QUIT \n");
          MenuControl = QUIT;
          lseek(MenuStateFD,0L,0);
          write(MenuStateFD,(char *)(&MenuControl),sizeof(int));
          send_pkt(SystemState.net.Aneg_in.id,&MenuControl,sizeof(int),err);
        }
      return(0);
    }
  if (event->window == theConfigurationWindow)
    {
      highlightChoice(event->window,"grey",
                      BUTTON_LEVEL1_WIDTH,BUTTON_LEVEL1_HEIGHT);

      x =DISTANCE;
      y=100;

      setColorWithName(theTeleroboticsGC,"blue");
      XDrawString(display,theTeleroboticsWindow, theTeleroboticsGC,
                  DISTANCE, y-5,
                  "QoS Configuration",
                  strlen("QoS Configuration"));

      QoSConfigurationDialog(x,y);

      XClearWindow(display,theTeleroboticsWindow);

      XClearWindow(display,theConfigurationWindow);
      theExposeEvent.window = theConfigurationWindow;
      processExpose(&theExposeEvent);
      XFlush(display);

      return(status);
    }

  if (event->window == theCallSetUpWindow)
    {
      highlightChoice(event->window,"grey",
                      BUTTON_LEVEL1_WIDTH,BUTTON_LEVEL1_HEIGHT);

      x = DISTANCE;
      y = 100;

      setColorWithName(theTeleroboticsGC,"blue");
      XDrawString(display,
                  theTeleroboticsWindow,
                  theTeleroboticsGC,
                  DISTANCE, y-5,
                  "Call Set Up (Negotiation/Renegotiation of QoS)",
                  strlen( "Call Set Up (Negotiation/Renegotiation of QoS)"));
      WhatInfo.i_set[0]=SystemStateInfo;
      WhatInfo.i_set[1]=NOT_SPECIFIED;
      RetrieveGlobalState(&SystemState,&Scheduler,WhatInfo);

      lseek(MenuStateFD,0L,0);
      read(MenuStateFD,(char *)(&MenuControl),sizeof(int));

      if (MenuControl == START)
        {

          CallSetUp(x,y,MenuStateFD);
        }
      else
        {
          if (SystemState.net.Aneg_in.status == TAKEN)
```

```
            {
              printf("Send to slave CALL_SET_UP \n");
              MenuControl = CALL_SET_UP;
              lseek(MenuStateFD,0L,0);
              write(MenuStateFD,(char *)(&MenuControl),sizeof(int));
              send_pkt(SystemState.net.Aneg_in.id,&MenuControl,sizeof(int),err);
            }
           CallSetUp(x,y,MenuStateFD);
         }
       XClearWindow(display,theTeleroboticsWindow);

       XClearWindow(display,theCallSetUpWindow);
       theExposeEvent.window = theCallSetUpWindow;
       processExpose(&theExposeEvent);
       XFlush(display);

       return(status);
     }
   if (event->window == theStartWindow)
     {
       highlightChoice(event->window,"grey",
                       BUTTON_LEVEL1_WIDTH,BUTTON_LEVEL1_HEIGHT);

       x = DISTANCE;
       y = 100;

       setColorWithName(theTeleroboticsGC,"blue");
       XDrawString(display,
                   theTeleroboticsWindow,
                   theTeleroboticsGC,
                   DISTANCE, y-5,
                   "Start",
                   strlen( "Start"));

       WhatInfo.i_set[0]=SystemStateInfo;
       WhatInfo.i_set[1]=NOT_SPECIFIED;
       RetrieveGlobalState(&SystemState,&Scheduler,WhatInfo);

       if (SystemState.net.Aneg_in.status == TAKEN)
         {
           printf("Send to slave START \n");
           MenuControl = START;
           lseek(MenuStateFD,0L,0);
           write(MenuStateFD,(char *)(&MenuControl),sizeof(int));
           send_pkt(SystemState.net.Aneg_in.id,&MenuControl,sizeof(int),err);
           /*sleep(1); sleep 1s to allow  propagate the event START */
         }
/* Start Procedure */

       if ((pid = fork()) == 0)
         {
           rtap(BUYER);
         }
/*       signal(SIGALRM,onalarm);  */

       XClearWindow(display,theTeleroboticsWindow);

       XClearWindow(display,theStartWindow);
       theExposeEvent.window = theStartWindow;
       processExpose(&theExposeEvent);
       XFlush(display);
       printf("Leave the START Window \n");
       return(status);
     }
```

```
   if (event->window == theStopWindow)
     {
       gettimeofday(&t1neg,&tz);

       highlightChoice(event->window,"grey",
                       BUTTON_LEVEL1_WIDTH,BUTTON_LEVEL1_HEIGHT);

       x = DISTANCE;
       y = 100;
       setColorWithName(theTeleroboticsGC,"blue");
       XDrawString(display,
                   theTeleroboticsWindow,
                   theTeleroboticsGC,
                   DISTANCE, y-5,
                   "Stop",
                   strlen( "Stop"));
       WhatInfo.i_set[0]=SystemStateInfo;
       WhatInfo.i_set[1]=NOT_SPECIFIED;
       RetrieveGlobalState(&SystemState,&Scheduler,WhatInfo);
/* Stop Procedure */


       if (SystemState.net.Aneg_in.status == TAKEN)
         {

           printf("Send to slave STOP \n");
           MenuControl = STOP;
           lseek(MenuStateFD,0L,0);
           write(MenuStateFD,(char *)(&MenuControl),sizeof(int));
/*         send_pkt(SystemState.net.Aneg_in.id,&MenuControl,sizeof(int),err);*/
         }


       XClearWindow(display,theTeleroboticsWindow);

       XClearWindow(display,theStopWindow);
       theExposeEvent.window = theStopWindow;
       processExpose(&theExposeEvent);
       XFlush(display);
       gettimeofday(&t2neg,&tz);
       getproctime(t1neg,t2neg,&clock);
       printf("STOP proctime = %d \n",clock);
       return(status);
     }
   if (event->window == theHelpWindow)
     {
       highlightChoice(event->window,"grey",
                       BUTTON_LEVEL1_WIDTH,BUTTON_LEVEL1_HEIGHT);

       x = DISTANCE;
       y = 100;

       setColorWithName(theTeleroboticsGC,"blue");
       XDrawString(display,
                   theTeleroboticsWindow,
                   theTeleroboticsGC,
                   DISTANCE, y-5,
                   "Help",
                   strlen( "Help"));

/* Help procedure */

       XClearWindow(display,theTeleroboticsWindow);
```

```
      XClearWindow(display,theHelpWindow);
      theExposeEvent.window = theHelpWindow;
      processExpose(&theExposeEvent);
      XFlush(display);

      return(status);
    }
} /* -- function processButton */
/*
** alaram function which kills the child process
*/

onalarm() /* kill child when alarm arrives */
{
  kill(pid,SIGKILL);
}

/*
** findMouse gets the current mouse coords in global coordinates
*/

findMouse(displayPtr,x,y)
Display  *displayPtr;
int *x, *y;
{
  Window     theRoot, theChild;
  int        wX,wY,rootX,rootY,status;
  unsigned int   wButtons;

  status = XQueryPointer(displayPtr,
                         RootWindow(displayPtr,DefaultScreen(displayPtr)),
                         &theRoot,
                         &theChild,
                         &rootX, &rootY,
                         &wX, &wY,
                         &wButtons);

  if (status == True)
    {
      *x = wX;
      *y = wY;
    }
  else
    {
      *x = 0;
      *y = 0;
    }
} /* -- function findMouse */
```

```
/*************************************************************************/
/* Filename: colorx.c                                                    */
/* Purpose : Sets up colors                                              */
/* Author  : Klara Nahrstedt                                             */
/* Update  : 06/16/94                                                    */
/*************************************************************************/

#include "/home/klara/tele.d/include.d/xbook.h"

extern Display     *display;
extern Colormap    color;
extern int         depth;
extern unsigned long black;
extern unsigned long white;

/*
** setColorWithName
*/

setColorWithName(gc,name)
GC     gc;
char   name[];
{

 XColor RGBColor, HardwareColor;
 int    status;

   if (depth > 1)
     {
      status = XLookupColor (display,
                             color,
                             name,
                             &RGBColor,
                             &HardwareColor);
      if (status != 0)
        {
          status = XAllocColor(display,
                               color,
                               &HardwareColor);
          if (status!=0)
            {
              /* set foreground color */
              XSetForeground(display,gc,HardwareColor.pixel);
              XFlush(display);
            }
         }
    }
} /* -- function setColorWithName */
```

```
/*************************************************************************/
/* Filename: cursorx.c                                                   */
/* Purpose : initialize different cursors                                */
/* Author  : Klara Nahrstedt                                             */
/* Update  : 06/22/94                                                    */
/*************************************************************************/

#include <X11/Xlib.h>
#include <X11/cursorfont.h>

#include "/home/klara/tele.d/include.d/retta.h"

Cursor theArrowCursor;
Cursor theTextCursor;
Cursor theBusyCursor;
Cursor theQuitCursor;
Cursor theButtonCursor;
Cursor theErrorCursor;


extern Display  *display;


initCursor()
{
  theArrowCursor = XCreateFontCursor(display,
                                     XC_top_left_arrow);
  theTextCursor = XCreateFontCursor(display,
                                    XC_pencil);
  theBusyCursor = XCreateFontCursor(display,
                                    XC_watch);
  theQuitCursor = XCreateFontCursor(display,
                                    XC_gumby);
  theButtonCursor = XCreateFontCursor(display,
                                    XC_hand1);
  theErrorCursor = XCreateFontCursor(display,XC_pirate);

} /* -- function initCursor */

freeCursors()
{
  XFreeCursor(display, theArrowCursor);
  XFreeCursor(display,theTextCursor);
  XFreeCursor(display,theBusyCursor);
  XFreeCursor(display,theQuitCursor);
  XFreeCursor(display,theButtonCursor);
  XFreeCursor(display,theErrorCursor);

  XFlush(display);
}/* --function freeCursor */
```

```
/*********************************************************************/
/* Filename: dialogx.c                                               */
/* Purpose : Pop-up window dialog box                                */
/* Author  : Klara Nahrstedt                                         */
/* Update  : 06/16/94                                                */
/*********************************************************************/

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>
#include <X11/keysymdef.h>

#include "/home/klara/tele.d/include.d/retta.h"

extern Display *display;
extern Cursor theArrowCursor;
extern Cursor theTextCursor;
extern Window theRootWindow;
extern Window theTeleroboticsWindow;

#define MAX_CHARS   80
#define MIN_WIDTH    200
#define MIN_HEIGHT  100


Window    theDialogWindow;
GC        theDialogGC;
char theDialogText[MAX_CHARS + 5];
char theDialogMessage[MAX_CHARS + 5];
char theDialogOKMsg[MAX_CHARS + 5];
char theDialogCanMsg[MAX_CHARS + 5];

Window theDOKWindow;
GC     theDOKGC;
Window theDStringWindow;
GC     theDStringGC;
Window theDCancelWindow;
GC     theDCancelGC;

/* stringDialog pops up a transient window that asks the user to enter
** in a file name
*/

stringDialog(x,y, theMessage, theOKText, theCancelText)
int x,y;   /* -- Upper left corner for dialog */
char theMessage[];  /* -- the Prompt */
char theOKText[];   /* -- the OK selection text */
char theCancelText[]; /* theCancel selection text */


{
  int theChoice = 0; /* standard is cancel */
  int width,height, value;

  value = textWidth(theMessage, TEXT1_FONT);
  if (value > MIN_WIDTH)
     width = value + 30;
  else
    width = MIN_WIDTH;

  value = (textHeight(TEXT1_FONT) + 10)* 3;
  if (value > MIN_HEIGHT)
    height = value;
  else
    height = MIN_HEIGHT;

  /* Open Windows */

  initDialogWindows(x,y,width,height);

  /* set up display info */

  if (strlen (theMessage) < MAX_CHARS)
    {
      strcpy(theDialogMessage, theMessage);
    }
  else
    {
      strncpy(theDialogMessage, theMessage, MAX_CHARS);
    }

  if (strlen( theOKText) < MAX_CHARS)
    {
      strcpy(theDialogOKMsg, theOKText);
    }
  else
    {
      strncpy(theDialogOKMsg,theOKText,MAX_CHARS);
    }

  if (strlen(theCancelText) < MAX_CHARS)
    {
      strcpy(theDialogCanMsg, theCancelText);
    }
  else
    {
      strncpy(theDialogCanMsg, theCancelText, MAX_CHARS);
    }

/* Display Dialog Info */

  theDialogText[0] = '\0';  /* -- NULL string to start out */
  displayDialog(theDialogWindow);
  displayDialog(theDStringWindow);
  displayDialog(theDCancelWindow);
  displayDialog(theDOKWindow);

/* Handle Dialog Window Events */
  theChoice = (-1);

  while (theChoice == -1)
    {
      theChoice = dialogEventLoop();
    }


/* Close Window and free resources */

  freeDialog();

/* return Choice */

  return (theChoice);
} /* -- function stringDialog */

/*
** initDialogWindows creates all the pop-up dialog box windows
** and GCs in their correct position
```

```
*/

initDialogWindows(x,y,width,height)
int x,y,width,height ;
{

  Window    openWindow();

  /* Main Dialog Box Window */

  theDialogWindow = openWindow(x,y,width,height,
                               POP_UP_WINDOW, /* -- IS a POP-UP */
                               "File Requestor",
                               NORMAL_STATE,
                               theRootWindow,
                               &theDialogGC,0);

  associateFont(theDialogGC,TEXT1_FONT);
  initEvents(theDialogWindow, IN_PALETTE);

  theDOKWindow = openWindow(10,
                            2*(height/3),
                            (width/3), (height/3) -10,
                            NORMAL_WINDOW,
                            "FileRequestor",
                            NORMAL_STATE,
                            theDialogWindow,
                            &theDOKGC,1);

  associateFont(theDOKGC,TEXT1_FONT);
  initEvents(theDOKWindow, IN_PALETTE);

  theDCancelWindow = openWindow(width -10-(width/3),
                               2*(height/3),
                               (width/3), (height/3) -10,
                               NORMAL_WINDOW,
                               "File Requestor",
                               NORMAL_STATE,
                               theDialogWindow,
                               &theDCancelGC,1);

  associateFont(theDCancelGC,TEXT1_FONT);
  initEvents(theDCancelWindow,IN_PALETTE);

  theDStringWindow = openWindow(10,
                               textHeight(TEXT1_FONT)+10,
                               width -20, (height/3) -10,
                               NORMAL_WINDOW,
                               "File Requestor",
                               NORMAL_STATE,
                               theDialogWindow,
                               &theDStringGC,0);
  associateFont(theDStringGC, TEXT1_FONT);
  initEvents(theDStringWindow, IN_PALETTE);

/* Set up cursors */

  XDefineCursor(display, theDialogWindow, theArrowCursor);
  XDefineCursor(display, theDCancelWindow, theArrowCursor);
  XDefineCursor(display, theDOKWindow, theArrowCursor);

  XDefineCursor(display, theDStringWindow, theTextCursor);

  XFlush(display);

}/* -- function initDialogWindows */

/*
** displayDialog
*/

displayDialog(window)
Window  window;
{
  int y;

  y = textHeight(TEXT1_FONT) + 5;

  if (window == theDialogWindow)
    {
      XDrawString(display, window, theDialogGC,
                  10,y,
                  theDialogMessage,
                  strlen(theDialogMessage));
    }

  if (window == theDStringWindow)
    {
      XDrawString(display, window, theDStringGC,
                  10, y,
                  theDialogText,
                  strlen(theDialogText));
    }

  if (window == theDOKWindow)
    {
      XDrawString(display, window, theDOKGC,
                  10,y,
                  theDialogOKMsg,
                  strlen(theDialogOKMsg));
    }

  if (window == theDCancelWindow)
    {
      XDrawString(display, window, theDCancelGC,
                  10,y,
                  theDialogCanMsg,
                  strlen(theDialogCanMsg));
    }
  XFlush(display);
}/* -- function displayDialog */
/*
** dialogEventLoop handles all the dialog events
*/

dialogEventLoop()

{
  int   status = (-1);
  XEvent    event;

  XNextEvent(display, &event);

  switch(event.type)
    {
    case ConfigureNotify:
    case Expose:
    case MapNotify:
```

```
      displayDialog(event.xany.window);
      break;
    case ButtonPress:
      if (event.xbutton.window == theDOKWindow)
        {
          fillRectangle(theDOKWindow, theDOKGC,
                        0,0,200,200);
          displayDialog(event.xbutton.window);
          status =1;
        }

      if (event.xbutton.window == theDCancelWindow)
        {
          fillRectangle(theDCancelWindow, theDCancelGC,
                        0,0,200,200);
          displayDialog(event.xbutton.window);
          status = 0;
        }

      break;
    case KeyPress:
      dialogKeyPress(&event);
      break;
    }
  return(status);
} /* -- function dialogEventLoop */
/*
** dialogKeyPress handles keybord input inot the stringDialog
*/

dialogKeyPress (event)
XKeyEvent   *event;
{
  int   length, l,i;
  int   theKeyBufferMaxLen = 64;
  int   theKeyBuffer[65];
  KeySym  theKeySym;
  XComposeStatus   theComposeStatus;

   for (i=0; i<65; i++)
   theKeyBuffer[i] =0;
  length = XLookupString(event,
                         theKeyBuffer,
                         theKeyBufferMaxLen,
                         &theKeySym,
                         &theComposeStatus);

  printf("KeyBuffer is %s \n", theKeyBuffer);

  l = strlen(theDialogText);

  if ((theKeySym >= ' ') &&
      (theKeySym <= '~') &&
      (length > 0))
    {
      if ((l+strlen(theKeyBuffer)) < MAX_TEXT_LENGTH)
        {
          strcat(theDialogText, theKeyBuffer);
          displayDialog(theDStringWindow);
        }
    }
  else
    {
      switch(theKeySym)
        {
        case XK_BackSpace:
        case XK_Delete:
          if (l>=1)
            {
              XClearWindow(display, theDStringWindow);
              l--;
              theDialogText[l] = '\0';
              displayDialog(theDStringWindow);
              XFlush(display);
            }
          break;
        default:;
        }
    }
}/* -- function dialogkeyPress */

/*
** freeDialog
*/

freeDialog()
{
  XFreeGC(display, theDialogGC);
  XFreeGC(display, theDOKGC);
  XFreeGC(display, theDStringGC);
  XFreeGC(display, theDCancelGC);

/*
** Destroy all windows
*/

  XDestroySubwindows(display, theDialogWindow);
  XDestroyWindow(display, theDialogWindow);
  XFlush(display);
}/* -- function freeDialog */
```

```
/*************************************************************************/
/* Filename: drawx.c                                                     */
/* Purpose : X11 drawing functions                                       */
/* Author  : Klara Nahrstedt                                             */
/* Update  : 06/16/94                                                    */
/*************************************************************************/

#include "/home/klara/tele.d/include.d/xbook.h"

/* -- external globals from initx.c */
#define FULL_CIRCLE        (360*64)
#define START_CIRCLE        0

extern Display   *display;

/* drawLine
** draw line from (x1,y1) to (x2,y2) in the window, using graphics
** context gc
*/

drawLine(window,gc,x1,y1,x2,y2)
Window window;   /* -- the window to draw it in */
GC     gc;       /* -- graphics context */
int x1,y1;       /* starting location */
int x2,y2;       /* ending location */

{
   XDrawLine(display,
             window,
             gc,
             x1,y1,
             x2,y2);
 } /* -- function drawLine */

/*
** drawRectangle
*/

drawRectangle(window,gc,x,y,width,height)
Window window;
GC   gc;
int  x,y;     /* starting location, upper left corner */
int width,height; /* size of the rectangle */

{
   XDrawRectangle(display,
                  window,
                  gc,
                  x,y,
                  width,height);
 } /* function drawrectangle */

/*
** fillRectangle
*/

fillRectangle(theDrawable,gc,x,y,width,height)
Drawable   theDrawable;
GC         gc;
int        x,y;
int        width,height;
{
    XFillRectangle(display,
                   theDrawable,
                   gc,
                   x,y,
                   width,height);
  } /* --function fillRectangle */

/*
** drawOval
*/

drawOval(window,gc,x,y,width,height)
Window     window;
GC         gc;
int        x,y;
int        width,height;
{
   XDrawArc(display,
            window,
            gc,
            x,y,
            width,height,
            START_CIRCLE,
            FULL_CIRCLE);

 } /* -- function drawOval */

/*
** fillOval
*/

fillOval(window,gc,x,y,width,height)
Window  window;
GC      gc;
int     x,y;
int     width,height;
{
   XFillArc(display,
            window,
            x,y,
            width,height,
            START_CIRCLE,
            FULL_CIRCLE);

 } /* --fillOval */
```

```
/*************************************************************************/
/* Filename: errorx.c                                                    */
/* Purpose : X Error handlers for the Telerobotics application.          */
/* Author  : Klara Nahrstedt                                             */
/* Update  : 06/16/94                                                    */
/*************************************************************************/

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xproto.h>


/*
** setErrorHandler sets up the porgramm's erro handler functions
*/

setErrorHandler()
{
  int errorHandler();
  int fatalErrorHandler();

/* Set up the normal error handler for things like bad window ID's */

  XSetErrorHandler(errorHandler);

/* Set up the fatal error handler for a broken connection with
** the Xserver */

  XSetErrorHandler(fatalErrorHandler);

} /* -- function setErrorHandler */

/*
** errorHandler handles non-fatal X errors
*/

errorHandler(display, theErrorEvent)
Display    *display;
XErrorEvent   *theErrorEvent;

{
  int bufferLength = 120;
  char theBuffer[130];

  XGetErrorText(display,
                theErrorEvent->error_code,
                theBuffer,
                bufferLength);

  fprintf(stderr,
          "X error: %s\n", theBuffer);

  fprintf(stderr,
          "Serial number of request: %ld Op Code : %d.%d Error Code: %d\n",
          theErrorEvent->serial,
          theErrorEvent->request_code,
          theErrorEvent->minor_code,
          theErrorEvent->error_code);

  fprintf(stderr,
          "Resource ID of failed request: %ld on display %s.\n",
          theErrorEvent->resourceid,
          DisplayString(display));

} /* -- function errorHandler */

/*
** fatalErrorHandler takes care of fatal X errors
*/

fatalErrorHandler(display)
Display  *display;
{
  fprintf(stderr,
          " X Error: Fatal IO error on display %s.\n",
          DisplayString(display));

  fprintf(stderr,
          "Bailing out near line one. \n");

  exit(1);
}/* -- function fataleErrorHandler*/
```

```
/*************************************************************************/
/* Filename: eventx.c                                                    */
/* Purpose : starts the main event loop of the teleopration application  */
/* Author  : Klara Nahrstedt                                             */
/* Update  : 06/16/94                                                    */
/*************************************************************************/

#include <X11/Xlib.h>
#include <X11/Xutil.h>

#include "/home/klara/tele.d/include.d/retta.h"


extern Display *display;


#define EV_MASK (KeyPressMask       | \
                 ButtonPressMask    | \
                 ExposureMask       | \
                 PointerMotionMask  | \
                 StructureNotifyMask)
/*
** eventLoop blocks awaiting an event from X.
*/

int eventLoop(StateFD)
int StateFD;
{
   XEvent     event;

   XNextEvent(display,&event);

   switch(event.type)
     {
     case Expose:
     case ConfigureNotify:
     case MapNotify:
                  processExpose(&event);
          break;
     case ButtonPress:
         return(processButton(&event,StateFD));
         break;
/*
    case ButtonRelease:
                  processRelease(&event);
                  break;
*/
     case KeyPress:
          return(processKeyPress(&event));
          break;

               }
   return(1);
 } /* -- function eventLoop */


/*
** initEvents
*/

initEvents(window,inPalette)
Window   window;
int  inPalette;
{
  if (inPalette == IN_PALETTE)
    {
      XSelectInput(display,
                   window,
                   EV_MASK);
    }
  else
    {
      XSelectInput(display,window,
                   (EV_MASK |
                    ButtonMotionMask | ButtonReleaseMask));
    }

 } /* -- function initEvents */
```

```
/*************************************************************************/
/* Filename: filex.c                                                     */
/* Purpose : processing of files with X interface                        */
/* Author  : Klara Nahrstedt                                             */
/* Update  : 06/16/94                                                    */
/*************************************************************************/

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <stdio.h>

#include "/home/klara/tele.d/include.d/retta.h"
char theCurrentFileName[120];

/*
extern char    theTeleroboticsTitle[MAX_TEXT_LENGTH + 5];

extern Display display;
extern Window  theRootWindow;
extern Window theTeleroboticsindow;

*/

loadFile(theFileName)
char theFileName[];
{

printf("reading in file %s\n", theFileName);

}/* --function loadFile*/


setFileName(theFileName)
char theFileName[];

{
  if (strlen(theFileName) < 110)
    {
      strcpy(theCurrentFileName, theFileName);
    }
} /* -- function setFileName*/
```

```
/*************************************************************************/
/* Filename : initx.c                                                    */
/* Purpose  : Initialization code to talk to the X server                */
/* Author   : Klara Nahrstedt                                            */
/* Update   : 06/16/94                                                   */
/*************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>


Display  *display;            /* -- which display */
int       screen;             /* -- which screen on display */
int       depth;
unsigned long black;
unsigned long white;
Window   theRootWindow;         /* System wide parent window */
Colormap color;                 /* default System color map */
Visual  *visual;

/*
** initX()
** sets up the connection to the X server and stores information
** about the environment
*/

initX(displayName)
char *displayName;

{
/*
** establish a connection to the X server. The connection
** is asked for on the local server for the local display
*/
  display = XOpenDisplay(displayName);

  if (display == NULL)
    {
      perror("XOpenDisplay");
      exit(1);
    }
/*
** check for the default screen and color plane depth.
** if depth == 1 , then we have a monochrome system.
*/

  screen = DefaultScreen(display);
  depth = DefaultDepth(display,screen);
  black = BlackPixel(display,screen);
  white = WhitePixel(display,screen);
  theRootWindow = RootWindow(display,screen);
  color = DefaultColormap(display,screen);
  visual = XDefaultVisual(display,screen);


} /* --function initX */
```

```
/*************************************************************************/
/* Filename :  keyx.c                                                    */
/* Purpose  : Handles keybord events for the X program                   */
/* Author   : Klara Nahrstedt                                            */
/* Update   : 06/16/94                                                   */
/*************************************************************************/

#include "/home/klara/tele.d/include.d/xbook.h"

extern Display  *dipslay;

/*
** processKeyPress handles the keybord input
*/

processKeyPress(event)
XKeyEvent *event;
{
   int length;
   int keyBufferMaxLen = 64;
   int keyBufferLength;
   char keyBuffer[65];
   KeySym theKeySym;
   XComposeStatus composeStatus;

   bzero((char *)(keyBuffer),65);
   length = XLookupString(event,
                          keyBuffer,
                          keyBufferMaxLen,
                          &theKeySym,
                          &composeStatus);
   if (event->state & Mod1Mask)  /* Meta key */
     {
       switch(keyBuffer[0])
         {
          case 'Q':
          case 'q': printf("Meta-Q hit\n");
                    return(0);
                    break;
          }
       printf("META [%s}\n", keyBuffer);
       return(1);
     }

   if ((theKeySym >= ' ') &&
       (theKeySym <= '~') &&
       (length > 0))
     {
       printf("ASCII key was hit: [%s]\n", keyBuffer);
       if ((keyBuffer[0] == 'q') ||
           (keyBuffer[0] == 'Q'))
         {
           return(0);
         }
     }
   else
     {
       switch(theKeySym)
         {
         case XK_Return : printf("Return\n"); break;
         case XK_BackSpace: printf("BackSpace\n"); break;
         default :;
         }
     }
   return(1);
} /* -- functin processKeyPress */
```

```
/*************************************************************************/
/* Filename : listx.c                                                    */
/* Purpose  : display/list  the possible parameter list                  */
/* Author   : Klara Nahrstedt                                            */
/* Update   : 06/16/94                                                   */
/*************************************************************************/

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

#include "/home/klara/tele.d/include.d/retta.h"
#include "/home/klara/tele.d/include.d/defs.h"
#include "/home/klara/tele.d/include.d/comm.h"

extern Display *display;

extern Cursor theArrowCursor;
extern Cursor theButtonCursor;

extern Window theRootWindow;
extern Window theCompressionWindow;
extern Window theInOutMQWindow;

extern APP_QOS MasterInputParam;
extern APP_QOS MasterOutputParam;
extern APP_QOS SlaveInputParam;
extern APP_QOS SlaveOutputParam;


/* List elements of Compression Choice */
Window theListWindow;
GC     theListGC;
Window theNoneWindow;
GC     theNoneGC;
Window theJPEGWindow;
GC     theJPEGGC;
Window theMPEGWindow;
GC     theMPEGGC;


initListCompression(x,y,side,inout)
int x,y;
int side,inout;
{
  int theChoice = (-1);

  initListCompressionWindows(x,y);

  displayListContent(theNoneWindow);
  displayListContent(theJPEGWindow);
  displayListContent(theMPEGWindow);

  while (theChoice == -1)
    {
      theChoice = listEventLoop(side,inout);
    }

freeListCompressionWindows();
}

initListCompressionWindows(x,y)
int x,y;
{
Window openWindow();
int n=3; /* number of list items */
int NoneButtonW = 0;

theListWindow = openWindow(x,y,
                           200,200,
                           NORMAL_WINDOW,
                           "Select Window",
                           NORMAL_STATE,
                           theInOutMQWindow,
                           &theListGC,
                           NoneButtonW);
associateFont(theListGC,TEXT1_FONT);
initEvents(theListWindow, IN_PALETTE);
XDefineCursor(display, theListWindow, theArrowCursor);

y += TEXT_WINDOW_HEIGHT;
x += 5;
theNoneWindow = openWindow(x,y,
                           TEXT_WINDOW_WIDTH,
                           TEXT_WINDOW_HEIGHT,
                           NORMAL_WINDOW,
                           "Select Window",
                           NORMAL_STATE,
                           theListWindow,
                           &theNoneGC,
                           NoneButtonW);
associateFont(theNoneGC, TEXT1_FONT);
initEvents(theNoneWindow,IN_PALETTE);

XDefineCursor(display, theNoneWindow, theButtonCursor);

y +=TEXT_WINDOW_HEIGHT;

theJPEGWindow = openWindow(x,y,
                           TEXT_WINDOW_WIDTH,
                           TEXT_WINDOW_HEIGHT,
                           NORMAL_WINDOW,
                           "Select Window",
                           NORMAL_STATE,
                           theListWindow,
                           &theJPEGGC,
                           NoneButtonW);
associateFont(theJPEGGC, TEXT1_FONT);
initEvents(theJPEGWindow,IN_PALETTE);

XDefineCursor(display, theJPEGWindow, theButtonCursor);

y +=TEXT_WINDOW_HEIGHT;

theMPEGWindow = openWindow(x,y,
                           TEXT_WINDOW_WIDTH,
                           TEXT_WINDOW_HEIGHT,
                           NORMAL_WINDOW,
                           "Select Window",
                           NORMAL_STATE,
                           theListWindow,
                           &theMPEGGC,
                           NoneButtonW);
associateFont(theMPEGGC, TEXT1_FONT);
initEvents(theMPEGWindow,IN_PALETTE);

XDefineCursor(display, theMPEGWindow, theButtonCursor);
```

```
  XFlush(display);

} /* -- function initListCompresionWindows*/

displayListContent(window)
Window  window;

{
  int y;
  y = textHeight(TEXT1_FONT);

  if (window == theNoneWindow)
    {
      XDrawString( display, window, theNoneGC,
                  5,y,
                  "None",
                  strlen("None"));
    }

  if (window == theJPEGWindow)
    {
      XDrawString( display, window, theJPEGGC,
                  5,y,
                  "JPEG",
                  strlen("JPEG"));
    }

  if (window == theMPEGWindow)
    {
      XDrawString( display, window, theMPEGGC,
                  5,y,
                  "MPEG",
                  strlen("MPEG"));
    }
  XFlush(display);

}/* displayListContent*/

listEventLoop(side,inout)
{
  int   status = (-1);
  XEvent    event;

  XNextEvent(display, &event);

  switch(event.type)
    {
    case ConfigureNotify:
    case Expose:
    case MapNotify:
      displayListContent(event.xany.window);
      break;
    case ButtonPress:
      if (event.xbutton.window == theNoneWindow)
        {
          if (side == MASTER && inout == INPUT)
            {
              MasterInputParam.stream[VIDEO].medium.app_spec.comp_spec.name = NONE;
              XClearWindow(display, theCompressionWindow);
              displayListContent(theCompressionWindow);
            }
          if (side == MASTER && inout == OUTPUT)
            {
              MasterOutputParam.stream[VIDEO].medium.app_spec.comp_spec.name = NONE;
              XClearWindow(display, theCompressionWindow);
              displayListContent(theCompressionWindow);
            }
          if (side == SLAVE && inout == INPUT)
            {
              SlaveInputParam.stream[VIDEO].medium.app_spec.comp_spec.name=NONE;
              XClearWindow(display, theCompressionWindow);
              displayListContent(theCompressionWindow);
            }
          if (side == SLAVE && inout == INPUT)
            {
              SlaveOutputParam.stream[VIDEO].medium.app_spec.comp_spec.name = NONE;
              XClearWindow(display, theCompressionWindow);
              displayListContent(theCompressionWindow);
            }

          status =1;
        }

      if (event.xbutton.window == theJPEGWindow)
        {
          status = 0;
        }
      if (event.xbutton.window == theMPEGWindow)
        {
          status = 0;
        }

      break;

    }
  return(status);
} /* -- function lisEventLoop */

freeListCompressionWindows()
{

    XFreeGC(display, theNoneGC);
    XFreeGC(display, theJPEGGC);
    XFreeGC(display, theMPEGGC);
    XFreeGC(display, theListGC);

    XDestroySubwindows(display, theListWindow);
    XDestroyWindow(display, theListWindow);
  }
```

```
/*************************************************************************/
/* Filename : quitx.c                                                    */
/* Purpose  : close down X                                               */
/* Author   : Klara Nahrstedt                                            */
/* Update   : 06/16/94                                                   */
/*************************************************************************/

#include "/home/klara/tele.d/include.d/xbook.h"
extern Display *display;


/*
** quitX()
** closes the connection to the X server
*/

quitX()
{
  freeWindowsAndGCs();

  freeCursors();

  freeFonts();

  XFlush(display);

  XCloseDisplay(display);
} /* -- function quitX */
```

```
/*************************************************************************/
/* Filename : setUpQoS.c                                                 */
/* Purpose  : set up x-windows (TX1) for input of QoS parameters         */
/* Author   : Klara Nahrstedt                                            */
/* Update   : 06/16/94                                                   */
/*************************************************************************/


#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>
#include <X11/keysymdef.h>

#include "/home/klara/tele.d/include.d/retta.h"

extern Display *display;

extern Cursor theArrowCursor;
extern Cursor theTextCursor;
extern Cursor theQuitCursor;
extern Cursor theButtonCursor;
extern Cursor theBusyCursor;

extern Window theDCWindow;
extern Window theRootWindow;

extern GC theDCGC;

Window theTX1QoSWindow;
Window theTX1DoneWindow;
Window theTX1CancelWindow;


/* subwindows of TX1QoS Window */

Window theTX11Window;
Window theTX12Window;

GC theTX1QoSGC;
GC theTX1DoneGC;
GC theTX1CancelGC;


GC theTX11GC;
GC theTX12GC;


QoSsetUpMasterSide(x,y)
int x,y;
{
  int width,height;
  int theChoice;

  width = MAIN_WINDOW_WIDTH - 4*DISTANCE;
  height = MAIN_WINDOW_HEIGHT - 100 - 5*DISTANCE;

  initTX1Windows(x,y, width,height);

  displayTX1(theTX1CancelWindow);
  displayTX1(theTX1DoneWindow);


  theChoice = -1;

  while (theChoice == -1)
    {
      theChoice = TX1EventLoop();
    }

freeTX1Windows();

} /* -- function QoSsetUpMasterSide */

initTX1Windows(x,y,width,height)
int x,y,width,height;
{

  Window openWindow();

  int ButtonW =1;
  int NoneButtonW =0;

  theTX1QoSWindow = openWindow( x,y,
                                width,height,
                                NORMAL_WINDOW,
                                "QoSSetUpMaster",
                                NORMAL_STATE,
                                theDCWindow,
                                &theTX1QoSGC,
                                NoneButtonW);
  associateFont(theTX1QoSGC,TEXT1_FONT);


   initEvents(theTX1QoSWindow,IN_PALETTE);

  XDefineCursor(display,theTX1QoSWindow,theArrowCursor);


  x = DISTANCE;
  y = height - 2*DISTANCE;

  theTX1CancelWindow = openWindow( x,y,
                                BUTTON_LEVEL2_WIDTH,
                                 BUTTON_LEVEL2_HEIGHT,
                                NORMAL_WINDOW,
                                "QoSSetUpMaster",
                                NORMAL_STATE,
                                theTX1QoSWindow,
                                &theTX1CancelGC,
                                ButtonW);
  associateFont(theTX1CancelGC,TEXT1_FONT);

  initEvents(theTX1CancelWindow,IN_PALETTE);

  XDefineCursor(display,theTX1CancelWindow,theQuitCursor);

  x += (BUTTON_LEVEL2_WIDTH + DISTANCE);

  theTX1DoneWindow = openWindow( x,y,
                                BUTTON_LEVEL2_WIDTH,
                                 BUTTON_LEVEL2_HEIGHT,
                                NORMAL_WINDOW,
                                "QoSSetUpMaster",
                                NORMAL_STATE,
                                theTX1QoSWindow,
                                &theTX1DoneGC,
                                ButtonW);
```

```
  associateFont(theTX1DoneGC,TEXT1_FONT);

  initEvents(theTX1DoneWindow,IN_PALETTE);

  XDefineCursor(display,theTX1DoneWindow,theButtonCursor);

  x =DISTANCE;
  y =DISTANCE;


  initTX1TextWindows(x,y);


 XFlush(display);
}/* -- function initTX1Windows */

/*
** initTX1TextWindows
*/

initTX1TextWindows(x,y)
int x,y;
{
  int NoneButtonW = 0;
  int ButtonW = 1;


  theTX11Window = openWindow( x,y,
                              BUTTON_LEVEL3_WIDTH,
                              BUTTON_LEVEL3_HEIGHT,
                             NORMAL_WINDOW,
                             "MediaQuality",
                             NORMAL_STATE,
                             theTX1QoSWindow,
                             &theTX11GC,
                             ButtonW);
  associateFont(theTX11GC,TEXT1_FONT);
  initEvents(theTX11Window,IN_PALETTE);

  XDefineCursor(display,theTX11Window,theButtonCursor);

      XDrawString(display,theTX1QoSWindow, theTX1QoSGC,
                  x + BUTTON_LEVEL3_WIDTH + DISTANCE,
                  y + BUTTON_LEVEL3_HEIGHT,
                  "Media Quality Parameters",
                  strlen("Media Quality Parameters"));

  y += (BUTTON_LEVEL3_HEIGHT + DISTANCE);

  theTX12Window = openWindow( x,y,
                             BUTTON_LEVEL3_WIDTH,
                             BUTTON_LEVEL3_HEIGHT,
                               NORMAL_WINDOW,
                               "MediaQuality",
                               NORMAL_STATE,
                               theTX1QoSWindow,
                               &theTX12GC,
                               ButtonW);
  associateFont(theTX12GC,TEXT1_FONT);
  initEvents(theTX12Window,IN_PALETTE);

  XDefineCursor(display,theTX12Window,theButtonCursor);

  XDrawString(display,theTX1QoSWindow, theTX1QoSGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Media Relations",
              strlen("Media Relations"));
  XFlush(display);

}/* -- function initTX1TextWindows */


/*
** displayTX1
*/

displayTX1(window)
Window window;

{
  int xt,yt;
  int y;

  y = textHeight(TEXT1_FONT) + 5;

  if (window == theTX1DoneWindow)
    {
      XDrawString(display,window, theTX1DoneGC,
                  10,y,
                  "Done",
                  strlen("Done"));
    }

  if (window == theTX1CancelWindow)
    {
      XDrawString(display,window, theTX1CancelGC,
                  10,y,
                  "Cancel",
                  strlen("Cancel"));
    }

  if (window == theTX1QoSWindow)
    {
      xt = DISTANCE;
      yt = DISTANCE;

      initTX1TextWindows(xt,yt);
    }
  if (window == theTX11Window)
    {
      XClearWindow(display,theTX1QoSWindow);
      freeTX1Text();

     setColorWithName(theTX1QoSGC, "blue");
      XDrawString(display,theTX1QoSWindow,
                  theTX1QoSGC,
                  DISTANCE, DISTANCE -5,
                  "Media Quality Parameter",
                  strlen("Media Quality Parameter"));
      xt = DISTANCE;
      yt = DISTANCE;

     QoSmediaQualityMasterSide(xt,yt);
      }

  if (window == theTX12Window)
```

```
    {
      XClearWindow(display,theTX1QoSWindow);
      freeTX1Text();

     setColorWithName(theTX1QoSGC, "blue");
      XDrawString(display,theTX1QoSWindow,
                  theTX1QoSGC,
                  DISTANCE, DISTANCE -5,
                  "Media Relations",
                  strlen("Media Relations"));
      }

  XFlush(display);
} /* -- function displayTX1 */

/*
** TX1EventLoop
*/

TX1EventLoop()
{
  int status = (-1);
  XEvent event;

  XNextEvent(display, &event);

  switch(event.type)
    {
    case ConfigureNotify:
    case Expose:
    case MapNotify:
      displayTX1(event.xany.window);
      break;
    case ButtonPress:
      if (event.xbutton.window == theTX1DoneWindow)
        {
          highlightChoice(theTX1DoneWindow,
                          "blue",
                          BUTTON_LEVEL2_WIDTH,
                          BUTTON_LEVEL2_HEIGHT);
          displayTX1(event.xbutton.window);
          status = 1;
        }

       if (event.xbutton.window == theTX1CancelWindow)
        {
          highlightChoice(theTX1CancelWindow,
                          "blue",
                          BUTTON_LEVEL2_WIDTH,
                          BUTTON_LEVEL2_HEIGHT);
          displayTX1(event.xbutton.window);
          status = 0;
        }

      if (event.xbutton.window == theTX11Window)
        {
          highlightChoice(theTX11Window,
                          "blue",
                          BUTTON_LEVEL3_WIDTH,
                          BUTTON_LEVEL3_HEIGHT);
          displayTX1(event.xbutton.window);
          XClearWindow(display,theTX1QoSWindow);
          setColorWithName(theTX1QoSGC,"black");
          displayTX1(theTX1QoSWindow);
        }

      if (event.xbutton.window == theTX12Window)
        {
          highlightChoice(theTX12Window,
                          "blue",
                          BUTTON_LEVEL3_WIDTH,
                          BUTTON_LEVEL3_HEIGHT);
          displayTX1(event.xbutton.window);
          XClearWindow(display,theTX1QoSWindow);
          setColorWithName(theTX1QoSGC,"black");
          displayTX1(theTX1QoSWindow);
        }

      break;
    }
  return(status);
}/* -- function TX1EventLoop */



freeTX1Windows()
{

  XFreeGC(display, theTX1QoSGC);
  XFreeGC(display, theTX1DoneGC);
  XFreeGC(display,theTX1CancelGC);

  XDestroySubwindows(display,theTX1QoSWindow);
  XDestroyWindow(display,theTX1QoSWindow);
}

freeTX1Text()
{

  XFreeGC(display,theTX11GC);
  XFreeGC(display,theTX12GC);

  XDestroyWindow(display,theTX11Window);
  XDestroyWindow(display,theTX12Window);
}
```

```
/*************************************************************************/
/* Filename : textx.c                                                    */
/* Purpose  : Text Drawing Routines                                      */
/* Author   : Klara Nahrstedt                                            */
/* Update   : 06/16/94                                                   */
/*************************************************************************/

#include <X11/Xlib.h>
#include "/home/klara/tele.d/include.d/retta.h"

XFontStruct   *theText1Font;
XFontStruct   *theText2Font;

extern Display  *display;

/*
** associateFont associates (sets) the given GC to use one of the
** two fonts for the application
*/

associateFont(gc,whichFont)
GC gc;
int whichFont;
{
  if (whichFont == TEXT1_FONT)
    {
      XSetFont(display,
               gc,
               theText1Font->fid);
    }
  else
    {
      XSetFont(display,
               gc,
               theText2Font->fid);
    }
} /* -- function associateFont*/

/*
** freeFonts frees up the font resources used by the application
*/

freeFonts()
{
  XFreeFont(display, theText1Font);
  XFreeFont(display, theText2Font);
}/* -- function freeFont */



/*
** initFont()
** loads in the given fonts into the vars theText1Font and theText2Font
*/

XFontStruct *
initFont(font1Name,font2Name)
char font1Name[];
char font2Name[];
{
   theText1Font = XLoadQueryFont(display,font1Name);
   theText2Font = XLoadQueryFont(display,font2Name);
 }/* --function initFont */

/*
** textHeight returns the max height of the tallest characters
** in the given font
*/

textHeight(whichFont)
int whichFont;

{
  int theHeight;

  if (whichFont == TEXT1_FONT)
    {
      theHeight = theText1Font->ascent +
        theText2Font->descent;
    }
  else
    {
      theHeight = theText2Font-> ascent +
        theText2Font->descent;
    }

  return(theHeight);
} /* -- function textHeight */


/*
** textWidth returns the width of the given string in the given font
*/

textWidth(theString, whichFont)
char theString[];
int whichFont;
{
  int theLength;

  if (whichFont == TEXT1_FONT)
    {
      theLength = XTextWidth(theText1Font,
                             theString,
                             strlen(theString));
    }
  else
    {
      theLength = XTextWidth(theText2Font,
                             theString,
                             strlen(theString));
    }
  return(theLength);
} /* textWidth */
```

```
/*************************************************************************/
/* Filename : utilx.c                                                    */
/* Purpose  : contains major x-functions to run the retta                */
/* Author   : Klara Nahrstedt                                            */
/* Update   : 06/16/94                                                   */
/*************************************************************************/

#include <X11/Xlib.h>
#include <X11/Xutil.h>

#include "/home/klara/tele.d/include.d/retta.h"

extern Display     *display;
extern Window      theRootWindow;
extern unsigned long    black;
extern unsigned long    white;

extern Cursor theArrowCursor;
extern Cursor theTextCursor;
extern Cursor theBusyCursor;
extern Cursor theQuitCursor;
extern Cursor theButtonCursor;

char    theTeleroboticsTitle[MAX_TEXT_LENGTH + 5];

Window theTeleroboticsWindow;
/* subwindows of telerobotics window */

Window theConfigurationWindow;
Window theExitWindow;
Window theCallSetUpWindow;
Window theStartWindow;
Window theStopWindow;
Window theHelpWindow;

GC theTeleroboticsGC;
GC theConfigurationGC;
GC theExitGC;
GC theCallSetUpGC;
GC theStartGC;
GC theStopGC;
GC theHelpGC;


/*
** initWindows
*/


initWindows(geometry, title, iconicState)
char geometry[];
char title[];
int iconicState;

{
  int geometryStatus;
  int x,y,x1,y1,width,height;
  int openWindow();


  geometryStatus = XParseGeometry(geometry, &x,&y,&width,&height);

  if ( !(geometryStatus & XValue))
    x = 10;
  if ( ! (geometryStatus & YValue))
    y = 10;
  if ( ! (geometryStatus & WidthValue))
    width = MAIN_WINDOW_WIDTH;
  if (! ( geometryStatus & HeightValue))
    height = MAIN_WINDOW_HEIGHT;

  if (strlen(title) < 100)
    {
       strcpy(theTeleroboticsTitle, title);
     }

  theTeleroboticsWindow = openWindow(x,y,width,height,
                                NORMAL_WINDOW, title,
                                iconicState, theRootWindow,
                                &theTeleroboticsGC,0);


  initEvents(theTeleroboticsWindow, IN_PALETTE);
  XDefineCursor(display,theTeleroboticsWindow,theArrowCursor);

  XFlush(display);


  width =BUTTON_LEVEL1_WIDTH;
  height = BUTTON_LEVEL1_HEIGHT;
  x=DISTANCE;
  y=DISTANCE;


  theExitWindow = openWindow(x,y,width,height,
                                  NORMAL_WINDOW, theTeleroboticsTitle,
                                  NORMAL_STATE, theTeleroboticsWindow,
                                  &theExitGC,1);


  initEvents(theExitWindow, IN_PALETTE);
  associateFont(theExitGC,TEXT1_FONT);

  XDefineCursor(display, theExitWindow,theQuitCursor);

  x +=(BUTTON_LEVEL1_WIDTH + DISTANCE) ;
  theConfigurationWindow = openWindow(x,y,width,height,
                                  NORMAL_WINDOW, theTeleroboticsTitle,
                                  NORMAL_STATE, theTeleroboticsWindow,
                                  &theConfigurationGC,1);
  initEvents(theConfigurationWindow, IN_PALETTE);
  associateFont(theConfigurationGC,TEXT1_FONT);

  XDefineCursor(display,theConfigurationWindow, theButtonCursor);

  x +=(BUTTON_LEVEL1_WIDTH + DISTANCE) ;
  theCallSetUpWindow = openWindow(x,y,width,height,
                                  NORMAL_WINDOW, theTeleroboticsTitle,
                                  NORMAL_STATE, theTeleroboticsWindow,
                                  &theCallSetUpGC,1);
  initEvents(theCallSetUpWindow, IN_PALETTE);
  associateFont(theCallSetUpGC,TEXT1_FONT);

  XDefineCursor(display,theCallSetUpWindow, theButtonCursor);

  x +=(BUTTON_LEVEL1_WIDTH + DISTANCE) ;
  theStartWindow = openWindow(x,y,width,height,
                                  NORMAL_WINDOW, theTeleroboticsTitle,
```

```
                                NORMAL_STATE, theTeleroboticsWindow,
                                &theStartGC,1);
  initEvents(theStartWindow, IN_PALETTE);
  associateFont(theStartGC,TEXT1_FONT);

  XDefineCursor(display,theStartWindow, theButtonCursor);

  x +=(BUTTON_LEVEL1_WIDTH + DISTANCE) ;
  theStopWindow = openWindow(x,y,width,height,
                                NORMAL_WINDOW, theTeleroboticsTitle,
                                NORMAL_STATE, theTeleroboticsWindow,
                                &theStopGC,1);
  initEvents(theStopWindow, IN_PALETTE);
  associateFont(theStopGC,TEXT1_FONT);

  XDefineCursor(display,theStopWindow, theButtonCursor);

  x +=(BUTTON_LEVEL1_WIDTH + DISTANCE) ;
  theHelpWindow = openWindow(x,y,width,height,
                                NORMAL_WINDOW, theTeleroboticsTitle,
                                NORMAL_STATE, theTeleroboticsWindow,
                                &theHelpGC,1);
  initEvents(theHelpWindow, IN_PALETTE);
  associateFont(theHelpGC,TEXT1_FONT);

  XDefineCursor(display,theHelpWindow, theButtonCursor);
  XFlush(display);


} /* function initWindows*/

/*
** processExpose redraw  a given window if it gets an Expose event.
*/

processExpose(event)
XExposeEvent  *event;

{

  if (event->window == theExitWindow)
    {
      XDrawString(display, theExitWindow,
                  theExitGC,
                  40,17, "Exit", strlen("Exit"));
      return;
    }

  if (event->window == theConfigurationWindow)
    {
      XDrawString(display, theConfigurationWindow, theConfigurationGC,
                  7,17,"QOS Config", strlen("QOS Config"));
      return;
    }


  if (event->window == theCallSetUpWindow)
    {
      XDrawString(display, theCallSetUpWindow,
                  theCallSetUpGC,
                  7,17, "Call Set Up", strlen("Call Set Up"));
      return;
    }

  if (event->window == theStartWindow)
    {
      XDrawString(display, theStartWindow,
                  theStartGC,
                  40,17, "Start", strlen("Start"));
      return;
    }


  if (event->window == theStopWindow)
    {
      XDrawString(display, theStopWindow,
                  theStopGC,
                  40,17, "Stop", strlen("Stop"));
      return;
    }

 if (event->window == theHelpWindow)
    {
      XDrawString(display, theHelpWindow,
                  theHelpGC,
                  40,17, "Help", strlen("Help"));
      return;
    }
} /* --function processExpose */


/*
** highlightChoice highlights a palette window
** when the user clicks a mouse button in the window
*/

highlightChoice(window,theName,buttonWidth,buttonHeight)
Window  window;
char theName[];
int buttonWidth,buttonHeight;

{
  XExposeEvent   theExposeEvent;

/*  printf("before fillRectangle \n"); */

  setColorWithName(theExitGC,theName);
  fillRectangle(window, theExitGC,
                0,0, buttonWidth,buttonHeight);

/*  printf("after fillRectangle \n"); */
  theExposeEvent.window = window;
  processExpose(&theExposeEvent);

/*  printf("after processExpose \n"); */
  XFlush(display);


} /* -- function highlightChoice */


/*
** releases all windows
*/

freeWindowsAndGCs()
{
```

```
  XFreeGC(display,theTeleroboticsGC);
  XFreeGC(display, theConfigurationGC);
  XFreeGC(display, theExitGC);
  XFreeGC(display, theCallSetUpGC);
  XFreeGC(display, theStartGC);
  XFreeGC(display, theStopGC);
  XFreeGC(display, theHelpGC);

  XDestroySubwindows(display, theTeleroboticsWindow);
  XDestroyWindow(display, theTeleroboticsWindow);

  XFlush(display);
}/*freeWindowsAndGCs */
```

```
/*
** xor.c
** XOR (rubber-banding) functions
*/

#include <X11/Xlib.h>
#include <X11/Xutil.h>

extern Display *display;

/*
** xorSetUp
** Sets up given window with Xor Graphics Context
*/

xorSetUp(window,theXorGC)
Window window;
GC   *theXorGC;
{
  createGC(window,theXorGC);

  XSetFunction(display,
               *theXorGC,
               GXcopy);

  XFlush(display);
} /* -- function xorSetUP */

/*
** Frees up the Graphics Context storage in the server
*/

xorShutDown(theXorGC)
GC theXorGC;
{
  XFreeGC(display,theXorGC);
} /* -- function xorShutDown */
```

```
/*************************************************************************/
/* Filename : DialogCMSMediaQuality.c                                    */
/* Purpose  : X window dialog for the menu " Media Quality Parameter"    */
/* Author   : Klara Nahrstedt                                            */
/* Update   : 07/03/95                                                   */
/*************************************************************************/

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>
#include <X11/keysymdef.h>

#include "/home/klara/tele.d/include.d/retta.h"
#include "/home/klara/tele.d/include.d/defs.h"
#include "/home/klara/tele.d/include.d/qos.h"

extern Display *display;

/* menu state variable */
extern int menu_state;

char theMMMasterInputDescription[MAX_CHAR];
char theMMSlaveInputDescription[MAX_CHAR];
char theMMMasterOutputDescription[MAX_CHAR];
char theMMSlaveOutputDescription[MAX_CHAR];

char theMMDescription[MAX_CHAR];
char compressionValue[5];
char frameRateValue[4];
char robotRateValue[4];
char robotIntraSpec[4];

extern Cursor theArrowCursor;
extern Cursor theTextCursor;
extern Cursor theQuitCursor;
extern Cursor theButtonCursor;
extern Cursor theBusyCursor;

extern Window theCallQoSWindow;
extern Window theRootWindow;
extern Window theErrorWindow;

extern GC theCallQoSGC;

Window theMQWindow;

Window theInOutMQWindow;
Window theMQ1Window;
Window theMQ2Window;
Window theMQCancelWindow;

/*********************** subwindows of InOutMQWindow ******************/

/* Button Windows */

Window theInOutMQQuitWindow;
Window theInOutMQDoneWindow;
Window theInOutMQCancelWindow;

/* Input Windows */

Window theInOutTextWindow;

/* subwindows in InOutMQWindow quality parameters */

/* video parameters windows */

Window theWidthFrameWindow;
GC     theWidthFrameGC;
Window theListWFWindow;
GC     theListWFGC;

Window theHeightFrameWindow;
GC     theHeightFrameGC;
Window theListHFWindow;
GC     theListHFGC;

Window theBitPerPixelWindow;
GC     theBitPerPixelGC;
Window theListBPPWindow;
GC     theListBPPGC;

Window theCompressionWindow;
GC     theCompressionGC;
Window theListCompressionWindow;
GC     theListCompressionGC;

/* List elements of Compression Choice */

Window theNoneWindow;
GC     theNoneGC;
Window theJPEGWindow;
GC     theJPEGGC;
Window theMPEGWindow;
GC     theMPEGGC;


Window theFrameLossWindow;
GC     theFrameLossGC;
Window theListFLWindow;
GC     theListFLGC;

Window theFrameRateWindow;
GC     theFrameRateGC;
Window theListFRWindow;
GC     theListFRGC;

/******************** Elements of Tune Choice ******************/

Window theStartTuneWindow;
GC     theStartTuneGC;
Window theStopTuneWindow;
GC     theStopTuneGC;


Window theVEEdelayWindow;
GC     theVEEdelayGC;
Window theListVEEWindow;
GC     theListVEEGC;

/* audio parameter windows */

Window theAudioSampleSizeWindow;
GC     theAudioSampleSizeGC;
Window theAudioSampleRateWindow;
GC     theAudioSampleRateGC;
```

```
Window theAudioSampleLossRateWindow;
GC     theAudioSampleLossRateGC;
Window theAEEdelayWindow;
GC     theAEEdelayGC;

/* robotics parameter windows */

Window theRobotSampleSizeWindow;
GC     theRobotSampleSizeGC;
Window theRobotSampleRateWindow;
GC     theRobotSampleRateGC;
Window theRobotSampleLossRateWindow;
GC     theRobotSampleLossRateGC;
Window theREEdelayWindow;
GC     theREEdelayGC;

/******* Robotics Intraframe Specification ***************/

Window theRobotIntraWindow;
GC     theRobotIntraGC;
int intraX,intraY;

Window theRobotNameNWindow;
GC     theRobotNameNGC;
Window theRobotSizeNWindow;
GC     theRobotSizeNGC;
Window theRobotRateNWindow;
GC     theRobotRateNGC;
Window theRobotPrioNWindow;
GC     theRobotPrioNGC;
Window theRobotLossNWindow;
GC     theRobotLossNGC;

Window theRobotNameOWindow;
GC     theRobotNameOGC;
Window theRobotSizeOWindow;
GC     theRobotSizeOGC;
Window theRobotRateOWindow;
GC     theRobotRateOGC;
Window theRobotPrioOWindow;
GC     theRobotPrioOGC;
Window theRobotLossOWindow;
GC     theRobotLossOGC;

Window theRobotNameAWindow;
GC     theRobotNameAGC;
Window theRobotSizeAWindow;
GC     theRobotSizeAGC;
Window theRobotRateAWindow;
GC     theRobotRateAGC;
Window theRobotPrioAWindow;
GC     theRobotPrioAGC;
Window theRobotLossAWindow;
GC     theRobotLossAGC;

Window theRobotNamePWindow;
GC     theRobotNamePGC;
Window theRobotSizePWindow;
GC     theRobotSizePGC;
Window theRobotRatePWindow;
GC     theRobotRatePGC;
Window theRobotPrioPWindow;
GC     theRobotPrioPGC;
Window theRobotLossPWindow;
GC     theRobotLossPGC;


GC theMQGC;
GC theInOutDoneGC;
GC theMQCancelGC;
GC theInOutCancelGC;
GC theInOutQuitGC;
GC theMQ1GC;
GC theMQ2GC;

GC theInOutTextGC;
GC theInOutGC;

/****************************** QoS parameters **************************/

extern APP_QOS  MasterInputParam;
extern APP_QOS  MasterOutputParam;
extern APP_QOS  SlaveInputParam;
extern APP_QOS  SlaveOutputParam;

extern MM_DEVICES MasterInputDevices;
extern MM_DEVICES MasterOutputDevices;
extern MM_DEVICES SlaveInputDevices;
extern MM_DEVICES SlaveOutputDevices;

/* stream description */


/************************* Communication Parameters *****************/
/* hostname */
/********************************************************************/
QoSMediaQuality(x,y,side)
int x,y,side;
{

  int width,height;
  int theChoice;

  width = MAIN_WINDOW_WIDTH - 6*DISTANCE;
  height = MAIN_WINDOW_HEIGHT - 100 - 9*DISTANCE;

  initMQWindows(x,y,width,height);

  displayMQ(theMQCancelWindow,side);

  printf("MediaQuality: <menu_state,side> = <%d,%d> \n",
         menu_state,side);
  theChoice = (-1);

  while (theChoice == -1)
    {
      theChoice = MQEventLoop(side);
    }

  freeMQWindows();


} /* -- function QoSMediaQuality */



initMQWindows(x,y,width,height)
```

```
int x,y,width,height;
{

  Window openWindow();

  int ButtonW =1;
  int NoneButtonW =0;

  theMQWindow = openWindow( x,y,
                            width,height,
                            NORMAL_WINDOW,
                            "QoSMediaQuality",
                            NORMAL_STATE,
                            theCallQoSWindow,
                            &theMQGC,
                            NoneButtonW);
  associateFont(theMQGC,TEXT1_FONT);


   initEvents(theMQWindow,IN_PALETTE);

  XDefineCursor(display,theMQWindow,theArrowCursor);


  x = DISTANCE;
  y = height - 2*DISTANCE;

  theMQCancelWindow = openWindow( x,y,
                            BUTTON_LEVEL2_WIDTH,
                             BUTTON_LEVEL2_HEIGHT,
                            NORMAL_WINDOW,
                            "QoSMediaQuality",
                            NORMAL_STATE,
                            theMQWindow,
                            &theMQCancelGC,
                            ButtonW);
  associateFont(theMQCancelGC,TEXT1_FONT);

  initEvents(theMQCancelWindow,IN_PALETTE);

  XDefineCursor(display,theMQCancelWindow,theQuitCursor);

   x =DISTANCE;
  y =DISTANCE;


  initMQTextWindows(x,y);


 XFlush(display);
}/* -- function initMQWindows */

/*
** initMQTextWindows
*/

initMQTextWindows(x,y)
int x,y;
{
  int NoneButtonW = 0;
  int ButtonW = 1;


  theMQ1Window = openWindow( x,y,
                                  BUTTON_LEVEL3_WIDTH,
                                  BUTTON_LEVEL3_HEIGHT,
                                NORMAL_WINDOW,
                                "DirectionMediaQuality",
                                NORMAL_STATE,
                                theMQWindow,
                                &theMQ1GC,
                                ButtonW);
  associateFont(theMQ1GC,TEXT1_FONT);
  initEvents(theMQ1Window,IN_PALETTE);

  XDefineCursor(display,theMQ1Window,theButtonCursor);

      XDrawString(display,theMQWindow, theMQGC,
                  x + BUTTON_LEVEL3_WIDTH + DISTANCE,
                  y + BUTTON_LEVEL3_HEIGHT,
                  "Quality Parameters for Input Media",
                  strlen("Quality Parameters for Input Media"));

  y += (BUTTON_LEVEL3_HEIGHT + DISTANCE);

  theMQ2Window = openWindow( x,y,
                             BUTTON_LEVEL3_WIDTH,
                             BUTTON_LEVEL3_HEIGHT,
                               NORMAL_WINDOW,
                               "DirectionMediaQuality",
                               NORMAL_STATE,
                               theMQWindow,
                               &theMQ2GC,
                               ButtonW);
  associateFont(theMQ2GC,TEXT1_FONT);
  initEvents(theMQ2Window,IN_PALETTE);

  XDefineCursor(display,theMQ2Window,theButtonCursor);

  XDrawString(display,theMQWindow, theMQGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Quality Parameters for Output Media",
              strlen("Quality Parameters for Output Media"));
  XFlush(display);

}/* -- function initMQTextWindows */


/*
** displayMQ
*/

displayMQ(window,side)
Window window;
int side;

{
  int xt,yt,width,height;
  int y;

  y = textHeight(TEXT1_FONT) + 2;

  if (window == theMQCancelWindow)
    {
      XDrawString(display,window, theMQCancelGC,
                  10,y,
                  "Quit",
```

```
                strlen("Quit"));
    }


  if (window == theMQWindow)
    {
      xt = DISTANCE;
      yt = DISTANCE;

      initMQTextWindows(xt,yt);


    }
  if (window == theMQ1Window)
    {
      XClearWindow(display,theMQWindow);
      freeMQText();
      xt = DISTANCE;
      yt = DISTANCE;

     setColorWithName(theMQGC, "blue");
      XDrawString(display,theMQWindow,
                  theMQGC,
                  DISTANCE, DISTANCE -5,
                  "Quality Parameters for Sending Media",
                  strlen("Quality Parameters for Sending Media"));

      QoSInOutMediaQuality(xt,yt,side,INPUT);

      }

  if (window == theMQ2Window)
    {
      XClearWindow(display,theMQWindow);
      freeMQText();
      xt = DISTANCE;
      yt = DISTANCE;
     setColorWithName(theMQGC, "blue");
      XDrawString(display,theMQWindow,
                  theMQGC,
                  DISTANCE, DISTANCE -5,
                  "Quality Parameters for Receiving Media",
                  strlen("Quality Parameters for Receiving Media"));

      QoSInOutMediaQuality(xt,yt,side,OUTPUT);
    }

  XFlush(display);
} /* -- function displayMQ */

/*
** MQEventLoop
*/

MQEventLoop(side)
int side;
{
  int status = (-1);
  XEvent event;

  XNextEvent(display, &event);

  switch(event.type)
    {
    case ConfigureNotify:
    case Expose:
    case MapNotify:
      displayMQ(event.xany.window,side);
      break;
    case ButtonPress:
       if (event.xbutton.window == theMQCancelWindow)
        {
          highlightChoice(theMQCancelWindow,
                          "blue",
                          BUTTON_LEVEL2_WIDTH,
                          BUTTON_LEVEL2_HEIGHT);
          displayMQ(event.xbutton.window,side);
          status = 0;
        }

      if (event.xbutton.window == theMQ1Window)
        {
          highlightChoice(theMQ1Window,
                          "blue",
                          BUTTON_LEVEL3_WIDTH,
                          BUTTON_LEVEL3_HEIGHT);
          displayMQ(event.xbutton.window,side);
          XClearWindow(display,theMQWindow);
          setColorWithName(theMQGC,"black");
          displayMQ(theMQWindow,side);

        }

      if (event.xbutton.window == theMQ2Window)
        {
          highlightChoice(theMQ2Window,
                          "blue",
                          BUTTON_LEVEL3_WIDTH,
                          BUTTON_LEVEL3_HEIGHT);
          displayMQ(event.xbutton.window,side);
          XClearWindow(display,theMQWindow);
          setColorWithName(theMQGC,"black");
          displayMQ(theMQWindow,side);
        }

      break;
    }
  return(status);
}/* -- function TX1EventLoop */



freeMQWindows()
{

  XFreeGC(display, theMQGC);
  XFreeGC(display,theMQCancelGC);



  XDestroySubwindows(display,theMQWindow);
  XDestroyWindow(display,theMQWindow);

}

freeMQText()
{
```

```
  XFreeGC(display,theMQ1GC);
  XFreeGC(display,theMQ2GC);


  XDestroyWindow(display,theMQ1Window);
  XDestroyWindow(display,theMQ2Window);
}


/*********************************************************************/
/*
** QoSInOutMediaQuality
*/

QoSInOutMediaQuality(x,y,side,inout)
int x,y,side,inout;

{
  int theChoice;
  int width,height;
  MQ_FLAG flags;
  int i;



/********************** init values*********************************/
      for (i=0; i<MAX_CHAR; i++)
        {
          theMMDescription[i] = '\0';
        }
    if (menu_state == CONFIGURE)
    {
      if (side == MASTER && inout == INPUT)
        {
          bzero((char *)(&MasterInputParam),sizeof(APP_QOS));
          setAppQoS(&MasterInputParam,INPUT);
        }
      if (side == MASTER && inout == OUTPUT)
        {
          bzero((char *)(&MasterOutputParam),sizeof(APP_QOS));
          setAppQoS(&MasterOutputParam,OUTPUT);
        }
      if (side == SLAVE && inout == INPUT)
        {
          bzero((char *)(&SlaveInputParam),sizeof(APP_QOS));
        }
      if (side == SLAVE && inout == OUTPUT)
        {
          bzero((char *) (&SlaveOutputParam),sizeof(APP_QOS));
        }

      for (i=0;i<=4;i++)
        {
          frameRateValue[i] = '\0';
          compressionValue[i] = '\0';
          robotRateValue[i] = '\0';
          robotIntraSpec[i] = '\0';
        }

      /*default value for compression */
      strcpy(compressionValue,"NONE");
      strcpy(frameRateValue,"300");
      strcpy(robotRateValue,"50");
      strcpy(robotIntraSpec,"no");
    }
  if (menu_state == CHANGE_SHOW)
    {
      switch(side)
        {
        case MASTER:
          if (inout == INPUT)
            {
              getcharVideoRate(MasterInputParam.stream[VIDEO].medium.app_spec.sample_rate
,
                               frameRateValue);
              getcharRobotRate(MasterInputParam.stream[ROBOT].medium.app_spec.sample_rate
,
                               robotRateValue);
              getcharYesNo(MasterInputParam.stream[ROBOT].intra,
                           robotIntraSpec);
              printf("robotRateValue %s \n",robotRateValue);
              strcpy(theMMDescription,theMMMasterInputDescription);

            }
          if (inout == OUTPUT)
            {
              getcharVideoRate(MasterOutputParam.stream[VIDEO].medium.app_spec.sample_rat
e,
                               frameRateValue);
              printf("MediaQuality: frame rate value = %d \n",
                     MasterOutputParam.stream[VIDEO].medium.app_spec.sample_rate);
              getcharRobotRate(MasterOutputParam.stream[ROBOT].medium.app_spec.sample_rat
e,
                               robotRateValue);
              getcharYesNo(MasterOutputParam.stream[ROBOT].intra,
                           robotIntraSpec);
              strcpy(theMMDescription,theMMMasterOutputDescription);
            }
          break;
        case SLAVE:
          if (inout == INPUT)
            {
              getcharVideoRate(SlaveInputParam.stream[VIDEO].medium.app_spec.sample_rate,
                               frameRateValue);
              printf("MediaQuality: frame rate value = %d \n",
                     SlaveInputParam.stream[VIDEO].medium.app_spec.sample_rate);
              getcharRobotRate(SlaveInputParam.stream[ROBOT].medium.app_spec.sample_rate,
                               robotRateValue);
              getcharYesNo(SlaveInputParam.stream[ROBOT].intra,
                           robotIntraSpec);
              strcpy(theMMDescription,theMMSlaveInputDescription);
            }
          if (inout == OUTPUT)
            {
              getcharVideoRate(SlaveOutputParam.stream[VIDEO].medium.app_spec.sample_rate
,
                               frameRateValue);
              getcharRobotRate(SlaveOutputParam.stream[ROBOT].medium.app_spec.sample_rate
,
                               robotRateValue);
              getcharYesNo(SlaveOutputParam.stream[ROBOT].intra,
                           robotIntraSpec);
              strcpy(theMMDescription,theMMSlaveOutputDescription);
            }
          break;
        }
    }
```

```
  bzero((char *)(&flags),sizeof(MQ_FLAG));
  flags.freeErrorWindow = TRUE;
  width = MAIN_WINDOW_WIDTH -8*DISTANCE;
  height = MAIN_WINDOW_HEIGHT - 100 -13*DISTANCE;


/*****************************************************************/
  initInOutMQ(x,y,width,height,side,inout);

  displayInOutMQ(theInOutMQQuitWindow,side,inout);
  displayInOutMQ(theInOutMQCancelWindow,side,inout);
  displayInOutMQ(theInOutMQDoneWindow,side,inout);

  theChoice= (-1);

  while (theChoice == -1)
    {
      theChoice = InOutMQEventLoop(side,x,y,&flags,inout);

      if (flags.descr  && flags.selectCompression == 0)
        flags.send = 1;
    }

freeInOutMQ();

}/* -- function QoSInputMediaQuality */


initInOutMQ(x,y,width,height,side,inout)
int x,y,width,height,side,inout;
{

  Window openWindow();

  int ButtonW = 1;
  int NoneButtonW = 0;

 theInOutMQWindow = openWindow(x,y,
                               width,height,
                               NORMAL_WINDOW,
                               "QoSInOut",
                               NORMAL_STATE,
                               theMQWindow,
                               &theInOutGC,
                               NoneButtonW);

  associateFont(theInOutGC,TEXT1_FONT);

  initEvents(theInOutMQWindow,IN_PALETTE);

  XDefineCursor(display,theInOutMQWindow,theArrowCursor);

  x = DISTANCE;
  y = height - 2*DISTANCE;

  theInOutMQQuitWindow = openWindow( x,y,
                               BUTTON_LEVEL2_WIDTH,
                                BUTTON_LEVEL2_HEIGHT,
                               NORMAL_WINDOW,
                               "QoSMediaQuality",
                               NORMAL_STATE,
                               theInOutMQWindow,
                               &theInOutQuitGC,
                               ButtonW);

  associateFont(theInOutQuitGC,TEXT1_FONT);

  initEvents(theInOutMQQuitWindow,IN_PALETTE);

  XDefineCursor(display,theInOutMQQuitWindow,theQuitCursor);

  x += (BUTTON_LEVEL2_WIDTH + DISTANCE);

  theInOutMQCancelWindow = openWindow( x,y,
                                BUTTON_LEVEL2_WIDTH,
                                 BUTTON_LEVEL2_HEIGHT,
                                NORMAL_WINDOW,
                                "QoSMediaQuality",
                                NORMAL_STATE,
                                theInOutMQWindow,
                                &theInOutCancelGC,
                                ButtonW);
  associateFont(theInOutCancelGC,TEXT1_FONT);

  initEvents(theInOutMQCancelWindow,IN_PALETTE);

  XDefineCursor(display,theInOutMQCancelWindow,theQuitCursor);

  x += (BUTTON_LEVEL2_WIDTH + DISTANCE);

  theInOutMQDoneWindow = openWindow( x,y,
                                BUTTON_LEVEL2_WIDTH,
                                 BUTTON_LEVEL2_HEIGHT,
                                NORMAL_WINDOW,
                                "QoSMediaQuality",
                                NORMAL_STATE,
                                theInOutMQWindow,
                                &theInOutDoneGC,
                                ButtonW);
  associateFont(theInOutDoneGC,TEXT1_FONT);

  initEvents(theInOutMQDoneWindow,IN_PALETTE);

  XDefineCursor(display,theInOutMQDoneWindow,theButtonCursor);

  x = DISTANCE;
  y = DISTANCE;

  initInOutText(x,y,side,inout);

}/* -- function initInputMQ*/

/*
** initInOutText
*/


initInOutText(x,y,side,inout)
int x,y,side,inout;
{
  int NoneButtonW = 0;
  int ButtonW = 1;

  theInOutTextWindow = openWindow(x + 400,
                                  y - 8,
                                  TEXT_WINDOW_WIDTH,
                                  TEXT_WINDOW_HEIGHT,
                                  NORMAL_WINDOW,
                                  "QoSInOut",
```

```
                                NORMAL_STATE,
                                theInOutMQWindow,
                                &theInOutTextGC,
                                NoneButtonW);
  associateFont(theInOutTextGC,TEXT1_FONT);

  initEvents(theInOutTextWindow,IN_PALETTE);

  XDefineCursor(display,theInOutTextWindow,theTextCursor);

  displayInOutMQ(theInOutTextWindow,side,inout);

  if (inout == INPUT)
    {
      if (menu_state == CONFIGURE)
        {
          XDrawString(display, theInOutMQWindow,
                      theInOutGC,
                      x,y,
                      "Describe the Input Multimedia Stream (e.g. v,a,r):",
           strlen("Describe the Input Multimedia Stream (e.g. v,a,r):"));

        }
      if (menu_state == CHANGE_SHOW)
        {
          XDrawString(display, theInOutMQWindow,
                      theInOutGC,
                      x,y,
                      "Input Multimedia Stream is  (e.g. v,a,r):",
                      strlen("Input Multimedia Stream is (e.g. v,a,r):"));
        }
      }
    if (inout == OUTPUT)
      {
        if (menu_state == CONFIGURE)
          {
            XDrawString(display, theInOutMQWindow,
                        theInOutGC,
                        x,y,
                        "Describe the Output Multimedia Stream (e.g. v,a,r):",
                        strlen("Describe the Output Multimedia Stream (e.g. v,a,r):"));
          }
        if (menu_state == CHANGE_SHOW)
          {
            XDrawString(display, theInOutMQWindow,
                        theInOutGC,
                        x,y,
                        " Output Multimedia Stream is (e.g. v,a,r):",
                        strlen("Output Multimedia Stream is (e.g. v,a,r):"));

          }
      }

  y += DISTANCE;

        XDrawString(display, theInOutMQWindow,
                    theInOutGC,
                    x,y,
                    "(video=v,audio=a,robotics=r)",
                    strlen("(video=v,audio=a,robotics=r)"));
} /* initInOutText */
```

```
initVideoText(x,y,side,inout,xl,yl)
int x,y,side;
int inout;
int *xl,*yl;
{
  int NoneButtonW = 0;
  int ButtonW = 1;
  int th;
  int frame_size;
  int vrate;
  setColorWithName(theInOutGC,"blue");
  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x,y,
              "Video Quality Parameters",
              strlen("Video Quality Parameters"));
  setColorWithName(theInOutGC,"black");

  setParamValue(VIDEO,S_TYPE,NONE,VIDEO,inout,side);

  openTextWindow(&theWidthFrameWindow,&theWidthFrameGC,x,y);
  displayInOutMQ(theWidthFrameWindow,side,inout);

  y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "Width of Frame (in Pixels)",
              strlen("Width of Frame (in Pixels)"));

/* Parameter Height of Frame */

  openTextWindow(&theHeightFrameWindow,&theHeightFrameGC,x,y);
  displayInOutMQ(theHeightFrameWindow,side,inout);

   y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
                 "Height of Frame (in Pixels)",
                 strlen("Height of Frame (in Pixels)"));

/* Bits per Pixel */

  openTextWindow(&theBitPerPixelWindow,&theBitPerPixelGC,x,y);
  displayInOutMQ(theBitPerPixelWindow, side,inout);
/*****************************************************************/
/* Computation of frame (sample) size                            */
/*****************************************************************/
  frame_size = 240*160; /* Width x Height x Bits/pixel */

  setParamValue(VIDEO,S_SIZE,NONE,frame_size,inout,side);
  y +=LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "Bits per Pixel",
              strlen("Bits per Pixel"));

/* Compression */
```

```
  openTextWindow(&theCompressionWindow,&theCompressionGC,x,y);
  displayInOutMQ(theCompressionWindow, side,inout);

  openListWindow(&theListCompressionWindow,&theListCompressionGC,x,y,xl,yl);
  displayInOutMQ(theListCompressionWindow,side,inout);

   y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "Compression ",
              strlen("Compression "));

/* Frame Loss Rate */

  openTextWindow(&theFrameLossWindow,&theFrameLossGC,x,y);
  displayInOutMQ(theFrameLossWindow, side,inout);

  /* Set Default Value in application QoS structure */

  setParamValue(VIDEO,S_LOSS,NONE,1,inout,side);

  y +=LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "Frame Loss Rate (in Frames/min)",
              strlen("Frame Loss Rate (in Frames/min)"));


/* Frame Rate */

  openTextWindow(&theFrameRateWindow,&theFrameRateGC,x,y);
  displayInOutMQ(theFrameRateWindow, side,inout);

  /* Set Defualt Value */
  if (menu_state == CONFIGURE)
    {
      vrate=getintVideoRate(frameRateValue);
      setParamValue(VIDEO,S_RATE,NONE,vrate,inout,side);
    }
  else
    {
      if (side == MASTER && inout==OUTPUT)
        {
          vrate=MasterOutputParam.stream[VIDEO].medium.app_spec.sample_rate;
          setParamValue(VIDEO,S_RATE,NONE,vrate,inout,side);
        }
      if (side == SLAVE && inout == INPUT)
        {
          vrate = SlaveInputParam.stream[VIDEO].medium.app_spec.sample_rate;
          setParamValue(VIDEO,S_RATE,NONE,vrate,inout,side);
        }
    }
  if (strcmp(frameRateValue,"0")!=0)
    {
      setParamValue(VIDEO,QUALITY,NONE,MOTION_VIDEO,inout,side);
    }
  else
    setParamValue(VIDEO,QUALITY,NONE,STILL_IMAGE,inout,side);

  openListWindow(&theListFRWindow,&theListFRGC,x,y,xl,yl);
  displayInOutMQ(theListFRWindow,side,inout);

  y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "Frame Rate (in Frames/minute)",
              strlen("Frame Rate (in Frames/minute)"));

/* End-to-End Delay */

  openTextWindow(&theVEEdelayWindow,&theVEEdelayGC,x,y);
  displayInOutMQ(theVEEdelayWindow, side,inout);

  /* Set Defualt Value */
  setParamValue(VIDEO,S_DELAY,NONE,1000,inout,side);
  y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "End-to-End Delay (in milisec)",
              strlen("End-to-End Delay (in milisec)"));


} /* initVideoText */


openTextWindow(window, gc,x,y)
Window  *window;
GC  *gc;
int x,y;

{
  int NoneButtonW = 0;

  *window = openWindow(x + 250,
                       y + 10,
                       TEXT_WINDOW_WIDTH,
                       TEXT_WINDOW_HEIGHT,
                       NORMAL_WINDOW,
                       "QoSInOut",
                       NORMAL_STATE,
                       theInOutMQWindow,
                       gc,
                       NoneButtonW);
  associateFont(*gc,TEXT1_FONT);
  initEvents(*window,IN_PALETTE);
  XDefineCursor(display,*window,theTextCursor);

}


openListWindow(window, gc,x,y,xl,yl)
Window  *window;
GC  *gc;
int x,y;
int *xl,*yl;
{
  int ButtonW = 1;

  *window = openWindow(x + 250 + TEXT_WINDOW_WIDTH + 20,
                       y + 5,
```

```
                    BUTTON_LEVEL2_WIDTH,
                    TEXT_WINDOW_HEIGHT,
                    NORMAL_WINDOW,
                    "QoSListParam",
                    NORMAL_STATE,
                    theInOutMQWindow,
                    gc,
                    ButtonW);
  associateFont(*gc,TEXT1_FONT);
  initEvents(*window,IN_PALETTE);
  XDefineCursor(display,*window,theButtonCursor);
/*
  XDrawString(display,*window,*gc,
              5,textHeight(TEXT1_FONT),
              "List",
              strlen("List"));
*/
  *xl = x + 250 + TEXT_WINDOW_WIDTH + 20;
  *yl = y +5;
}


/*
** initAudioText
*/

initAudioText(x,y,side,inout)
int x,y;
int side;
int inout;
{
  setParamValue(AUDIO,S_TYPE,NONE,AUDIO,inout,side);

  setColorWithName(theInOutGC,"blue");
  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x,y,
              "Audio Quality Parameters",
              strlen("Audio Quality Parameters"));

  setColorWithName(theInOutGC,"black");

/* Sample Size */

  openTextWindow(&theAudioSampleSizeWindow,&theAudioSampleSizeGC,x,y);
  displayInOutMQ(theAudioSampleSizeWindow, side,inout);
  setParamValue(AUDIO,S_SIZE,NONE,400,inout,side);

  y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "Sample Size (in Bytes)",
              strlen("Sample Size (in Bytes)"));

  openTextWindow(&theAudioSampleRateWindow, &theAudioSampleRateGC,x,y);
  displayInOutMQ(theAudioSampleRateWindow, side,inout);
  setParamValue(AUDIO,S_RATE,NONE,160,inout,side);

 y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "Sample Rate (in Samples/s)",
              strlen("Sample Rate (in Samples/s)"));

  openTextWindow(&theAudioSampleLossRateWindow, &theAudioSampleLossRateGC,x,y);
  displayInOutMQ(theAudioSampleLossRateWindow, side,inout);
  setParamValue(AUDIO,S_LOSS,NONE,1,inout,side);

  y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "Sample Loss Rate (in Samples/min)",
              strlen("Sample Loss Rate (in Samples/min)"));
  openTextWindow(&theAEEdelayWindow, &theAEEdelayGC,x,y);
  displayInOutMQ(theAEEdelayWindow, side,inout);
  setParamValue(AUDIO,S_DELAY,NONE,100,inout,side);

  y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "End-to-End Delay (in milisec)",
              strlen("End-to-End Delay (in milisec)"));
}/* initAudioText*/

/*
** initRoboticsText
*/

initRoboticsText(x,y,side,inout)
int x,y;
int side;
int inout;
{

  int rate;

  setParamValue(ROBOT,S_TYPE,NONE,ROBOT,inout,side);

  setColorWithName(theInOutGC,"blue");
  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x,y,
              "Robotics Quality Parameters",
              strlen("Robotics Quality Parameters"));
  setColorWithName(theInOutGC,"black");

  openTextWindow(&theRobotSampleSizeWindow, &theRobotSampleSizeGC,x,y);
  displayInOutMQ(theRobotSampleSizeWindow, side,inout);
  setParamValue(ROBOT,S_SIZE,NONE,64,inout,side);

  y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "Sample Size (in Bytes)",
              strlen("Sample Size (in Bytes)"));

  openTextWindow(&theRobotSampleRateWindow, &theRobotSampleRateGC,x,y);
  displayInOutMQ(theRobotSampleRateWindow, side,inout);
```

```
  rate = getintRobotRate(robotRateValue);

  if (menu_state == CONFIGURE)
    setParamValue(ROBOT,S_RATE,NONE,50,inout,side);

/****************** quality of the medium ******************************/
/* Note: The quality is determined in dependence of the rate            */
/***********************************************************************/

  setParamValue(ROBOT,S_RATE,NONE,rate,inout,side);
  if (rate >=50 & rate <=100)
    setParamValue(ROBOT,QUALITY,NONE,LOW,inout,side);
  if (rate > 100 && rate <=300)
    setParamValue(ROBOT,QUALITY,NONE,MEDIUM,inout,side);
  if (rate >300 && rate <= 500)
    setParamValue(ROBOT,QUALITY,NONE,HIGH,inout,side);

    y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "Sample Rate (in Samples/s)",
              strlen("Sample Rate (in Samples/s)"));

  openTextWindow(&theRobotSampleLossRateWindow, &theRobotSampleLossRateGC,x,y);
  displayInOutMQ(theRobotSampleLossRateWindow, side,inout);
  setParamValue(ROBOT,S_LOSS,NONE,2,inout,side);

  y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "Sample Loss Rate (in Samples/min)",
              strlen("Sample Loss Rate (in Samples/min)"));

  openTextWindow(&theREEdelayWindow, &theREEdelayGC,x,y);
  displayInOutMQ(theREEdelayWindow, side,inout);
  setParamValue(ROBOT,S_DELAY,NONE,20,inout,side);

  y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "End-to-End Delay (in milisec)",
              strlen("End-to-End Delay (in milisec)"));

  openTextWindow(&theRobotIntraWindow,&theRobotIntraGC,x,y);
  displayInOutMQ(theRobotIntraWindow,side,inout);
  setParamValue(ROBOT,S_INTRA,NONE,FALSE,inout,side);

  y += LINE_DISTANCE;

  XDrawString(display, theInOutMQWindow,
              theInOutGC,
              x + DISTANCE,y,
              "Intraframe Specification (yes/no)",
              strlen("Intraframe Specification (yes/no)"));


}/* initRoboticsText */
```

```
initRobotIntraWindows(x,y,side,inout)
int x,y;
int side;
int inout;
{

  int X,Y;
  int NX, OX,AX,PX;
  int rate;
  setParamValue(ROBOT,S_INTRA,NONE,TRUE,inout,side);

  X = x + 250 + TEXT_WINDOW_WIDTH + 20;
  Y = y;

  setColorWithName(theInOutGC,"blue");
  XDrawString(display,theInOutMQWindow,
              theInOutGC,
              X + DISTANCE,Y,
              "Robotics Intraframe Parameter Specification",
              strlen("Robotics Intraframe Parameter Specification"));

  setColorWithName(theInOutGC,"black");
  NX = X-150;
  openTextWindow(&theRobotNameNWindow,&theRobotNameNGC,NX,Y);
  displayInOutMQ(theRobotNameNWindow,side,inout);
  setParamValue(ROBOT,C_NAME,N_COMP,N_COMP,inout,side);

  OX = X-150 + TEXT_WINDOW_WIDTH +10;

  openTextWindow(&theRobotNameOWindow,&theRobotNameOGC,OX,Y);
  displayInOutMQ(theRobotNameOWindow,side,inout);
  setParamValue(ROBOT,C_NAME,O_COMP,O_COMP,inout,side);

  AX = OX + TEXT_WINDOW_WIDTH + 10;
  openTextWindow(&theRobotNameAWindow,&theRobotNameAGC,AX,Y);
  displayInOutMQ(theRobotNameAWindow,side,inout);
  setParamValue(ROBOT,C_NAME,A_COMP,A_COMP,inout,side);

  PX = AX + TEXT_WINDOW_WIDTH + 10;

  openTextWindow(&theRobotNamePWindow,&theRobotNamePGC,PX,Y);
  displayInOutMQ(theRobotNamePWindow,side,inout);
  setParamValue(ROBOT,C_NAME,P_COMP,P_COMP,inout,side);

  Y +=LINE_DISTANCE;

  XDrawString(display,theInOutMQWindow,
              theInOutGC,
              X + DISTANCE,Y,
              "Name",
              strlen("Name"));

/********************** Size Spec *********************************/

  openTextWindow(&theRobotSizeNWindow,&theRobotSizeNGC,NX,Y);
  displayInOutMQ(theRobotSizeNWindow,side,inout);
  setParamValue(ROBOT,C_SIZE,N_COMP,12,inout,side);

  openTextWindow(&theRobotSizeOWindow,&theRobotSizeOGC,OX,Y);
  displayInOutMQ(theRobotSizeOWindow,side,inout);
  setParamValue(ROBOT,C_SIZE,O_COMP,12,inout,side);

  openTextWindow(&theRobotSizeAWindow,&theRobotSizeAGC,AX,Y);
  displayInOutMQ(theRobotSizeAWindow,side,inout);
```

```
  setParamValue(ROBOT,C_SIZE,A_COMP,12,inout,side);

  openTextWindow(&theRobotSizePWindow,&theRobotSizePGC,PX,Y);
  displayInOutMQ(theRobotSizePWindow,side,inout);
  setParamValue(ROBOT,C_SIZE,P_COMP,12,inout,side);

  Y +=LINE_DISTANCE;

  XDrawString(display,theInOutMQWindow,
              theInOutGC,
              X + DISTANCE,Y,
              "Size",
              strlen("Size"));

/****************** Rate Spec **********************************/

  openTextWindow(&theRobotRateNWindow,&theRobotRateNGC,NX,Y);
  displayInOutMQ(theRobotRateNWindow,side,inout);
  rate = getintRobotRate(robotRateValue);
  setParamValue(ROBOT,C_RATE,N_COMP,rate,inout,side);

  openTextWindow(&theRobotRateOWindow,&theRobotRateOGC,OX,Y);
  displayInOutMQ(theRobotRateOWindow,side,inout);
  setParamValue(ROBOT,C_RATE,O_COMP,rate,inout,side);

  openTextWindow(&theRobotRateAWindow,&theRobotRateAGC,AX,Y);
  displayInOutMQ(theRobotRateAWindow,side,inout);
  setParamValue(ROBOT,C_RATE,A_COMP,rate,inout,side);

  openTextWindow(&theRobotRatePWindow,&theRobotRatePGC,PX,Y);
  displayInOutMQ(theRobotRatePWindow,side,inout);
  setParamValue(ROBOT,C_RATE,P_COMP,rate,inout,side);


  Y +=LINE_DISTANCE;

  XDrawString(display,theInOutMQWindow,
              theInOutGC,
              X +DISTANCE,Y,
              "Rate",
              strlen("Rate"));

/****************** Importance  **********************************/

  openTextWindow(&theRobotPrioNWindow,&theRobotPrioNGC,NX,Y);
  displayInOutMQ(theRobotPrioNWindow,side,inout);
  setParamValue(ROBOT,C_PRIO,N_COMP,MEDIUM_IMPORTANCE,inout,side);

  openTextWindow(&theRobotPrioOWindow,&theRobotPrioOGC,OX,Y);
  displayInOutMQ(theRobotPrioOWindow,side,inout);
  setParamValue(ROBOT,C_PRIO,O_COMP,MEDIUM_IMPORTANCE,inout,side);

  openTextWindow(&theRobotPrioAWindow,&theRobotPrioAGC,AX,Y);
  displayInOutMQ(theRobotPrioAWindow,side,inout);
  setParamValue(ROBOT,C_PRIO,A_COMP,MEDIUM_IMPORTANCE,inout,side);

  openTextWindow(&theRobotPrioPWindow,&theRobotPrioPGC,PX,Y);
  displayInOutMQ(theRobotPrioPWindow,side,inout);
  setParamValue(ROBOT,C_PRIO,P_COMP,HIGH_IMPORTANCE,inout,side);

  Y +=LINE_DISTANCE;

  XDrawString(display,theInOutMQWindow,
              theInOutGC,
              X + DISTANCE,Y,
              "Importance",
              strlen("Importance"));

/****************** Loss Rate **********************************/


  openTextWindow(&theRobotLossNWindow,&theRobotLossNGC,NX,Y);
  displayInOutMQ(theRobotLossNWindow,side,inout);
  setParamValue(ROBOT,C_LOSS,N_COMP,2,inout,side);


  openTextWindow(&theRobotLossOWindow,&theRobotLossOGC,OX,Y);
  displayInOutMQ(theRobotLossOWindow,side,inout);
  setParamValue(ROBOT,C_LOSS,O_COMP,2,inout,side);


  openTextWindow(&theRobotLossAWindow,&theRobotLossAGC,AX,Y);
  displayInOutMQ(theRobotLossAWindow,side,inout);
  setParamValue(ROBOT,C_LOSS,A_COMP,2,inout,side);


  openTextWindow(&theRobotLossPWindow,&theRobotLossPGC,PX,Y);
  displayInOutMQ(theRobotLossPWindow,side,inout);
  setParamValue(ROBOT,C_LOSS,P_COMP,0,inout,side);

  Y +=LINE_DISTANCE;

  XDrawString(display,theInOutMQWindow,
              theInOutGC,
              X + DISTANCE,Y,
              "Loss Rate",
              strlen("Loss Rate"));

}/* initRobotIntraWindows*/

displayInOutMQ(window,side,inout)
Window window;
int side;
int inout;
{
  int xt,yt,width,height;
  int y;

  y = textHeight(TEXT1_FONT);

  if (window == theListCompressionWindow)
    {
      XDrawString(display,window, theListCompressionGC,
                  5,y,
                  "List",
                  strlen("List"));
    }

  if (window == theNoneWindow)
    {
      XDrawString( display, window, theNoneGC,
                  5,y,
                  "NONE",
                  strlen("NONE"));

    }

 if (window == theJPEGWindow)
```

```
    {
      XDrawString( display, window, theJPEGGC,
                   5,y,
                   "JPEG",
                   strlen("JPEG"));
    }

  if (window == theMPEGWindow)
    {
      XDrawString( display, window, theMPEGGC,
                   5,y,
                   "MPEG",
                   strlen("MPEG"));
    }
  if (window == theInOutMQQuitWindow)
    {
      XDrawString(display,window, theInOutQuitGC,
                   5,y,
                   "Quit",
                   strlen("Quit"));
    }
  if (window == theInOutMQCancelWindow)
    {
      XDrawString(display,window, theInOutCancelGC,
                   5,y,
                   "Cancel",
                   strlen("Cancel"));
    }

  if (window == theInOutMQDoneWindow)
    {
      XDrawString(display,window, theInOutDoneGC,
                   10,y,
                   "Done",
                   strlen("Done"));
    }

  if (window == theInOutTextWindow )
    {
      XDrawString(display,window, theInOutTextGC,
                   10,y,
                   theMMDescription,
                   strlen(theMMDescription));
    }

/*  Video Windows Deafulat values */

  if (window == theWidthFrameWindow)
    {

      XDrawString(display,window, theWidthFrameGC,
                   10,y,
                   "240",
                   strlen("240"));
    }

  if (window == theHeightFrameWindow)
    {

      XDrawString(display,window, theHeightFrameGC,
                   10,y,
                   "160",
                   strlen("160"));
    }

 if (window == theBitPerPixelWindow)
   {

     XDrawString(display,window, theBitPerPixelGC,
                   10,y,
                   "8",
                   strlen("8"));
   }



 if (window == theCompressionWindow)
   {
     if (strcmp(compressionValue,"NONE") == 0)
     XDrawString(display,window, theCompressionGC,
                   10,y,
                   "NONE",
                   strlen("NONE"));
     if (strcmp(compressionValue,"JPEG") == 0)
     XDrawString(display,window, theCompressionGC,
                   10,y,
                   "JPEG",
                   strlen("JPEG"));
     if (strcmp(compressionValue,"MPEG") == 0)
       XDrawString(display,window, theCompressionGC,
                     10,y,
                     "MPEG",
                     strlen("MPEG"));
   }
 if (window == theFrameLossWindow)
   {

     XDrawString(display,window, theFrameLossGC,
                   10,y,
                   "1",
                   strlen("1"));
   }


 if (window == theFrameRateWindow)
   {
     XDrawString(display,window, theFrameRateGC,
                   10,y,
                   frameRateValue,
                   strlen(frameRateValue));
   }

 if (window == theListFRWindow)
   {
     XDrawString(display,window,theListFRGC,
                   5,y,
                   "Tune",
                   strlen("Tune"));
   }

 if (window == theStartTuneWindow)
   {
     XDrawString(display,window,theListFRGC,
                   5,y,
                   "Start Tune",
                   strlen("Start Tune"));
   }
```

```
  if (window == theStopTuneWindow)
    {
      XDrawString(display,window,theListFRGC,
                  5,y,
                  "Stop Tune",
                  strlen("Stop Tune"));
    }

  if (window == theVEEdelayWindow)
    {
      XDrawString(display,window, theVEEdelayGC,
                  10,y,
                  "1000",
                  strlen("1000"));
    }

/* Audio Windows Default values */

  if (window == theAudioSampleSizeWindow)
    {

      XDrawString(display,window, theAudioSampleSizeGC,
                  10,y,
                  "400",
                  strlen("400"));
    }


  if (window == theAudioSampleRateWindow)
    {

      XDrawString(display,window, theAudioSampleRateGC,
                  10,y,
                  "160",
                  strlen("160"));
    }

  if (window == theAudioSampleLossRateWindow)
    {

      XDrawString(display,window, theAudioSampleLossRateGC,
                  10,y,
                  "1",
                  strlen("1"));
    }


  if (window == theAEEdelayWindow)
    {

      XDrawString(display,window, theAEEdelayGC,
                  10,y,
                  "100",
                  strlen("100"));
    }

/* Robotics Windows Default values */


  if (window == theRobotSampleSizeWindow)
    {

      XDrawString(display,window, theRobotSampleSizeGC,
                  10,y,
                  "64",
                  strlen("64"));
    }


  if (window == theRobotSampleRateWindow)
    {

      XDrawString(display,window, theRobotSampleRateGC,
                  10,y,
                  robotRateValue,
                  strlen(robotRateValue));
    }


  if (window == theRobotSampleLossRateWindow)
    {

      XDrawString(display,window, theRobotSampleLossRateGC,
                  10,y,
                  "2",
                  strlen("2"));
    }


  if (window == theREEdelayWindow)
    {

      XDrawString(display,window, theREEdelayGC,
                  10,y,
                  "20",
                  strlen("20"));
    }

  if (window == theRobotIntraWindow)
    {
      XDrawString(display,window, theRobotIntraGC,
                  10,y,
                  robotIntraSpec,
                  strlen(robotIntraSpec));
    }
  if (window == theRobotNameNWindow)
    {
      XDrawString(display,window, theRobotNameNGC,
                  10,y,
                  "N",
                  strlen("N"));
    }
  if (window == theRobotNameOWindow)
    {
      XDrawString(display,window, theRobotNameOGC,
                  10,y,
                  "O",
                  strlen("O"));
    }
  if (window == theRobotNameAWindow)
    {
      XDrawString(display,window, theRobotNameAGC,
                  10,y,
                  "A",
                  strlen("A"));
    }
  if (window == theRobotNamePWindow)
```

```
    {
      XDrawString(display,window, theRobotNamePGC,
                  10,y,
                  "P",
                  strlen("P"));
    }

  if (window == theRobotSizeNWindow)
    {
      XDrawString(display,window, theRobotSizeNGC,
                  10,y,
                  "12",
                  strlen("12"));
    }
  if (window == theRobotSizeOWindow)
    {
      XDrawString(display,window, theRobotSizeOGC,
                  10,y,
                  "12",
                  strlen("12"));
    }
  if (window == theRobotSizeAWindow)
    {
      XDrawString(display,window, theRobotSizeAGC,
                  10,y,
                  "12",
                  strlen("12"));
    }
  if (window == theRobotSizePWindow)
    {
      XDrawString(display,window, theRobotSizePGC,
                  10,y,
                  "12",
                  strlen("12"));
    }
  if (window == theRobotRateNWindow)
    {
      XDrawString(display,window, theRobotRateNGC,
                  10,y,
                  robotRateValue,
                  strlen(robotRateValue));
    }
  if (window == theRobotRateOWindow)
    {
      XDrawString(display,window, theRobotRateOGC,
                  10,y,
                  robotRateValue,
                  strlen(robotRateValue));
    }
  if (window == theRobotRateAWindow)
    {
      XDrawString(display,window, theRobotRateAGC,
                  10,y,
                  robotRateValue,
                  strlen(robotRateValue));
    }
  if (window == theRobotRatePWindow)
    {
      XDrawString(display,window, theRobotRatePGC,
                  10,y,
                  robotRateValue,
                  strlen(robotRateValue));
    }
  if (window == theRobotPrioNWindow)
    {
      XDrawString(display,window, theRobotPrioNGC,
                  10,y,
                  "2",
                  strlen("2"));
    }
  if (window == theRobotPrioOWindow)
    {
      XDrawString(display,window, theRobotPrioOGC,
                  10,y,
                  "2",
                  strlen("2"));
    }
  if (window == theRobotPrioAWindow)
    {
      XDrawString(display,window, theRobotPrioNGC,
                  10,y,
                  "2",
                  strlen("2"));
    }
  if (window == theRobotPrioPWindow)
    {
      XDrawString(display,window, theRobotPrioNGC,
                  10,y,
                  "1",
                  strlen("1"));
    }
  if (window == theRobotLossNWindow)
    {
      XDrawString(display,window, theRobotLossNGC,
                  10,y,
                  "2",
                  strlen("2"));
    }
  if (window == theRobotLossOWindow)
    {
      XDrawString(display,window, theRobotLossOGC,
                  10,y,
                  "2",
                  strlen("2"));
    }
  if (window == theRobotLossAWindow)
    {
      XDrawString(display,window, theRobotLossAGC,
                  10,y,
                  "2",
                  strlen("2"));
    }
  if (window == theRobotLossPWindow)
    {
      XDrawString(display,window, theRobotLossPGC,
                  10,y,
                  "0",
                  strlen("0"));
    }
} /* -- function displayInOuMQ */

InOutMQEventLoop(side,x,y,flag,inout)
int side;
int x,y;
MQ_FLAG *flag; /* flag if MMdescription was given */
int inout;
{
  int status = (-1);
```

```
  XEvent event;
  int i;
  int xl,yl;
  int vrate,rrate;

  XNextEvent(display, &event);

  switch(event.type)
    {
    case ConfigureNotify:
    case Expose:
    case MapNotify:
      displayInOutMQ(event.xany.window,side,inout);
      break;
    case ButtonPress:
      if (event.xbutton.window == theInOutMQQuitWindow)
        {
          highlightChoice(theInOutMQQuitWindow,
                          "blue",
                          BUTTON_LEVEL2_WIDTH,
                          BUTTON_LEVEL2_HEIGHT);
          switch(side)
            {
            case MASTER:
              if (inout == INPUT)
                {
                  strcpy(theMMMasterInputDescription,theMMDescription);
                }
              if (inout == OUTPUT)
                {
                  strcpy(theMMMasterOutputDescription,theMMDescription);
                }
              break;
            case SLAVE:
              if (inout == INPUT)
                strcpy(theMMSlaveInputDescription,theMMDescription);
              if (inout == OUTPUT)
                strcpy(theMMSlaveOutputDescription,theMMDescription);
              break;
            }
          status = 0;
          if (flag->video)
            freeVideoText();
          if (flag->audio)
            freeAudioText();
          if (flag->robot)
            freeRobotText(flag);

          if (flag->selectCompression ==1 &&
              flag->freeListCompressionDone == 0)
            freeListCompressionWindows();


          if (flag->selectTune ==1 &&
              flag->freeListTuneDone == 0)
            freeListTuneWindows();

/* set flags to init values after quit */
          flag->selectCompression = 0;
          flag->descr = 0;
          flag->video = OTHER;
          flag->audio = OTHER;
          flag->robot = OTHER;
          flag->selectTune = 0;
          flag->changeRobotRate = FALSE;
          flag->initRobotIntra = FALSE;
          flag->freeListTuneDone = 0;
          flag->freeListCompressionDone = 0;
          if (flag->freeErrorWindow == FALSE)
            {
              freeError();
              flag->freeErrorWindow = TRUE;
            }

          XClearWindow(display,theInOutMQWindow);
          displayInOutMQ(event.xbutton.window,side,inout);

        }
      if (event.xbutton.window == theErrorWindow)
        {
          if (flag->selectCompression)
            {
              strcpy(compressionValue,"NONE");
              XClearWindow(display,theCompressionWindow);
              displayInOutMQ(theCompressionWindow,side,inout);
            }
          if (flag->freeErrorWindow == FALSE)
            {
              freeError();
              flag->freeErrorWindow=TRUE;
            }
        }
      if (event.xbutton.window == theInOutMQCancelWindow)
        {

          highlightChoice(theInOutMQCancelWindow,
                          "grey",
                          BUTTON_LEVEL2_WIDTH,
                          BUTTON_LEVEL2_HEIGHT);

          if (flag->freeErrorWindow == FALSE)
            {
              freeError();
              flag->freeErrorWindow = TRUE;
            }
          if ((strcmp(compressionValue,"JPEG")==0) ||
              (strcmp(compressionValue,"MPEG") == 0))
            {

              strcpy(compressionValue,"NONE");
              XClearWindow(display,theListCompressionWindow);
              displayInOutMQ(theListCompressionWindow,side,inout);
              XClearWindow(display,theCompressionWindow);
              displayInOutMQ(theCompressionWindow,side,inout);
              setCompressionValue(side,inout);
              if (flag->freeListCompressionDone == 0)
                freeListCompressionWindows();
              flag->selectCompression =0;
              flag->freeListCompressionDone =1;
            }
          if (strcmp(robotIntraSpec,"yes")==0)
            {
              strcpy(robotIntraSpec,"no");
              setParamValue(ROBOT,S_INTRA,NONE,FALSE,inout,side);
              XClearWindow(display,theRobotIntraWindow);
              displayInOutMQ(theRobotIntraWindow,side,inout);
            }
          if (strcmp(robotRateValue,"50")!=0)
```

```
        {
          strcpy(robotRateValue,"50");
          setParamValue(ROBOT,S_RATE,NONE,50,inout,side);

          XClearWindow(display,theRobotSampleRateWindow);
          displayInOutMQ(theRobotSampleRateWindow,side,inout);
          flag->changeRobotRate = FALSE;
        }
      if (flag->selectTune)
        {
          strcpy(frameRateValue,"300");
          setParamValue(VIDEO,S_RATE,NONE,300,inout,side);
          flag->selectTune =0;
          XClearWindow(display,theFrameRateWindow);
          displayInOutMQ(theFrameRateWindow,side,inout);

          if (flag->freeListTuneDone == 0)
            {
              freeListTuneWindows();
              flag->freeListTuneDone = 0;
            }
        }

      XClearWindow(display,theInOutMQCancelWindow);
      displayInOutMQ(event.xbutton.window,side,inout);

    }
   if (event.xbutton.window == theInOutMQDoneWindow)
     {
      highlightChoice(theInOutMQDoneWindow,
                      "grey",
                      BUTTON_LEVEL2_WIDTH,
                      BUTTON_LEVEL2_HEIGHT);

      if (flag->selectCompression)
        {
          printf("select compression is %d",flag->selectCompression);
          strcpy(compressionValue,"NONE");
          XClearWindow(display,theCompressionWindow);
          displayInOutMQ(theCompressionWindow,side,inout);
          freeListCompressionWindows();
          flag->selectCompression = 0;
          flag->freeListCompressionDone = 1;
          XClearWindow(display,theListCompressionWindow);
          displayInOutMQ(theListCompressionWindow,side,inout);
        }
      if (flag->changeRobotRate)
        {
          rrate=getintRobotRate(robotRateValue);
          printf("robot rate %d \n",rrate);
/*******************check the robot rate ****************************/
          if (rrate % 10 == 0)
            {
              setParamValue(ROBOT,S_RATE,NONE,rrate,inout,side);
              if (strcmp(robotIntraSpec,"yes") == 0)
                {
                  XClearWindow(display,theRobotRateNWindow);
                  displayInOutMQ(theRobotRateNWindow,side,inout);
                  setParamValue(ROBOT,C_RATE,N_COMP,rrate,inout,side);
                  XClearWindow(display,theRobotRateOWindow);
                  displayInOutMQ(theRobotRateOWindow,side,inout);
                  setParamValue(ROBOT,C_RATE,O_COMP,rrate,inout,side);
                  XClearWindow(display,theRobotRateAWindow);
                  displayInOutMQ(theRobotRateAWindow,side,inout);
                  setParamValue(ROBOT,C_RATE,A_COMP,rrate,inout,side);
                  XClearWindow(display,theRobotRatePWindow);
                  displayInOutMQ(theRobotRatePWindow,side,inout);
                  setParamValue(ROBOT,C_RATE,P_COMP,rrate,inout,side);
                }
            }
          else
            {
              if (rrate >50 && rrate <500 && flag->freeErrorWindow)
                {
                  ShowError(BAD_VALUE,ROBOT,S_RATE);
                  flag->freeErrorWindow = FALSE;
                }
            }
          if (rrate < 10 && flag->freeErrorWindow)
            {
              ShowError(BAD_MINBOUND,ROBOT,S_RATE);
              flag->freeErrorWindow = FALSE;
            }
          else
            {
              setParamValue(ROBOT,S_RATE,NONE,rrate,inout,side);
            }

          if (rrate > 500 && flag->freeErrorWindow)
            {
              ShowError(BAD_MAXBOUND,ROBOT,S_RATE);
              flag->freeErrorWindow = FALSE;
            }
/*********************************************************************/
          flag->changeRobotRate=FALSE;
          XClearWindow(display,theRobotSampleRateWindow);
          displayInOutMQ(theRobotSampleRateWindow,side,inout);
        }
      if (flag->selectTune)
        {
/***************** check video rate **********************************/

          vrate=getintVideoRate(frameRateValue);
          setParamValue(VIDEO,S_RATE,NONE,vrate,inout,side);
          freeListTuneWindows();
          flag->selectTune =0;
          flag->freeListTuneDone =1;
          XClearWindow(display,theListFRWindow);
          displayInOutMQ(theListFRWindow,side,inout);
        }
      else
        {
          vrate=getintVideoRate(frameRateValue);
          setParamValue(VIDEO,S_RATE,NONE,vrate,inout,side);
        }
/*********************************************************************/
/* Intraframe Specification starts if the user choose yes            */
/*********************************************************************/

      if (strcmp(robotIntraSpec,"no") == 0)
        setParamValue(ROBOT,S_INTRA,NONE,FALSE,inout,side);

      if (strcmp(robotIntraSpec,"yes") == 0)
        {
          initRobotIntraWindows(intraX,intraY,side,inout);
          setParamValue(ROBOT,S_INTRA,NONE,TRUE,inout,side);
          flag->initRobotIntra = TRUE;
        }
```

```
if (flag->descr == 0)
  {
    flag->descr=1;
    XClearWindow(display,theInOutMQWindow);
    freeMMDescr(side);
    XClearWindow(display,event.xbutton.window);
    displayInOutMQ(event.xbutton.window,side,inout);
    for (i=0;i<10; i++)
      {
        if (theMMDescription[i] == 'v')
          {
            /****** chek support of video device ****/
            if (inout == INPUT && side == MASTER)
              {
                if (MasterInputDevices.dev_support[VIDEO].support == TRUE)
                  {
                    flag->video =1;
                    initVideoText(x,y,side,inout,&xl,&yl);
                    y += 8*LINE_DISTANCE;
                  }
                else
                  {
                    flag->video = FALSE;
                    if (flag->freeErrorWindow)
                      {
                        ShowError(NOT_SUPPORTED,VIDEO,S_TYPE);
                        flag->freeErrorWindow = FALSE;
                      }
                  }
              }
            if (inout == OUTPUT && side == MASTER)
              {
                if (MasterOutputDevices.dev_support[VIDEO].support == TRUE)
                  {
                    flag->video =1;
                    initVideoText(x,y,side,inout,&xl,&yl);
                    y += 8*LINE_DISTANCE;
                  }
                else
                  {
                    flag->video =FALSE;
                    if (flag->freeErrorWindow)
                      {
                        ShowError(NOT_SUPPORTED,VIDEO,S_TYPE);
                        flag->freeErrorWindow = FALSE;
                      }
                  }
              }
            /* in case of slave, the operator give the parameters,
               but the parameters are checked against the devices
               at the slave side */

            if(side == SLAVE)
              {
                flag->video = TRUE;
                initVideoText(x,y,side,inout,&xl,&yl);
                y += 8*LINE_DISTANCE;
              }
          }
        if (theMMDescription[i] == 'a')
          {
            /***** map audio support against audio device ****/
            if (inout == INPUT && side == MASTER)
              {
                if (MasterInputDevices.dev_support[AUDIO].support == TRUE)
                  {
                    flag->audio = 1;
                    initAudioText(x,y,side,inout);
                    y +=5*LINE_DISTANCE;
                  }
                else
                  {
                    flag->audio = FALSE;
                    if (flag->freeErrorWindow)
                      {
                        ShowError(NOT_SUPPORTED,AUDIO,NONE);
                        flag->freeErrorWindow = FALSE;
                      }
                  }
              }
            if (inout == OUTPUT && side == MASTER)
              {
                if (MasterOutputDevices.dev_support[AUDIO].support == TRUE)
                  {
                    flag->audio = TRUE;
                    initAudioText(x,y,side,inout);
                    y +=5*LINE_DISTANCE;
                  }
                else
                  {
                    flag->audio = FALSE;
                    if (flag->freeErrorWindow)
                      {
                        ShowError(NOT_SUPPORTED,AUDIO,NONE);
                        flag->freeErrorWindow = FALSE;
                      }
                  }
              }
            /***** Slave currently also doesn't support audio*/
            flag->audio = FALSE;
            if (flag->freeErrorWindow)
              {
                ShowError(NOT_SUPPORTED,AUDIO,NONE);
                flag->freeErrorWindow = FALSE;
              }
          }
        if (theMMDescription[i] == 'r')
          {
            /******** robot data is supported at both sides in both
              directions *********/
            flag->robot = 1;
            intraX = x;
            intraY = y;
            initRoboticsText(x,y,side,inout);
            if ((menu_state == CHANGE_SHOW) && (strcmp(robotIntraSpec,"yes") =
= 0))
              {
                initRobotIntraWindows(intraX,intraY,side,inout);
                flag->initRobotIntra = TRUE;
              }
            y +=6*LINE_DISTANCE;


          }
      } /* for */

  } /* if */
```

```
          XClearWindow(display,theInOutMQDoneWindow);
          displayInOutMQ(event.xbutton.window,side,inout);

        } /* Done Button if */

      if (event.xbutton.window == theListFRWindow)
        {
          highlightChoice(theListFRWindow,
                          "grey",
                          BUTTON_LEVEL2_WIDTH,
                          TEXT_WINDOW_HEIGHT);
          displayInOutMQ(event.xbutton.window,side,inout);
          if (flag->selectTune == 0)
            {
              initFRTune(xl,yl,side,inout);
              flag->selectTune = 1;
              flag->freeListTuneDone = 0;
            }
          highlightChoice(theStartTuneWindow,
                          "grey",
                          TEXT_WINDOW_WIDTH + 50,
                          TEXT_WINDOW_HEIGHT);
           displayInOutMQ(theStartTuneWindow,side,inout);
           displayInOutMQ(theStopTuneWindow,side,inout);

        }
      if (event.xbutton.window == theStartTuneWindow)
        {
          highlightChoice(theStartTuneWindow,
                          "grey",
                          TEXT_WINDOW_WIDTH + 50,
                          TEXT_WINDOW_HEIGHT);
          displayInOutMQ(event.xbutton.window,side,inout);
          XClearWindow(display,theStopTuneWindow);
          displayInOutMQ(theStopTuneWindow,side,inout);
          vrate=getintVideoRate(frameRateValue);
          if ((vrate % 60 == 0) || (60 % vrate == 0))
            {
              setParamValue(VIDEO,S_RATE,NONE,vrate,inout,side);
              startTuneVideo(x,y,side,inout,vrate);
            }
          else
            {
              strcpy(frameRateValue,"300");
              XClearWindow(display, theFrameRateWindow);
              displayInOutMQ(theFrameRateWindow,side,inout);
              setParamValue(VIDEO,S_RATE,NONE,300,inout,side);
              if (vrate > 600)
                {
                  ShowError(BAD_MAXBOUND,VIDEO,S_RATE);
                  flag->freeErrorWindow = FALSE;
                }
              if (vrate < 60)
                {
                  ShowError(BAD_VALUE,VIDEO,S_RATE);
                  flag->freeErrorWindow = FALSE;
                }
              freeListTuneWindows();
              flag->selectTune =0;
              flag->freeListTuneDone =1;
              XClearWindow(display,theListFRWindow);
              displayInOutMQ(theListFRWindow,side,inout);
            }
/* start window with video file at the requested frame rate */
        }
      if (event.xbutton.window == theStopTuneWindow)
        {
          highlightChoice(theStopTuneWindow,
                          "grey",
                          TEXT_WINDOW_WIDTH + 50,
                          TEXT_WINDOW_HEIGHT);
          displayInOutMQ(event.xbutton.window,side,inout);
          XClearWindow(display,theStartTuneWindow);
          displayInOutMQ(theStartTuneWindow,side,inout);
          stopTuneVideo();
         /* close window where the rate is shown */
        }
       if (event.xbutton.window == theListCompressionWindow)
         {
          highlightChoice(theListCompressionWindow,
                          "grey",
                          BUTTON_LEVEL2_WIDTH,
                          TEXT_WINDOW_HEIGHT);
          displayInOutMQ(event.xbutton.window,side,inout);

          initListCompression(xl,yl,side,inout);
          flag->selectCompression = 1;
          highlightChoice(theNoneWindow,
                          "grey",
                          TEXT_WINDOW_WIDTH,
                          TEXT_WINDOW_HEIGHT);
          displayInOutMQ(theNoneWindow,side,inout);
          displayInOutMQ(theJPEGWindow,side,inout);
          displayInOutMQ(theMPEGWindow,side,inout);

         }

      if (event.xbutton.window == theNoneWindow)
        {
          strcpy(compressionValue,"NONE");
          highlightChoice(theNoneWindow,
                          "grey",
                          TEXT_WINDOW_WIDTH,
                          TEXT_WINDOW_HEIGHT);
          displayInOutMQ(theNoneWindow,side,inout);
          XClearWindow(display,theJPEGWindow);
          XClearWindow(display,theMPEGWindow);
          displayInOutMQ(theJPEGWindow,side,inout);
          displayInOutMQ(theMPEGWindow,side,inout);
          setCompressionValue(side,inout);
          XClearWindow(display, theCompressionWindow);
          displayInOutMQ(theCompressionWindow,side,inout);
        }

      if (event.xbutton.window == theJPEGWindow)
        {
          strcpy(compressionValue,"JPEG");
          highlightChoice(theJPEGWindow,
                          "grey",
                          TEXT_WINDOW_WIDTH,
                          TEXT_WINDOW_HEIGHT);
          displayInOutMQ(theJPEGWindow,side,inout);
          XClearWindow(display,theNoneWindow);
          XClearWindow(display,theMPEGWindow);
          displayInOutMQ(theNoneWindow,side,inout);
          displayInOutMQ(theMPEGWindow,side,inout);
/*        setCompressionValue(side,inout); not supported */
          XClearWindow(display, theCompressionWindow);
```

```
          displayInOutMQ(theCompressionWindow,side,inout);
          if (flag->freeErrorWindow)
            {
              ShowError(NOT_SUPPORTED,VIDEO,S_COMPRESS);
              flag->freeErrorWindow=FALSE;
            }
        }
      if (event.xbutton.window == theMPEGWindow)
        {
          strcpy(compressionValue,"MPEG");
          highlightChoice(theMPEGWindow,
                          "grey",
                          TEXT_WINDOW_WIDTH,
                          TEXT_WINDOW_HEIGHT);
          displayInOutMQ(theMPEGWindow,side,inout);
          XClearWindow(display,theJPEGWindow);
          XClearWindow(display,theNoneWindow);
          displayInOutMQ(theJPEGWindow,side,inout);
          displayInOutMQ(theNoneWindow,side,inout);
/*        setCompressionValue(side,inout); */
          XClearWindow(display, theCompressionWindow);
          displayInOutMQ(theCompressionWindow,side,inout);
          if (flag->freeErrorWindow)
            {
              ShowError(NOT_SUPPORTED,VIDEO,S_COMPRESS);
              flag->freeErrorWindow=FALSE;
            }
        }
       break;
     case KeyPress:
       dialogKeyPress(&event,flag,side,inout);
       break;
     }
   return(status);
 }/* -- function InputMQEventLoop */

/***************************** set Procedures *****************************/
/*
** setCompressionValue
*/

setCompressionValue(side,inout)
int side;
int inout;
{
  if (side == MASTER && inout == INPUT)
    {
      if (strcmp(compressionValue,"NONE") == 0)
        MasterInputParam.stream[VIDEO].medium.app_spec.comp_spec.name = NONE;
      if (strcmp(compressionValue,"JPEG") == 0)
        MasterInputParam.stream[VIDEO].medium.app_spec.comp_spec.name = JPEG_COMP;
      if (strcmp(compressionValue,"MPEG") == 0)
        MasterInputParam.stream[VIDEO].medium.app_spec.comp_spec.name = MPEG;
    }
  if (side == MASTER && inout == OUTPUT)
    {
      if (strcmp(compressionValue,"NONE") == 0)
        MasterOutputParam.stream[VIDEO].medium.app_spec.comp_spec.name = NONE;
      if (strcmp(compressionValue,"JPEG") == 0)
        MasterOutputParam.stream[VIDEO].medium.app_spec.comp_spec.name = JPEG_COMP;
      if (strcmp(compressionValue,"MPEG") == 0)
        MasterOutputParam.stream[VIDEO].medium.app_spec.comp_spec.name = MPEG;
    }
  if (side == SLAVE && inout == INPUT)
    {
      if (strcmp(compressionValue,"NONE") == 0)
        SlaveInputParam.stream[VIDEO].medium.app_spec.comp_spec.name = NONE;
      if (strcmp(compressionValue,"JPEG") == 0)
        SlaveInputParam.stream[VIDEO].medium.app_spec.comp_spec.name = JPEG_COMP;
      if (strcmp(compressionValue,"MPEG") == 0)
        SlaveInputParam.stream[VIDEO].medium.app_spec.comp_spec.name = MPEG;
    }
  if (side == SLAVE && inout == INPUT)
    {
      if (strcmp(compressionValue,"NONE") == 0)
        SlaveOutputParam.stream[VIDEO].medium.app_spec.comp_spec.name = NONE;
      if (strcmp(compressionValue,"JPEG") == 0)
        SlaveOutputParam.stream[VIDEO].medium.app_spec.comp_spec.name = JPEG_COMP;
      if (strcmp(compressionValue,"MPEG") == 0)
        SlaveOutputParam.stream[VIDEO].medium.app_spec.comp_spec.name = MPEG;
    }

} /* -- function setCompressionValue */


 freeInOutMQ()
 {
   XFreeGC(display, theInOutQuitGC);
   XFreeGC(display, theInOutCancelGC);

   XFreeGC(display, theInOutDoneGC);
   XFreeGC(display, theInOutGC);


   XDestroySubwindows(display,theInOutMQWindow);
   XDestroyWindow(display,theInOutMQWindow);

 }

 /*
 ** freeMMDescr() frees theMMDescription Window
 */


freeMMDescr(side)
int side;
 {
   XFreeGC(display,theInOutTextGC);

   XDestroyWindow(display, theInOutTextWindow);
 }

 freeVideoText()
 {
   XFreeGC(display, theWidthFrameGC);
   XFreeGC(display, theHeightFrameGC);
   XFreeGC(display, theBitPerPixelGC);
   XFreeGC(display, theCompressionGC);
```

```
   XFreeGC(display, theFrameLossGC);
   XFreeGC(display, theFrameRateGC);
   XFreeGC(display, theVEEdelayGC);
   XFreeGC(display,theListCompressionGC);
   XFreeGC(display,theListFRGC);
 }

 freeAudioText()
 {
   XFreeGC(display, theAudioSampleSizeGC);
   XFreeGC(display, theAudioSampleRateGC);
   XFreeGC(display, theAudioSampleLossRateGC);
   XFreeGC(display, theAEEdelayGC);
 }

freeRobotText(flag)
MQ_FLAG *flag;
 {
   XFreeGC(display, theRobotSampleSizeGC);
   XFreeGC(display, theRobotSampleRateGC);
   XFreeGC(display, theRobotSampleLossRateGC);
   XFreeGC(display, theREEdelayGC);
   XFreeGC(display, theRobotIntraGC);

   if (flag->initRobotIntra == TRUE)
     {
       XFreeGC(display, theRobotNameNGC);
       XFreeGC(display, theRobotNameOGC);
       XFreeGC(display, theRobotNameAGC);
       XFreeGC(display, theRobotNamePGC);
       XFreeGC(display, theRobotSizeNGC);
       XFreeGC(display, theRobotSizeOGC);
       XFreeGC(display, theRobotSizeAGC);
       XFreeGC(display, theRobotSizePGC);
       XFreeGC(display, theRobotRateNGC);
       XFreeGC(display, theRobotRateOGC);
       XFreeGC(display, theRobotRateAGC);
       XFreeGC(display, theRobotRatePGC);
       XFreeGC(display, theRobotPrioNGC);
       XFreeGC(display, theRobotPrioOGC);
       XFreeGC(display, theRobotPrioAGC);
       XFreeGC(display, theRobotPrioPGC);
       XFreeGC(display, theRobotLossNGC);
       XFreeGC(display, theRobotLossOGC);
       XFreeGC(display, theRobotLossAGC);
       XFreeGC(display, theRobotLossPGC);
     }

 }

 /*
 ** dialogKeyPress handles keybord input inot the stringDialog
 */

dialogKeyPress (event,flags,side,inout)
     XKeyEvent   *event;
     MQ_FLAG     *flags;
     int side;
     int inout;
{
  int   length, l,i;
  int   theKeyBufferMaxLen = 64;
  int   theKeyBuffer[65];
  KeySym  theKeySym;
  XComposeStatus   theComposeStatus;

for (i=0; i<65; i++)
  theKeyBuffer[i] =0;


length = XLookupString(event,
                       theKeyBuffer,
                       theKeyBufferMaxLen,
                       &theKeySym,
                       &theComposeStatus);

printf("KeyBuffer is %s \n", theKeyBuffer);

if (event->window == theInOutTextWindow)
  {
    l = strlen(theMMDescription);
  }
if (event->window == theFrameRateWindow)
  {
    l = strlen(frameRateValue);
  }
if (event->window == theRobotSampleRateWindow)
  {
    l = strlen(robotRateValue);
  }
if (event->window == theRobotIntraWindow)
  {
    l = strlen(robotIntraSpec);
  }

if ((theKeySym >= ' ') &&
    (theKeySym <= '~') &&
    (length > 0))
  {
    if ((l+strlen(theKeyBuffer)) < MAX_TEXT_LENGTH)
      {
        if (event->window == theInOutTextWindow)
          {
            strcat(theMMDescription, theKeyBuffer);
            displayInOutMQ(theInOutTextWindow,side,inout);
          }
        if (event->window == theFrameRateWindow)
          {
            strcat(frameRateValue,theKeyBuffer);
            displayInOutMQ(theFrameRateWindow,side,inout);
          }
         if (event->window == theRobotSampleRateWindow)
           {
             strcat(robotRateValue,theKeyBuffer);
             displayInOutMQ(theRobotSampleRateWindow,side,inout);
             flags->changeRobotRate = TRUE;
           }
         if (event->window == theRobotIntraWindow)
           {
             strcat(robotIntraSpec,theKeyBuffer);
             displayInOutMQ(theRobotIntraWindow,side,inout);
           }
      }
  }
 else
   {
     switch(theKeySym)
       {
```

```
        case XK_BackSpace:
        case XK_Delete:
          if (l>=1)
            {
              if (event->window == theInOutTextWindow)
                {
                  XClearWindow(display, theInOutTextWindow);
                  l--;
                  theMMDescription[l] = '\0';
                  displayInOutMQ(theInOutTextWindow,side,inout);
                  XFlush(display);
                }
              if (event->window == theFrameRateWindow)
                {
                  XClearWindow(display, theFrameRateWindow);
                  l--;
                  frameRateValue[l] = '\0';
                  displayInOutMQ(theFrameRateWindow,side,inout);
                  XFlush(display);
                }
              if (event->window == theRobotSampleRateWindow)
                {
                  XClearWindow(display, theRobotSampleRateWindow);
                  l--;
                  robotRateValue[l] = '\0';
                  displayInOutMQ(theRobotSampleRateWindow,side,inout);
                  flags->changeRobotRate =TRUE;
                  XFlush(display);
                }
              if (event->window == theRobotIntraWindow)
                {
                  XClearWindow(display, theRobotIntraWindow);
                  l--;
                  robotIntraSpec[l] = '\0';
                  displayInOutMQ(theRobotIntraWindow,side,inout);
                  XFlush(display);
                }

            }/* if */
          break;
        default:;
        }/* switch */
    }/* else */

 }/* -- function dialogkeyPress */

/*****************************************************************/
/* Initiate SubWindows of the Tune (theListFRWindow) Button      */
/*****************************************************************/
initFRTune(x,y,side,inout)
int x,y;
int side,inout;
{
  Window openWindow();
  int NoneButtonW = 0;

   y += TEXT_WINDOW_HEIGHT;
  x += BUTTON_LEVEL2_WIDTH;
  theStartTuneWindow = openWindow(x,y,
                           TEXT_WINDOW_WIDTH + 50,
                           TEXT_WINDOW_HEIGHT,
                           NORMAL_WINDOW,
                           "Select Window",
                           NORMAL_STATE,
                           theInOutMQWindow,
                           &theStartTuneGC,
                           NoneButtonW);
  associateFont(theStartTuneGC, TEXT1_FONT);
  initEvents(theStartTuneWindow,IN_PALETTE);

  XDefineCursor(display, theStartTuneWindow, theButtonCursor);

  y +=TEXT_WINDOW_HEIGHT + 5;

  theStopTuneWindow = openWindow(x,y,
                           TEXT_WINDOW_WIDTH + 50,
                           TEXT_WINDOW_HEIGHT,
                           NORMAL_WINDOW,
                           "Select Window",
                           NORMAL_STATE,
                           theInOutMQWindow,
                           &theStopTuneGC,
                           NoneButtonW);
  associateFont(theStopTuneGC, TEXT1_FONT);
  initEvents(theStopTuneWindow,IN_PALETTE);

  XDefineCursor(display, theStopTuneWindow, theButtonCursor);

  XFlush(display);
}
initListCompression(x,y,side,inout)
int x,y;
int side,inout;
{
  Window openWindow();
  int NoneButtonW = 0;

   y += TEXT_WINDOW_HEIGHT;
  x += BUTTON_LEVEL2_WIDTH;
  theNoneWindow = openWindow(x,y,
                           TEXT_WINDOW_WIDTH,
                           TEXT_WINDOW_HEIGHT,
                           NORMAL_WINDOW,
                           "Select Window",
                           NORMAL_STATE,
                           theInOutMQWindow,
                           &theNoneGC,
                           NoneButtonW);
  associateFont(theNoneGC, TEXT1_FONT);
  initEvents(theNoneWindow,IN_PALETTE);

  XDefineCursor(display, theNoneWindow, theButtonCursor);

  y +=TEXT_WINDOW_HEIGHT + 5;

  theJPEGWindow = openWindow(x,y,
                           TEXT_WINDOW_WIDTH,
                           TEXT_WINDOW_HEIGHT,
                           NORMAL_WINDOW,
                           "Select Window",
                           NORMAL_STATE,
                           theInOutMQWindow,
                           &theJPEGGC,
                           NoneButtonW);
  associateFont(theJPEGGC, TEXT1_FONT);
  initEvents(theJPEGWindow,IN_PALETTE);

  XDefineCursor(display, theJPEGWindow, theButtonCursor);
```

```
  y +=TEXT_WINDOW_HEIGHT +5;

  theMPEGWindow = openWindow(x,y,
                             TEXT_WINDOW_WIDTH,
                             TEXT_WINDOW_HEIGHT,
                             NORMAL_WINDOW,
                             "Select Window",
                             NORMAL_STATE,
                             theInOutMQWindow,
                             &theMPEGGC,
                             NoneButtonW);
  associateFont(theMPEGGC, TEXT1_FONT);
  initEvents(theMPEGWindow,IN_PALETTE);

  XDefineCursor(display, theMPEGWindow, theButtonCursor);

  XFlush(display);

} /* -- function initListCompresionWindows*/

/************************************************************/
/* A general procedure to set values in application QoS     */
/* Structure                                                */
/************************************************************/

setParamValue(type,param,comp,value,inout,side)
int type;
int param;
int comp;
int value;
int inout;
int side;
{

  switch (param)
    {
    case S_TYPE:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].type = type;
          MasterInputParam.stream[type].direction = inout;
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].type = type;
          MasterOutputParam.stream[type].direction = inout;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].type = type;
          SlaveInputParam.stream[type].direction = inout;
        }
      if (side == SLAVE && inout == OUTPUT)
        {
         SlaveOutputParam.stream[type].type = type;
         SlaveOutputParam.stream[type].direction = inout;
        }
      break;
    case S_INTRA:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].intra = value;
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].intra = value;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].intra = value;
        }
      if (side == SLAVE && inout == OUTPUT)
        {
         SlaveOutputParam.stream[type].intra = value;
        }
      break;
    case QUALITY:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].medium.app_spec.quality = value;
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].medium.app_spec.quality = value;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].medium.app_spec.quality = value;
        }
      if (side == SLAVE && inout == OUTPUT)
        {
          SlaveOutputParam.stream[type].medium.app_spec.quality = value;
        }
      break;
    case S_SIZE:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].medium.app_spec.sample_size = value;
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].medium.app_spec.sample_size = value;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].medium.app_spec.sample_size = value;
        }
      if (side == SLAVE && inout == OUTPUT)
        {
          SlaveOutputParam.stream[type].medium.app_spec.sample_size = value;
        }
          break;
    case S_RATE:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].medium.app_spec.sample_rate = value;
          printf("value %d \n",value);
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].medium.app_spec.sample_rate = value;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].medium.app_spec.sample_rate = value;
        }
      if (side == SLAVE && inout == OUTPUT)
```

```
        {
          SlaveOutputParam.stream[type].medium.app_spec.sample_rate = value;
        }
      break;
    case S_LOSS:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].medium.net_spec.loss_rate = value;
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].medium.net_spec.loss_rate = value;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].medium.net_spec.loss_rate = value;
        }
      if (side == SLAVE && inout == OUTPUT)
        {
          SlaveOutputParam.stream[type].medium.net_spec.loss_rate = value;
        }
      break;
    case S_DELAY:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].medium.net_spec.end_to_end_delay=value;
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].medium.net_spec.end_to_end_delay=value;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].medium.net_spec.end_to_end_delay=value;
        }
      if (side == SLAVE && inout == OUTPUT)
        {
          SlaveOutputParam.stream[type].medium.net_spec.end_to_end_delay=value;
        }
      break;
    case S_PRIO:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].medium.net_spec.importance=value;
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].medium.net_spec.importance=value;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].medium.net_spec.importance=value;
        }
      if (side == SLAVE && inout == OUTPUT)
        {
          SlaveOutputParam.stream[type].medium.net_spec.importance=value;
        }
      break;
    case C_NAME:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].component_spec[comp].name = value;
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].component_spec[comp].name = value;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].component_spec[comp].name = value;
        }
      if (side == SLAVE && inout == OUTPUT)
        {
          SlaveOutputParam.stream[type].component_spec[comp].name = value;
        }
      break;
    case C_SIZE:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].component_spec[comp].size = value;
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].component_spec[comp].size = value;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].component_spec[comp].size = value;
        }
      if (side == SLAVE && inout == OUTPUT)
        {
          SlaveOutputParam.stream[type].component_spec[comp].size = value;
        }
      break;
    case C_RATE:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].component_spec[comp].rate =value;
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].component_spec[comp].rate =value;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].component_spec[comp].rate =value;
        }
      if (side == SLAVE && inout == OUTPUT)
        {
          SlaveOutputParam.stream[type].component_spec[comp].rate =value;
        }
      break;
    case C_PRIO:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].component_spec[comp].importance = value;
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].component_spec[comp].importance = value;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].component_spec[comp].importance = value;
        }
      if (side == SLAVE && inout == OUTPUT)
        {
          SlaveOutputParam.stream[type].component_spec[comp].importance = value;
```

```
        }
      break;
    case C_LOSS:
      if (side == MASTER && inout == INPUT)
        {
          MasterInputParam.stream[type].component_spec[comp].loss = value;
        }
      if (side == MASTER && inout == OUTPUT)
        {
          MasterOutputParam.stream[type].component_spec[comp].loss = value;
        }
      if (side == SLAVE && inout == INPUT)
        {
          SlaveInputParam.stream[type].component_spec[comp].loss = value;
        }
      if (side == SLAVE && inout == OUTPUT)
        {
          SlaveOutputParam.stream[type].component_spec[comp].loss = value;
        }
      break;
    }
}
freeListCompressionWindows()
{

  XFreeGC(display, theNoneGC);
  XFreeGC(display, theJPEGGC);
  XFreeGC(display, theMPEGGC);
  XDestroyWindow(display,theNoneWindow);
  XDestroyWindow(display,theJPEGWindow);
  XDestroyWindow(display,theMPEGWindow);
  }
freeListTuneWindows()
{
  XFreeGC(display, theStartTuneGC);
  XDestroyWindow(display,theStartTuneWindow);
  XFreeGC(display, theStopTuneGC);
  XDestroyWindow(display,theStopTuneWindow);
}
```

```
/*************************************************************************/
/* Filename: dialogCMSMediaRelation.c                                    */
/* Purpose : X window dialog for menu " Media Relations " at Master Side */
/* Author  : Klara Nahrstedt                                             */
/* Update  : 07/5/95                                                     */
/*************************************************************************/

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>
#include <X11/keysymdef.h>

#include "/home/klara/tele.d/include.d/retta.h"
#include "/home/klara/tele.d/include.d/defs.h"
#include "/home/klara/tele.d/include.d/qos.h"


extern Display *display;

char theSyncDescription[MAX_CHAR];
char theIntegration[MAX_CHAR]; /* yes or no */
char theConvertFromMediaDescription[MAX_CHAR];
char theConvertToMediaDescription[MAX_CHAR];

extern char MMMasterInputDescription[MAX_CHAR];
extern char MMMasterOutputDescription[MAX_CHAR];
extern char MMSlaveInputDescription[MAX_CHAR];
extern char MMSlaveOutputDescription[MAX_CHAR];

/************************* QoS parameters ****************************/

extern APP_QOS MasterInputParam;
extern APP_QOS MasterOutputParam;
extern APP_QOS SlaveInputParam;
extern APP_QOS SlaveOutputParam;

/* menu state variable */

extern int menu_state;

extern Cursor theArrowCursor;
extern Cursor theTextCursor;
extern Cursor theQuitCursor;
extern Cursor theButtonCursor;
extern Cursor theBusyCursor;

extern Window theCallQoSWindow;
extern Window theRootWindow;
extern Window theErrorWindow;
extern GC theCallQoSGC;

/* Windows in theCallQoSWindow*/

Window theMRWindow; /* subwindow in CallQoSWindow */
GC     theMRGC;
/* Subwindows - Buttons in MRWindow */
Window theMRQuitWindow;
GC     theMRQuitGC;
Window theMRCancelWindow;
GC     theMRCancelGC;
Window theMRDoneWindow;
GC     theMRDoneGC;
Window theSynchronizationWindow;
GC     theSynchronizationGC;
Window theIntegrationWindow;
GC     theIntegrationGC;
Window theConvertFromMediaWindow;
GC     theConvertFromMediaGC;
Window theConvertToMediaWindow;
GC     theConvertToMediaGC;
Window theCommunicationWindow;
GC     theCommunicationGC;

/* test */



QoSMediaRelations(x,y,side)
int x,y,side;
{
  int width,height;
  int theChoice;
  int i;
  MR_FLAG flag;
  width = MAIN_WINDOW_WIDTH - 6*DISTANCE;
  height = MAIN_WINDOW_HEIGHT - 100 - 9*DISTANCE;

  if (menu_state == CONFIGURE)
    {
      for (i=0; i<MAX_CHAR;i++)
        {
          theSyncDescription[i] = '\0';
          theIntegration[i] = '\0';
        }
      strcpy(theSyncDescription,"none");
      strcpy(theIntegration,"no");
    }


  initMRWindows(x,y,width,height,side);

  displayMR(theMRQuitWindow,side);
  displayMR(theMRCancelWindow,side);
  displayMR(theMRDoneWindow,side);

  theChoice = (-1);

  bzero((char *)(&flag),sizeof(MR_FLAG));

  while (theChoice==-1)
    {
      theChoice = MREventLoop(side,x,y,&flag);
    }

  freeMRWindows(side);
}/* -- function QoSMediaRelations */

initMRWindows(x,y,width,height,side)
int x,y,width,height,side;

{

  Window openWindow();
  int ButtonW = 1;
  int NoneButtonW = 0;
```

```
  theMRWindow = openWindow(x,y,
                           width, height,
                           NORMAL_WINDOW,
                           "QoSMediaRelations",
                           NORMAL_STATE,
                           theCallQoSWindow,
                           &theMRGC,
                           NoneButtonW);
  associateFont(theMRGC,TEXT1_FONT);
  initEvents(theMRWindow,IN_PALETTE);
  XDefineCursor(display, theMRWindow,theArrowCursor);

  x = DISTANCE;
  y = height - 2*DISTANCE;

  theMRQuitWindow = openWindow(x,y,
                               BUTTON_LEVEL2_WIDTH,
                               BUTTON_LEVEL2_HEIGHT,
                               NORMAL_WINDOW,
                               "QoSMediaRelations",
                               NORMAL_STATE,
                               theMRWindow,
                               &theMRQuitGC,
                               ButtonW);
  associateFont(theMRQuitGC,TEXT1_FONT);
  initEvents(theMRQuitWindow,IN_PALETTE);
  XDefineCursor(display, theMRQuitWindow,theButtonCursor);

  x += (DISTANCE+BUTTON_LEVEL2_WIDTH);

  theMRCancelWindow = openWindow(x,y,
                                 BUTTON_LEVEL2_WIDTH,
                                 BUTTON_LEVEL2_HEIGHT,
                                 NORMAL_WINDOW,
                                 "QoSMediaRelations",
                                 NORMAL_STATE,
                                 theMRWindow,
                                 &theMRCancelGC,
                                 ButtonW);
  associateFont(theMRCancelGC,TEXT1_FONT);
  initEvents(theMRCancelWindow,IN_PALETTE);
  XDefineCursor(display, theMRCancelWindow,theButtonCursor);

  x += (DISTANCE + BUTTON_LEVEL2_WIDTH);

  theMRDoneWindow = openWindow(x,y,
                               BUTTON_LEVEL2_WIDTH,
                               BUTTON_LEVEL2_HEIGHT,
                               NORMAL_WINDOW,
                               "QoSMediaRelations",
                               NORMAL_STATE,
                               theMRWindow,
                               &theMRDoneGC,
                               ButtonW);
  associateFont(theMRDoneGC,TEXT1_FONT);
  initEvents(theMRDoneWindow,IN_PALETTE);
  XDefineCursor(display, theMRDoneWindow,theButtonCursor);

  x = DISTANCE;
  y = DISTANCE;

  initMRText(x,y,side);
}/* -- function initMRWindows*/

initMRText(x,y,side)
int x,y,side;
{
  int NoneButtonW = 0;

  theSynchronizationWindow = openWindow(x + 400,
                                        y - 15,
                                        TEXT_WINDOW_WIDTH,
                                        TEXT_WINDOW_HEIGHT,
                                        NORMAL_WINDOW,
                                        "Synchronization",
                                        NORMAL_STATE,
                                        theMRWindow,
                                        &theSynchronizationGC,
                                        NoneButtonW);
  associateFont(theSynchronizationGC,TEXT1_FONT);
  initEvents(theSynchronizationWindow,IN_PALETTE);
  XDefineCursor(display, theSynchronizationWindow,theTextCursor);

  XDrawString(display,theMRWindow,
              theMRGC,
              x,y,
              "Synchronization of Receiving Media (e.g., v,a,r):",
              strlen("Synchronization of Receiving Media (e.g., v,a,r):"));
  displayMR(theSynchronizationWindow,side);
/********************* standard value ********************************/

    setRelationValue(R_SYNCH,FALSE,side);

  y +=2*DISTANCE;


  theIntegrationWindow = openWindow(x + 300,
                                    y - 15,
                                    TEXT_WINDOW_WIDTH,
                                    TEXT_WINDOW_HEIGHT,
                                    NORMAL_WINDOW,
                                    "Synchronization",
                                    NORMAL_STATE,
                                    theMRWindow,
                                    &theIntegrationGC,
                                    NoneButtonW);
  associateFont(theIntegrationGC,TEXT1_FONT);
  initEvents(theIntegrationWindow,IN_PALETTE);
  XDefineCursor(display, theIntegrationWindow,theTextCursor);
  XDrawString(display,theMRWindow,
              theMRGC,
              x,y,
              "Integration of Sending Media (yes/no):",
              strlen("Integration of Sending Media (yes/no)"));
  displayMR(theIntegrationWindow,side);
/********************* standard value ********************************/

    setRelationValue(R_INTEG,FALSE,side);

  y +=2*DISTANCE;

  XDrawString(display, theMRWindow,
              theMRGC,
              x,y,
              "Precedence of Media",
              strlen("Precedence of Media"));
```

```
  y +=DISTANCE;

  theConvertFromMediaWindow = openWindow(x + 300,
                                  y - 15,
                                  TEXT_WINDOW_WIDTH,
                                  TEXT_WINDOW_HEIGHT,
                                  NORMAL_WINDOW,
                                  "Integration",
                                  NORMAL_STATE,
                                  theMRWindow,
                                  &theConvertFromMediaGC,
                                  NoneButtonW);
  associateFont(theConvertFromMediaGC,TEXT1_FONT);
  initEvents(theConvertFromMediaWindow,IN_PALETTE);
  XDefineCursor(display, theConvertFromMediaWindow,theTextCursor);
  XDrawString(display, theMRWindow,
              theMRGC,
              x+85,y,
              "first(read robot data):",
              strlen("first(read robot data):"));
  displayMR(theConvertFromMediaWindow,side);

  theConvertToMediaWindow = openWindow(x +560 ,
                                  y - 15,
                                  TEXT_WINDOW_WIDTH,
                                  TEXT_WINDOW_HEIGHT,
                                  NORMAL_WINDOW,
                                  "Integration",
                                  NORMAL_STATE,
                                  theMRWindow,
                                  &theConvertToMediaGC,
                                  NoneButtonW);
  associateFont(theConvertToMediaGC,TEXT1_FONT);
  initEvents(theConvertToMediaWindow,IN_PALETTE);
  XDefineCursor(display, theConvertToMediaWindow,theTextCursor);
  XDrawString(display, theMRWindow,
              theMRGC,
              x+360,y,
              "second(write robot data):",
              strlen("second(write robot data):"));
  displayMR(theConvertToMediaWindow,side);
/******************** standard value ********************************/

    setRelationValue(R_CONV,FALSE,side);

  y +=2*DISTANCE;

  theCommunicationWindow = openWindow(x + 200,
                                  y - 15,
                                  TEXT_WINDOW_WIDTH,
                                  TEXT_WINDOW_HEIGHT,
                                  NORMAL_WINDOW,
                                  "Integration",
                                  NORMAL_STATE,
                                  theMRWindow,
                                  &theCommunicationGC,
                                  NoneButtonW);
  associateFont(theCommunicationGC,TEXT1_FONT);
  initEvents(theCommunicationWindow,IN_PALETTE);
  XDefineCursor(display, theCommunicationWindow,theTextCursor);
  XDrawString(display, theMRWindow,
              theMRGC,
              x,y,
              "Communication Relation :",
              strlen("Communication Relation :"));
  displayMR(theCommunicationWindow,side);
/******************** standard value ********************************/

    setRelationValue(R_COMM,UNICAST,side);

}/* -- function initMRText*/

displayMR(window,side)
Window window;
int side;
{

  int y;

  y = textHeight(TEXT1_FONT) + 2;

  if (window == theMRQuitWindow)
    {
      XDrawString(display,window,theMRQuitGC,
                  5,y,
                  "Quit",
                  strlen("Quit"));
    }

    if (window == theMRCancelWindow)
    {
      XDrawString(display,window,theMRCancelGC,
                  5,y,
                  "Cancel",
                  strlen("Cancel"));
    }

  if (window == theMRDoneWindow)
    {
      XDrawString(display,window,theMRDoneGC,
                  10,y,
                  "Done",
                  strlen("Done"));
    }

  if (window == theSynchronizationWindow)
    {
      XDrawString(display,window,theSynchronizationGC,
                  5,y -2,
                  theSyncDescription,
                  strlen(theSyncDescription));
   }
  if (window == theCommunicationWindow)
    {
      XDrawString(display,window,theCommunicationGC,
                  5,y-2,
                  "unicast",
                  strlen("unicast"));
    }

  if (window == theConvertFromMediaWindow)
    {
      XDrawString(display,window,theConvertFromMediaGC,
                  5,y-2,
                  "yes",
                  strlen("yes"));
```

```
    }

    if (window == theConvertToMediaWindow)
    {
      XDrawString(display,window,theConvertToMediaGC,
                  5,y-2,
                  "yes",
                  strlen("yes"));
    }

  if (window == theIntegrationWindow)
    {
      XDrawString(display,window,theIntegrationGC,
                  5,y-2,
                  theIntegration,
                  strlen(theIntegration));



    }
}/* -- function displayMR */

MREventLoop(side,x,y,flag)
int side;
int x,y;
MR_FLAG *flag;
{
  int status = (-1);
  XEvent   event;
  int i;

  XNextEvent(display,&event);

  switch(event.type)
    {
    case ConfigureNotify:
    case Expose:
    case MapNotify:
      displayMR(event.xany.window,side);
      break;
    case ButtonPress:
      if (event.xbutton.window == theMRQuitWindow)
        {
          status = 0;
          if (flag->errorWindow == TRUE)
            {
              freeError();
              flag->errorWindow = FALSE;
            }
        }
      if (event.xbutton.window == theMRCancelWindow)
        {
          status = 1;
          if (flag->errorWindow == TRUE)
            {
              freeError();
              flag->errorWindow = FALSE;
            }

          if (strcmp(theSyncDescription,"none")!=0)
            {
              strcpy(theSyncDescription,"none");
              flag->selectSync = FALSE;
             }
        }

      if (event.xbutton.window == theErrorWindow)
        {
          if (flag->selectSync == TRUE)
            {
              strcpy(theSyncDescription,"none");
              flag->selectSync = FALSE;

              displayMR(theSynchronizationWindow,side);
            }
          if (flag->errorWindow == TRUE)
            {
              freeError();
              flag->errorWindow = FALSE;
            }
        }
      if (event.xbutton.window == theMRDoneWindow)
        {
          highlightChoice(theMRDoneWindow,
                          "blue",
                          BUTTON_LEVEL2_WIDTH,
                          BUTTON_LEVEL2_HEIGHT);

          if (strcmp(theSyncDescription,"none")!=0)
            {
              strcpy(theSyncDescription,"none");
              flag->selectSync = FALSE;
              ShowError(NOT_SUPPORTED,OTHER,R_SYNCH);
              flag->errorWindow = TRUE;
              XClearWindow(display,theSynchronizationWindow);
              displayMR(theSynchronizationWindow,side);
            }

          XClearWindow(display,theMRDoneWindow);
          displayMR(event.xbutton.window,side);
        }
      break;
    case KeyPress:
      dialogKey(&event,side,flag);
      break;
    }/* switch */
  return(status);
}/* -- function MREventLoop */

dialogKey(event,side,flag)
XKeyEvent   *event;
int side;
MR_FLAG *flag;
{

  int length,l,i;
  int theKeyBufferMaxLen = 64;
  int theKeyBuffer[65];
  KeySym theKeySym;
  XComposeStatus  theComposeStatus;

  for (i=0;i<65;i++)
    theKeyBuffer[i] =0;


  length = XLookupString(event,
                         theKeyBuffer,
                         theKeyBufferMaxLen,
```

```
                        &theKeySym,
                        &theComposeStatus);

  if (event->window == theSynchronizationWindow)
    {
      XClearWindow(display,theSynchronizationWindow);
      l = strlen(theSyncDescription);
    }
  if (event->window == theConvertFromMediaWindow)
    {
      l = strlen(theConvertFromMediaDescription);
    }
  if (event->window == theConvertToMediaWindow)
    {
      l = strlen(theConvertToMediaWindow);
    }

   if ((theKeySym >= ' ') &&
       (theKeySym <= '~') &&
       (length > 0))
     {
       if ((l+strlen(theKeyBuffer)) < MAX_TEXT_LENGTH)
         {
           if (event->window == theSynchronizationWindow)
             {
               strcat(theSyncDescription,theKeyBuffer);
               displayMR(theSynchronizationWindow,side);
               flag->selectSync = TRUE;
             }
         }
     }
   else
     {
       switch(theKeySym)
         {
         case XK_BackSpace:
         case XK_Delete:
           if (l>=1)
             {
               if (event->window == theSynchronizationWindow)
                 {
                   XClearWindow(display,event->window);
                   l--;
                   theSyncDescription[l] = '\0';
                   displayMR(event->window,side);
                   XFlush(display);
                   flag->selectSync = TRUE;
                 }
             }
           break;
         default:;
         }/* switch */
     }/* else */
 }/* -- function dialogkeyPress */

freeMRWindows(side)
int side;
{
  XFreeGC(display,theMRGC);
  XFreeGC(display,theMRQuitGC);
  XFreeGC(display,theMRCancelGC);
  XFreeGC(display,theMRDoneGC);
  XFreeGC(display,theSynchronizationGC);
  XFreeGC(display,theIntegrationGC);
  XFreeGC(display,theConvertFromMediaGC);
  XFreeGC(display,theConvertToMediaGC);
  XDestroySubwindows(display,theMRWindow);
  XDestroyWindow(display,theMRWindow);
}

/**** set Values into application QoS structure about relations *********/

setRelationValue(relation,value,side)
int relation;
int value;
int side;
{
  switch (relation)
    {
    case R_SYNCH:
      /*************************************************************************/
      /* At the Master Input side there is no synchronization, only          */
      /* receiving (Output) side needs to provide synchronization relation   */
      /* Therefor, MasterInputParam.relation.done[R_SYNCH) = FALSE; always   */
      /*************************************************************************/
      if (side == MASTER)
        {
          MasterInputParam.relations.done[R_SYNCH] = FALSE;
          MasterOutputParam.relations.done[R_SYNCH] = value;
        }
      /*************************************************************************/
      /* At the Slave Input side there is no synchronization, only           */
      /* receiving (Output) side needs to provide synchronization relation   */
      /*************************************************************************/
      if (side == SLAVE)
        {
          SlaveInputParam.relations.done[R_SYNCH]=FALSE;
          SlaveOutputParam.relations.done[R_SYNCH] = value; /* FALSE or TRUE */
        }
      break;
    case R_INTEG:
      /*************************************************************************/
      /* At the        Output side there is no integration, only             */
      /* sending (Input) side needs to provide integration relation, the     */
      /* receving side desintegrates, hence it has the same value as input   */
      /* side                                                                */
      /*************************************************************************/
      if (side == MASTER)
        {
          MasterInputParam.relations.done[R_INTEG] = value;
          SlaveOutputParam.relations.done[R_INTEG] = value;
        }
      if (side == SLAVE)
        {
          SlaveInputParam.relations.done[R_INTEG] = value;
          MasterOutputParam.relations.done[R_INTEG] = value; /* FALSE or TRUE */
        }
      break;
    case R_CONV:
      /*************************************************************************/
      /* We consider conversion relation at the receiver side, but there     */
      /* are  also other possibilities                                       */
      /*************************************************************************/

      if (side == MASTER )
        {
          MasterInputParam.relations.done[R_CONV] = FALSE;
          MasterOutputParam.relations.done[R_CONV] = value;
```

```
        }
      if (side == SLAVE)
        {
          SlaveInputParam.relations.done[R_CONV] = FALSE;
          SlaveOutputParam.relations.done[R_CONV] = value; /* FALSE or TRUE */
        }
      break;
    case R_COMM:
      /*********************************************************************/
      /* The specification of communication is done at the sender side,    */
      /* at the receiver side it is always unicast                         */
      /*********************************************************************/

      if (side == MASTER)
        {
          MasterInputParam.relations.done[R_COMM] = value;
          MasterOutputParam.relations.done[R_COMM] = UNICAST;
        }
      if (side == SLAVE)
        {
          SlaveInputParam.relations.done[R_COMM] = value;
          SlaveOutputParam.relations.done[R_COMM] = UNICAST; /* UNICAST or MULTICAST */
        }
      break;
    }
}
```

```
/*************************************************************************/
/* Filename: dialogCallMS.c                                              */
/* Purpose : Set a signal, unidirectional connection from MASTER/SLAVE
 to SLAVE/MASTER*/
/* Author  : KLara Nahrstedt                                             */
/* Update  : 7/03/95                                                     */
/*************************************************************************/

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>
#include <X11/keysymdef.h>

#include "/home/klara/tele.d/include.d/defs.h"
#include "/home/klara/tele.d/include.d/retta.h"
#include "/home/klara/tele.d/include.d/comm.h"

/* Menu state variable CONFIGURE or CHANGE */
/* it translates into NEG or RENEG option  */

extern Display    *display;
extern int menu_state;
extern Cursor theArrowCursor;
extern Cursor theTextCursor;
extern Cursor theQuitCursor;
extern Cursor theButtonCursor;
extern Cursor theBusyCursor;

extern Window theDCWindow;
extern GC     theDCGC;

/* subwindows of Call window */

 /* window for communicatio about the negotiation */

Window theCallQoSWindow;
GC     theCallQoSGC;

/* buttons in theCallQoSwindow */

Window theCallMSQuitWindow;
GC     theCallMSQuitGC;
Window theCallMSCancelWindow;
GC     theCallMSCancelGC;
Window theCallMSNegWindow;
GC     theCallMSNegGC;
Window theCallMSChangeParamWindow;
GC     theCallMSChangeParamGC;
Window theCallMSChangeRelWindow;
GC     theCallMSChangeRelGC;
Window theCallMSDoneWindow;
GC     theCallMSDoneGC;

/* text windows in theCallQoSwindow */

Window theAddInfoWindow;
GC     theAddInfoGC;
Window theGetImageWindow;
GC     theGetImageGC;
Window theSetPositionWindow;
GC     theSetPositionGC;
Window theParkPositionWindow;
GC     theParkPositionGC;
Window theReadyPositionWindow;
GC     theReadyPositionGC;

char theAddInfoDescription[4];

/* network connected parameters which are shared */

/* quality parameters to be negotiated */

extern MM_DEVICES MasterInputDevices;
extern MM_DEVICES MasterOutputDevices;
extern MM_DEVICES SlaveInputDevices;
extern MM_DEVICES SlaveOutputDevices;

extern APP_QOS MasterInputParam;
extern APP_QOS MasterOutputParam;
extern APP_QOS SlaveInputParam;
extern APP_QOS SlaveOutputParam;

ADD_INFO AddInfo;



int CallMSSetUp(x,y,side,MenuStateFD)
int x;
int y;
int side;
int MenuStateFD;
{
  int width,height;
  int theChoice;
  ADD_FLAG flags;
  /*************************************************************************/
  /* Initialize Variables                                                  */
  /*************************************************************************/

  width = MAIN_WINDOW_WIDTH - 4*DISTANCE;
  height = MAIN_WINDOW_HEIGHT - 100 - 5*DISTANCE;
  bzero((char *)(&flags),sizeof(ADD_FLAG));
  bzero((char *)(&AddInfo),sizeof(ADD_INFO));


  strcpy(theAddInfoDescription,"no");

  initCallMSWindows(x,y,width,height,side);

  displayCallMSWindows(theCallMSQuitWindow,side);
  displayCallMSWindows(theCallMSCancelWindow,side);
  displayCallMSWindows(theCallMSNegWindow,side);
  displayCallMSWindows(theCallMSChangeParamWindow,side);
  displayCallMSWindows(theCallMSChangeRelWindow,side);
  if (side == MASTER)
    {
      displayCallMSWindows(theAddInfoWindow,side);
    }
  displayCallMSWindows(theCallMSDoneWindow,side);

  theChoice = (-1);

  while (theChoice == -1)
    {
      theChoice = CallMSEventLoop(side, &flags,MenuStateFD);
    }

  freeCallMSWindows();
```

```
}/* CallMSSetUp*/


initCallMSWindows(x,y,width,height,side)
int x,y,width,height,side;
{
  Window openWindow();
  int ButtonW = 1;
  int NoneButtonW = 0;

  theCallQoSWindow = openWindow(x,y,
                                width,height,
                                NORMAL_WINDOW,
                                "CallSetUpMS",
                                NORMAL_STATE,
                                theDCWindow,
                                &theCallQoSGC,
                                NoneButtonW);

  associateFont(theCallQoSGC, TEXT1_FONT);
  initEvents(theCallQoSWindow, IN_PALETTE);
  XDefineCursor(display, theCallQoSWindow, theArrowCursor);

  x = DISTANCE;
  y = height - 2*DISTANCE;

  theCallMSQuitWindow = openWindow(x,y,
                                   BUTTON_LEVEL2_WIDTH,
                                   BUTTON_LEVEL2_HEIGHT,
                                   NORMAL_WINDOW,
                                   "CallSetUpMS",
                                   NORMAL_STATE,
                                   theCallQoSWindow,
                                   &theCallMSQuitGC,
                                   ButtonW);

  associateFont(theCallMSQuitGC, TEXT1_FONT);
  initEvents(theCallMSQuitWindow, IN_PALETTE);
  XDefineCursor(display, theCallMSQuitWindow, theButtonCursor);

  x +=(DISTANCE+BUTTON_LEVEL2_WIDTH);

  theCallMSCancelWindow = openWindow(x,y,
                                   BUTTON_LEVEL2_WIDTH,
                                   BUTTON_LEVEL2_HEIGHT,
                                   NORMAL_WINDOW,
                                   "CallSetUpMS",
                                   NORMAL_STATE,
                                   theCallQoSWindow,
                                   &theCallMSCancelGC,
                                   ButtonW);

  associateFont(theCallMSCancelGC, TEXT1_FONT);
  initEvents(theCallMSCancelWindow, IN_PALETTE);
  XDefineCursor(display, theCallMSCancelWindow, theButtonCursor);

  x +=(DISTANCE + BUTTON_LEVEL2_WIDTH);
  theCallMSDoneWindow = openWindow(x,y,
                                   BUTTON_LEVEL2_WIDTH,
                                   BUTTON_LEVEL2_HEIGHT,
                                   NORMAL_WINDOW,
                                   "CallSetUpMS",
                                   NORMAL_STATE,
                                   theCallQoSWindow,
                                   &theCallMSDoneGC,
                                   ButtonW);


  associateFont(theCallMSDoneGC, TEXT1_FONT);
  initEvents(theCallMSDoneWindow, IN_PALETTE);
  XDefineCursor(display, theCallMSDoneWindow, theButtonCursor);

  x +=(DISTANCE + BUTTON_LEVEL2_WIDTH);

  theCallMSNegWindow = openWindow(x,y,
                                   BUTTON_LEVEL2_WIDTH,
                                   BUTTON_LEVEL2_HEIGHT,
                                   NORMAL_WINDOW,
                                   "CallSetUpMS",
                                   NORMAL_STATE,
                                   theCallQoSWindow,
                                   &theCallMSNegGC,
                                   ButtonW);

  associateFont(theCallMSNegGC, TEXT1_FONT);
  initEvents(theCallMSNegWindow, IN_PALETTE);
  XDefineCursor(display, theCallMSNegWindow, theButtonCursor);

  x +=(DISTANCE + BUTTON_LEVEL2_WIDTH);
  theCallMSChangeParamWindow = openWindow(x,y,
                                   BUTTON_LEVEL2_WIDTH,
                                   BUTTON_LEVEL2_HEIGHT,
                                   NORMAL_WINDOW,
                                   "CallSetUpMS",
                                   NORMAL_STATE,
                                   theCallQoSWindow,
                                   &theCallMSChangeParamGC,
                                   ButtonW);

  associateFont(theCallMSChangeParamGC, TEXT1_FONT);
  initEvents(theCallMSChangeParamWindow, IN_PALETTE);
  XDefineCursor(display, theCallMSChangeParamWindow, theButtonCursor);


  x +=(DISTANCE + BUTTON_LEVEL2_WIDTH);
  theCallMSChangeRelWindow = openWindow(x,y,
                                   BUTTON_LEVEL2_WIDTH,
                                   BUTTON_LEVEL2_HEIGHT,
                                   NORMAL_WINDOW,
                                   "CallSetUpMS",
                                   NORMAL_STATE,
                                   theCallQoSWindow,
                                   &theCallMSChangeRelGC,
                                   ButtonW);

  associateFont(theCallMSChangeRelGC, TEXT1_FONT);
  initEvents(theCallMSChangeRelWindow, IN_PALETTE);
  XDefineCursor(display, theCallMSChangeRelWindow, theButtonCursor);

  /********** print text ***********************************/
  x=DISTANCE;
  y=DISTANCE;

  XDrawString(display,theCallQoSWindow,
              theCallQoSGC,
              x,y,
              "Press =Connect= Button to Begin with Call Set Up ",
              strlen( "Press =Connect= Button to begin with call set up "));
```

```
  if (side == MASTER)
    {
      x = DISTANCE;
      y = 3*DISTANCE;

      XDrawString(display,theCallQoSWindow,
                  theCallQoSGC,
                  x,y,
                  "Additional Information (yes/no) ",
                  strlen( "Additional Information (yes/no) "));

      theAddInfoWindow = openWindow(x + 250,
                                    y - 10 ,
                                    TEXT_WINDOW_WIDTH,
                                    TEXT_WINDOW_HEIGHT,
                                    NORMAL_WINDOW,
                                    "CallSetUpMS",
                                    NORMAL_STATE,
                                    theCallQoSWindow,
                                    &theAddInfoGC,
                                    NoneButtonW);

      associateFont(theAddInfoGC, TEXT1_FONT);
      initEvents(theAddInfoWindow, IN_PALETTE);
      XDefineCursor(display,theAddInfoWindow, theTextCursor);
    }

}/*initCallMSWindows */


displayCallMSWindows(window,side)
     Window window;
     int side;
{
  int y;

  y = textHeight(TEXT1_FONT) + 2;

  if (window == theCallMSQuitWindow)
    {

      XDrawString(display,theCallMSQuitWindow,
              theCallMSQuitGC,
              5,y,
              "Quit",
              strlen( "Quit"));
    }


  if (window == theCallMSCancelWindow)
    {

      XDrawString(display,window,
              theCallMSCancelGC,
              5,y,
              "Cancel",
              strlen( "Cancel"));
    }

  if (window == theCallMSDoneWindow)
    {

      XDrawString(display,window,
                  theCallMSDoneGC,
                  5,y,
                  "Done",
                  strlen( "Done"));
    }

  if (window == theCallMSNegWindow)
    {

      XDrawString(display,window,
              theCallMSNegGC,
              2,y,
              "Connect",
              strlen( "Connect"));
    }


  if (window == theCallMSChangeParamWindow)
    {

      XDrawString(display,window,
              theCallMSChangeParamGC,
              2,y,
              "Mod/Par",
              strlen( "Mod/Par"));
    }


  if (window == theCallMSChangeRelWindow)
    {
      XDrawString(display,window,
              theCallMSChangeRelGC,
              2,y,
              "Mod/Rel",
              strlen( "Mod/Rel"));
    }
  if (side == MASTER)
    {
      if (window == theAddInfoWindow)
        {
          XDrawString(display,window,
                      theAddInfoGC,
                      5,y-3,
                      theAddInfoDescription,
                      strlen(theAddInfoDescription));
        }

      if (window == theGetImageWindow)
        {
          XDrawString(display,window,
                      theGetImageGC,
                      5,y-3,
                      "Get Image",
                      strlen("Get Image"));
        }

      if (window == theSetPositionWindow)
        {
          XDrawString(display,window,
                      theSetPositionGC,
                      5,y-3,
                      "Set Position",
                      strlen("Set Position"));
        }
```

```
      if (window == theParkPositionWindow)
        {
          XDrawString(display,window,
                      theParkPositionGC,
                      5,y-3,
                      "Park",
                      strlen("Park"));
        }
      if (window == theReadyPositionWindow)
        {
          XDrawString(display,window,
                      theReadyPositionGC,
                      5,y-3,
                      "Ready",
                      strlen("Ready"));
        }
    }
  XFlush(display);

}/* displayCallMSWindows*/


int CallMSEventLoop(side,flags,MenuStateFD)
int side;
ADD_FLAG *flags;
int MenuStateFD;
{
  int status = (-1);
  XEvent event;
  int xt,yt;
  int i;
  NOTIFY notification;
  int MenuControl;
  int changedVideoRate;


  xt = DISTANCE;
  yt = DISTANCE;


  XNextEvent(display,&event);

  switch(event.type)
    {
    case ConfigureNotify:
    case Expose:
    case MapNotify:
      displayCallMSWindows(event.xbutton.window,side);
      break;
    case ButtonPress:
      if (event.xbutton.window == theCallMSQuitWindow)
        {
          status = 0;
        }
      if (event.xbutton.window == theCallMSCancelWindow)
        {
          status = 1;
        }

      if (event.xbutton.window == theCallMSDoneWindow)
        {
          highlightChoice(theCallMSDoneWindow,
                          "grey",
                          BUTTON_LEVEL2_WIDTH,
                          BUTTON_LEVEL2_HEIGHT);
      displayCallMSWindows(theCallMSDoneWindow,side);

      if ( strcmp(theAddInfoDescription,"yes") == 0)
        {
          if (flags->AddInfoWindows == FALSE)
            {
              addInfo(xt,yt,side);
              flags->AddInfoWindows = TRUE;
            }
        }
      XClearWindow(display,theCallMSDoneWindow);
      displayCallMSWindows(theCallMSDoneWindow,side);
    }
  if (side == MASTER)
    {
      if (event.xbutton.window == theGetImageWindow)
        {
          highlightChoice(theGetImageWindow,
                          "grey",
                          BUTTON_LEVEL2_WIDTH + 20,
                          BUTTON_LEVEL2_HEIGHT);
          displayCallMSWindows(theGetImageWindow,side);
          setAddInfo(GET_IMAGE,TRUE,NONE,REQUEST,side);
        }
      if (event.xbutton.window == theParkPositionWindow)
        {
          XClearWindow(display,theReadyPositionWindow);
          displayCallMSWindows(theReadyPositionWindow,side);

          highlightChoice(theParkPositionWindow,
                          "grey",
                          BUTTON_LEVEL2_WIDTH ,
                          BUTTON_LEVEL2_HEIGHT);
          displayCallMSWindows(theParkPositionWindow,side);
          setAddInfo(SET_POSITION,TRUE,PARK,REQUEST,side);
        }
      if (event.xbutton.window == theReadyPositionWindow)
        {
          XClearWindow(display,theParkPositionWindow);
          displayCallMSWindows(theParkPositionWindow,side);

          highlightChoice(theReadyPositionWindow,
                          "grey",
                          BUTTON_LEVEL2_WIDTH ,
                          BUTTON_LEVEL2_HEIGHT);
          displayCallMSWindows(theReadyPositionWindow,side);
          setAddInfo(SET_POSITION,TRUE,READY,REQUEST,side);
        }

      if (event.xbutton.window == theSetPositionWindow)
        {
          highlightChoice(theSetPositionWindow,
                          "grey",
                          BUTTON_LEVEL2_WIDTH + 20,
                          BUTTON_LEVEL2_HEIGHT);
          displayCallMSWindows(theSetPositionWindow,side);
          if (flags->SetPositionWindows == FALSE)
            {
              showPosition(xt,yt,side);
              flags->SetPositionWindows = TRUE;
            }
          XClearWindow(display,theSetPosit
```

```
            displayCallMSWindows(theSetPositionWindow,side);

          }
        }
      if (event.xbutton.window == theCallMSNegWindow)
        {
          XClearWindow(display,theCallQoSWindow);
          if (side == MASTER)
            freeAddInfoWindows(flags);
          XDrawString(display,theCallQoSWindow,
                      theCallQoSGC,
                      DISTANCE, DISTANCE-5,
                      "Please, wait, negotiation of QoS is going on!",
                      strlen("Please, wait, negotiation of QoS is going on!"));
          if (side == MASTER)
            {
/************* Call Set Up from Master to Slave ****************************/
            bcopy((char *)(&SlaveOutputParam),(char *)(&AddInfo.other_qos),sizeof(APP_
QOS));
            QoSBroker(&MasterInputParam,
                      &AddInfo,
                      &notification,
                      BUYER,
                      INPUT,
                      NEGOTIATE);
            }
          if (side == SLAVE)
            {
              bcopy((char *)(&SlaveInputParam),(char *)(&AddInfo.other_qos),sizeof(APP_Q
OS));
/*
              for (i=0;i<MEDIA_NUMBER;i++)
                {
                  printf("MMSetup: Master Output Param=(type,size,rate)=(%d,%d,%d)\n",
                         MasterOutputParam.stream[i].type,
                         MasterOutputParam.stream[i].medium.app_spec.sample_size,
                         MasterOutputParam.stream[i].medium.app_spec.sample_rate);
                }
*/

            QoSBroker(&MasterOutputParam,
                      &AddInfo,
                      &notification,
                      BUYER,
                      OUTPUT,
                      NEGOTIATE);


            }
          XClearWindow(display, theCallQoSWindow);
          switch (notification.note)
            {
            case NEG_SUCCESS:
              switch(notification.reason)
                {
                case GET_IMAGE_SUCCESS:
                  XDrawString(display,theCallQoSWindow,
                              theCallQoSGC,
                              DISTANCE, DISTANCE+15,
                              "Get-Image operation is done. Exit the current window and
start with CALL SET-UP to establish QoS connection!",
                              strlen("Get-Image operation is done. Exit the current wind
ow and start with CALL SET-UP to establish QoS connection!"));
                  break;
                default:
                  XDrawString(display,theCallQoSWindow,
                              theCallQoSGC,
                              DISTANCE, DISTANCE-5,
                              "Negotiation of QoS was successful, multimedia connection i
s established!",
                              strlen("Negotiation of QoS was successful, multimedia conne
ction is established!"));
                  break;
                }
              break;
            case NEG_REJECT:
              XDrawString(display,theCallQoSWindow,
                          theCallQoSGC,
                          DISTANCE, DISTANCE-5,
                          "Negotiation of QoS was rejected",
                          strlen("Negotiation of QoS was rejected"));
              switch(notification.reason)
                {
                case VIDEO_NOT_SUPPORTED:
                  XDrawString(display,theCallQoSWindow,
                              theCallQoSGC,
                              DISTANCE, DISTANCE+5,
                              "Video not supported",
                              strlen("Video not supported"));
                  break;
                case END_TO_END_TEST_FAILURE:
                  XDrawString(display,theCallQoSWindow,
                              theCallQoSGC,
                              DISTANCE, DISTANCE + 25,
                              "End-to-end delay test failed",
                              strlen("End-to-end delay test failed"));
                  break;
                case SCHEDULE_NOT_FEASIBLE:
                  XDrawString(display,theCallQoSWindow,
                              theCallQoSGC,
                              DISTANCE+5, DISTANCE+5,
                              "Schedule not feasible",
                              strlen("Schedule not feasible"));
                  break;

                }
              break;
            case NEG_MODIFY:
              XDrawString(display,theCallQoSWindow,
                          theCallQoSGC,
                          DISTANCE, DISTANCE-5,
                          "Negotiation of QoS was modified, multimedia connection establi
shed with modified QoS!",
                          strlen("Negotiation of QoS was modified, multimedia connection
established with modified QoS!"));
              break;
            }
        }

      if (event.xbutton.window == theCallMSChangeParamWindow)
        {
          XClearWindow(display,theCallQoSWindow);
          menu_state = CHANGE_SHOW;
          setColorWithName(theCallQoSGC,"blue");
          XDrawString(display,theCallQoSWindow,
                      theCallQoSGC,
                      DISTANCE, DISTANCE-5,
                      "Media Quality Parameter
```

```
                    strlen("Media Quality Parameters"));
          QoSMediaQuality(xt,yt,side);


          lseek(MenuStateFD,0L,0);
          read(MenuStateFD,(char *)(&MenuControl),sizeof(int));
          if (MenuControl == START)
            {

              MenuControl=RENEGOTIATE;
              lseek(MenuStateFD,0L,0);
              write(MenuStateFD,(char *)(&MenuControl),sizeof(int));
              changedVideoRate = SlaveInputParam.stream[VIDEO].medium.app_spec.sample_ra
te;
              /*get changed rate in frames per minute, must compute
                frames per second */
              changedVideoRate = changedVideoRate/60;
              printf("video rate = %d \n",changedVideoRate);
              write(MenuStateFD,(char *)(&changedVideoRate),sizeof(int));
              sleep(5);

/*
              QoSBroker(&MasterOutputParam,
                        &AddInfo,
                        &notification,
                        BUYER,
                        OUTPUT,
                        RENEGOTIATE);
*/


            }

          XClearWindow(display,theCallQoSWindow);
          setColorWithName(theCallQoSGC,"black");

          XDrawString(display,theCallQoSWindow,
                      theCallQoSGC,
                      xt,yt,
                      "Press =Connect= Button to Begin with Call Set Up ",
                      strlen( "Press =Connect= Button to begin with call set up "));
        }

      if (event.xbutton.window == theCallMSChangeRelWindow)
        {
          XClearWindow(display,theCallQoSWindow);
          menu_state = CHANGE_SHOW;
          setColorWithName(theCallQoSGC,"blue");
          XDrawString(display,theCallQoSWindow,
                      theCallQoSGC,
                      DISTANCE, DISTANCE-5,
                      "Media Relations",
                      strlen("Media Relations"));
          QoSMediaRelations(xt,yt,side);
          XClearWindow(display,theCallQoSWindow);
          setColorWithName(theCallQoSGC,"black");

          XDrawString(display,theCallQoSWindow,
                      theCallQoSGC,
                      xt,yt,
                      "Press =Connect= Button to Begin with Call Set Up ",
                      strlen( "Press =Connect= Button to begin with call set up "));
        }
      break;
        case KeyPress:
          dialogKeyInfo(&event,side);
          break;
        }
      return(status);

}/* CallMSEventLoop*/

dialogKeyInfo(event,side)
XKeyEvent  *event;
int side;
{
  int length,l,i;
  int theKeyBufferMaxLen = 64;
  int theKeyBuffer[65];
  KeySym theKeySym;
  XComposeStatus theComposeStatus;

  for (i=0; i<65; i++)
    theKeyBuffer[i] = 0;
  length = XLookupString(event,
                         theKeyBuffer,
                         theKeyBufferMaxLen,
                         &theKeySym,
                         &theComposeStatus);
  if (event->window == theAddInfoWindow && side == MASTER)
    {
      l = strlen(theAddInfoDescription);
    }


  if ((theKeySym >= ' ') &&
      (theKeySym <= '~') &&
      (length > 0))
    {
      if ((l+strlen(theKeyBuffer)) < MAX_TEXT_LENGTH)
        {
          if (event->window == theAddInfoWindow && side == MASTER)
            {
              strcat(theAddInfoDescription,theKeyBuffer);
              displayCallMSWindows(theAddInfoWindow,side);
            }
        }
    }
  else
    {
      switch(theKeySym)
        {
        case XK_BackSpace:
        case XK_Delete:
          if (l>=1)
            {
              if (event->window == theAddInfoWindow && side == MASTER)
                {
                  XClearWindow(display, theAddInfoWindow);
                  l--;
                  theAddInfoDescription[l] = '\0';
                  displayCallMSWindows(theAddInfoWindow,side);
                  XFlush(display);
                }
            }
          break;
        default:;
        }/*switch*/
```

```
    }/*else*/

}/* dialogKeyInfo*/

/************** add information for QoS negotiation ******************/

addInfo(x,y,side)
int x;
int y;
int side;
{
  int ButtonW = 1;

  y = 3*y;
  theGetImageWindow = openWindow(x + 250 + TEXT_WINDOW_WIDTH,
                                 y -10 + TEXT_WINDOW_HEIGHT ,
                                 BUTTON_LEVEL2_WIDTH + 20,
                                 BUTTON_LEVEL2_HEIGHT,
                                 NORMAL_WINDOW,
                                 "CallSetUpMS",
                                 NORMAL_STATE,
                                 theCallQoSWindow,
                                 &theGetImageGC,
                                 ButtonW);

  associateFont(theGetImageGC, TEXT1_FONT);
  initEvents(theGetImageWindow, IN_PALETTE);
  XDefineCursor(display,theGetImageWindow, theButtonCursor);
  displayCallMSWindows(theGetImageWindow,side);
  setAddInfo(GET_IMAGE,FALSE,NONE,REQUEST,side);

  theSetPositionWindow = openWindow(x + 250 + TEXT_WINDOW_WIDTH,
                                 y -10 + TEXT_WINDOW_HEIGHT + BUTTON_LEVEL2_HEIGHT,
                                 BUTTON_LEVEL2_WIDTH +20,
                                 BUTTON_LEVEL2_HEIGHT,
                                 NORMAL_WINDOW,
                                 "CallSetUpMS",
                                 NORMAL_STATE,
                                 theCallQoSWindow,
                                 &theSetPositionGC,
                                 ButtonW);

  associateFont(theSetPositionGC, TEXT1_FONT);
  initEvents(theSetPositionWindow, IN_PALETTE);
  XDefineCursor(display,theSetPositionWindow, theButtonCursor);
  displayCallMSWindows(theSetPositionWindow,side);
  setAddInfo(SET_POSITION,FALSE,NONE,REQUEST,side);
}/* addInfo */

showPosition(x,y,side)
int x;
int y;
int side;
{
  int ButtonW = 1;

  x = x + 250 + 2*(BUTTON_LEVEL2_HEIGHT + 20) + TEXT_WINDOW_WIDTH;
  y = y -10 + TEXT_WINDOW_HEIGHT + 4*BUTTON_LEVEL2_HEIGHT;

  theParkPositionWindow = openWindow(x,
                                 y,
                                 BUTTON_LEVEL2_WIDTH,
                                 BUTTON_LEVEL2_HEIGHT,
                                 NORMAL_WINDOW,
                                 "CallSetUpMS",
                                 NORMAL_STATE,
                                 theCallQoSWindow,
                                 &theParkPositionGC,
                                 ButtonW);

  associateFont(theParkPositionGC, TEXT1_FONT);
  initEvents(theParkPositionWindow, IN_PALETTE);
  XDefineCursor(display,theParkPositionWindow, theButtonCursor);
  displayCallMSWindows(theParkPositionWindow,side);

  y = y + BUTTON_LEVEL2_HEIGHT;
  theReadyPositionWindow = openWindow(x,
                                 y,
                                 BUTTON_LEVEL2_WIDTH,
                                 BUTTON_LEVEL2_HEIGHT,
                                 NORMAL_WINDOW,
                                 "CallSetUpMS",
                                 NORMAL_STATE,
                                 theCallQoSWindow,
                                 &theReadyPositionGC,
                                 ButtonW);

  associateFont(theReadyPositionGC, TEXT1_FONT);
  initEvents(theReadyPositionWindow, IN_PALETTE);
  XDefineCursor(display,theReadyPositionWindow, theButtonCursor);
  highlightChoice(theReadyPositionWindow,
                          "grey",
                          BUTTON_LEVEL2_WIDTH,
                          BUTTON_LEVEL2_HEIGHT);
  displayCallMSWindows(theReadyPositionWindow,side);

  setAddInfo(SET_POSITION,TRUE,READY,REQUEST,side);
}/* showPosition */

setAddInfo(param,done,value,direction,side)
int param;
int done;
int value;
int direction;
int side;
{
  switch(param)
    {
    case GET_IMAGE:
      AddInfo.info[GET_IMAGE].side = side;
      AddInfo.info[GET_IMAGE].direction = direction;
      AddInfo.info[GET_IMAGE].param = GET_IMAGE;
      AddInfo.info[GET_IMAGE].done = done;
      AddInfo.info[GET_IMAGE].value = value;
      break;
    case SET_POSITION:
      AddInfo.info[SET_POSITION].side = side;
      AddInfo.info[SET_POSITION].direction = direction;
      AddInfo.info[SET_POSITION].param = SET_POSITION;
      AddInfo.info[SET_POSITION].done = done;
      AddInfo.info[SET_POSITION].value = value;
      break;

    }
}/* setAddInfo*/

freeAddInfoWindows(flags)
ADD_FLAG *flags;
```

```
{
  XFreeGC(display, theAddInfoGC);
  XDestroyWindow(display,theAddInfoWindow);

  if (flags->AddInfoWindows == TRUE)
    {
      XFreeGC(display,theGetImageGC);
      XFreeGC(display,theSetPositionGC);
      XDestroyWindow(display,theGetImageWindow);
      XDestroyWindow(display,theSetPositionWindow);
    }
  if (flags->SetPositionWindows == TRUE)
    {
      XFreeGC(display,theParkPositionGC);
      XFreeGC(display,theReadyPositionGC);
      XDestroyWindow(display,theParkPositionWindow);
      XDestroyWindow(display,theReadyPositionWindow);
    }
}/*freeAddInfoWindows*/


freeCallMSWindows()
{

 XFreeGC(display,theCallQoSGC);
 XFreeGC(display,theCallMSQuitGC);
 XFreeGC(display,theCallMSCancelGC);
 XFreeGC(display,theCallMSNegGC);
 XFreeGC(display, theCallMSChangeParamGC);
 XFreeGC(display, theCallMSChangeRelGC);
 XFreeGC(display,theCallMSDoneGC);

 XDestroySubwindows(display,theCallQoSWindow);
 XDestroyWindow(display,theCallQoSWindow);

}/* freeCallMSWindows */
```

```
/*********************************************************************/
/* Filename: callSetUp.c                                             */
/* Purpose :Dialog window for Call Set Up between master and slaves
** this activity includes further: call of QoS broker                */
/* Author : Klara Nahrstedt                                          */
/* Update : 07/01/95                                                 */
/*********************************************************************/


#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>
#include <X11/keysymdef.h>

#include "/home/klara/tele.d/include.d/retta.h"


/*  Menu state variable  */

extern int menu_state;

extern Display  *display;

extern Cursor theArrowCursor;
extern Cursor theTextCursor;
extern Cursor theQuitCursor;
extern Cursor theButtonCursor;
extern Cursor theBusyCursor;

extern Window theRootWindow;
extern Window theTeleroboticsWindow;


/*
** subwindows of Telerobotics window
** Call set up between master and slave
*/

Window theDCWindow;
/* Subwindows of CallWindow */
/* Button windows */

Window theCallCancelWindow;
Window theCallMasterSlaveWindow;
Window theCallSlaveMasterWindow;


GC theDCGC;
GC theCallCancelGC;
GC theCallMasterSlaveGC;
GC theCallSlaveMasterGC;


CallSetUp(x,y,MenuStateFD)
int x,y;
int MenuStateFD;
{
  int theChoice;
  int width,height;

  width = MAIN_WINDOW_WIDTH - 2*DISTANCE;
  height = MAIN_WINDOW_HEIGHT - 100 - DISTANCE;

  initCallWindows(x,y,width,height);

  displayCallDialog(theCallCancelWindow,MenuStateFD);

  /* Handle Dialog Window Events */
  theChoice  = (-1) ;

  while (theChoice == -1)
    {
      theChoice = CallEventLoop(MenuStateFD);
    }


   freeCallDialog();

} /* QoSConfigurationDialog*/

initCallWindows(x,y,width,height)
int x,y,width,height;
{
   Window openWindow();
   int ButtonW =1;
   int NoneButtonW =0;

   /* Main DialogConfiguration Box window */

   theDCWindow = openWindow(x,y,width,height,
                            NORMAL_WINDOW,
                            "Call Set Up",
                            NORMAL_STATE,
                            theTeleroboticsWindow,
                            &theDCGC,NoneButtonW);
   associateFont(theDCGC, TEXT1_FONT);
   initEvents(theDCWindow,IN_PALETTE);
   XDefineCursor(display,theDCWindow, theArrowCursor);


   x = DISTANCE;
   y = DISTANCE;

   width =  width - 2*DISTANCE;
   height = height - 4*DISTANCE ;

   initTextWindows(x,y,width,height);

   x = DISTANCE;
   y = height + 2*DISTANCE;

   theCallCancelWindow = openWindow(x,y,
                                    BUTTON_LEVEL2_WIDTH,BUTTON_LEVEL2_HEIGHT,
                                    NORMAL_WINDOW,
                                    "Call Set Up",
                                    NORMAL_STATE,
                                    theDCWindow,
                                    &theCallCancelGC,ButtonW);

   associateFont(theCallCancelGC, TEXT1_FONT);
   initEvents(theCallCancelWindow,IN_PALETTE);
   XDefineCursor(display,theCallCancelWindow, theQuitCursor);


   XFlush(display);
 }/* --function initDialogWindows */
/*
** initTextWindows
```

```
*/

initTextWindows(x,y,width,height)
int x,y;
{
  Window openWindow();
  int ButtonW = 1;
  int NoneButtonW =0;

  x = DISTANCE;
  y = DISTANCE;

  theCallMasterSlaveWindow = openWindow(x,y,
                            BUTTON_LEVEL3_WIDTH,
                            BUTTON_LEVEL3_HEIGHT,
                            NORMAL_WINDOW,
                            "Call From Master To Slave",
                            NORMAL_STATE,
                            theDCWindow,
                            &theCallMasterSlaveGC,
                            ButtonW);

  initEvents(theCallMasterSlaveWindow, IN_PALETTE);

  XDrawString(display,theDCWindow, theDCGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Call Set Up From Master To Slave",
              strlen( "Call Set Up From Master To Slave"));

  y +=(BUTTON_LEVEL3_HEIGHT + DISTANCE);

  theCallSlaveMasterWindow = openWindow(x,y,
                            BUTTON_LEVEL3_WIDTH,
                            BUTTON_LEVEL3_HEIGHT,
                            NORMAL_WINDOW,
                            "Call From Slave To Master",
                            NORMAL_STATE,
                            theDCWindow,
                            &theCallSlaveMasterGC,
                            ButtonW);

  initEvents(theCallSlaveMasterWindow, IN_PALETTE);


  XDrawString(display,theDCWindow, theDCGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Call Set Up From Slave To Master",
              strlen( "Call Set Up From Slave To Master"));

   XDefineCursor(display,theCallMasterSlaveWindow, theButtonCursor);
   XDefineCursor(display,theCallSlaveMasterWindow, theButtonCursor);


}/* -- initTextWindows */



/*
** displayDialog
*/

displayCallDialog(window,MenuStateFD)
Window  window;
int MenuStateFD;
{
  int xt,yt;
  int y,width,height;

  y = textHeight(TEXT1_FONT) + 2;


  if (window == theCallCancelWindow)
    {
      XDrawString(display, window, theCallCancelGC,
                  10,y,
                  "Quit",
                  strlen("Quit"));
    }

  if (window == theDCWindow)
    {

      xt = DISTANCE;
      yt = DISTANCE;

      width = MAIN_WINDOW_WIDTH - 4*DISTANCE;
     height = MAIN_WINDOW_HEIGHT - 100 -5*DISTANCE;

     initTextWindows(xt,yt,width,height);
    }

  if (window ==theCallMasterSlaveWindow)
    {
      XClearWindow(display,theDCWindow);
      freeCallText();
      xt = DISTANCE;
      yt = DISTANCE;

      setColorWithName(theDCGC, "blue");
      XDrawString(display, theDCWindow,theDCGC,
                  DISTANCE,DISTANCE-5,
                  "Call Set Up From Master To Slave",
                  strlen("Call Set Up From Master To Slave"));

     /* next window menu */

       CallMSSetUp(xt,yt,MASTER,MenuStateFD);

    }

  if (window == theCallSlaveMasterWindow)
    {
      XClearWindow(display,theDCWindow);
      freeCallText();
      xt = DISTANCE;
      yt = DISTANCE;

      setColorWithName(theDCGC, "blue");
      XDrawString(display, theDCWindow,theDCGC,
                  DISTANCE,DISTANCE-5,
                  "Call Set From Slave To Master",
                  strlen("Call Set Up From Slave To Master"));

      CallMSSetUp(xt,yt,SLAVE,MenuStateFD);

    }
```

```
    XFlush(display);
}/* -- function displayCallDialog */

/*
** CallEventLoop handles all the dialog events
*/

CallEventLoop(MenuStateFD)
int MenuStateFD;
{
  int   status = (-1);
  XEvent    event;

  XNextEvent(display, &event);

  switch(event.type)
    {
    case ConfigureNotify:
    case Expose:
    case MapNotify:
      displayCallDialog(event.xany.window,MenuStateFD);
      break;
    case ButtonPress:
      if (event.xbutton.window == theCallCancelWindow)
        {
          status = 0;
        }

      if (event.xbutton.window == theCallMasterSlaveWindow)
        {
          displayCallDialog(event.xbutton.window,MenuStateFD);
          XClearWindow(display,theDCWindow);
          setColorWithName(theDCGC,"black");
          displayCallDialog(theDCWindow,MenuStateFD);
        }

      if (event.xbutton.window ==theCallSlaveMasterWindow)
        {
          displayCallDialog(event.xbutton.window,MenuStateFD);
          XClearWindow(display,theDCWindow);
          setColorWithName(theDCGC,"black");
          displayCallDialog(theDCWindow,MenuStateFD);
        }

      break;
    }
  return(status);
} /* -- function dialogEventLoop */


/*
** freeDialog
*/

freeCallDialog()
{
  XFreeGC(display, theDCGC);
  XFreeGC(display, theCallCancelGC);
  XFreeGC(display, theCallMasterSlaveGC);
  XFreeGC(display, theCallSlaveMasterGC);

/*
** Destroy all windows
*/

  XDestroySubwindows(display, theDCWindow);
  XDestroyWindow(display, theDCWindow);
  XFlush(display);
}/* -- function freeDialog */

/*
** freeCallText
*/

freeCallText()
{
 XFreeGC(display,theCallMasterSlaveGC);
 XFreeGC(display,theCallSlaveMasterGC);

 XDestroyWindow(display,theCallMasterSlaveWindow);
 XDestroyWindow(display,theCallSlaveMasterWindow);

}
```

```
/*****************************************************************************/
/* Filename : dialogConfMS.c                                                 */
/* Purpose  : X window dialog for menu "Configure QoS Parameters at Master Side" */
/* Author   : Klara Nahrstedt                                                */
/* Update   : 06/17/94                                                       */
/*****************************************************************************/

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>
#include <X11/keysymdef.h>

#include "/home/klara/tele.d/include.d/retta.h"

extern Display *display;

extern Cursor theArrowCursor;
extern Cursor theTextCursor;
extern Cursor theQuitCursor;
extern Cursor theButtonCursor;
extern Cursor theBusyCursor;

extern Window theDCWindow;
extern Window theRootWindow;

extern GC theDCGC;

Window theCallQoSWindow;
Window theTX1CancelWindow;


/* subwindows of TX1QoS Window */

Window theTX11Window;
Window theTX12Window;

GC theCallQoSGC;
GC theTX1CancelGC;


GC theTX11GC;
GC theTX12GC;


QoSsetUp(x,y,side)
int x,y,side;
{
  int width,height;
  int theChoice;

  width = MAIN_WINDOW_WIDTH - 4*DISTANCE;
  height = MAIN_WINDOW_HEIGHT - 100 - 5*DISTANCE;

  initTX1Windows(x,y, width,height);

  displayTX1(theTX1CancelWindow,side);

  theChoice = -1;

  while (theChoice == -1)
    {
      theChoice = TX1EventLoop(side);
    }

  freeTX1Windows();

} /* -- function QoSsetUpMasterSide */

initTX1Windows(x,y,width,height)
int x,y,width,height;
{

  Window openWindow();

  int ButtonW =1;
  int NoneButtonW =0;

  theCallQoSWindow = openWindow( x,y,
                                 width,height,
                                 NORMAL_WINDOW,
                                 "QoSSetUpMaster",
                                 NORMAL_STATE,
                                 theDCWindow,
                                 &theCallQoSGC,
                                 NoneButtonW);
  associateFont(theCallQoSGC,TEXT1_FONT);


   initEvents(theCallQoSWindow,IN_PALETTE);

  XDefineCursor(display,theCallQoSWindow,theArrowCursor);


  x = DISTANCE;
  y = height - 2*DISTANCE;

  theTX1CancelWindow = openWindow( x,y,
                                 BUTTON_LEVEL2_WIDTH,
                                  BUTTON_LEVEL2_HEIGHT,
                                 NORMAL_WINDOW,
                                 "QoSSetUpMaster",
                                 NORMAL_STATE,
                                 theCallQoSWindow,
                                 &theTX1CancelGC,
                                 ButtonW);
  associateFont(theTX1CancelGC,TEXT1_FONT);

  initEvents(theTX1CancelWindow,IN_PALETTE);

  XDefineCursor(display,theTX1CancelWindow,theQuitCursor);

   x =DISTANCE;
  y =DISTANCE;


  initTX1TextWindows(x,y);


 XFlush(display);
}/* -- function initTX1Windows */

/*
** initTX1TextWindows
*/

initTX1TextWindows(x,y)
int x,y;
```

```
{
  int NoneButtonW = 0;
  int ButtonW = 1;


  theTX11Window = openWindow( x,y,
                              BUTTON_LEVEL3_WIDTH,
                              BUTTON_LEVEL3_HEIGHT,
                            NORMAL_WINDOW,
                            "MediaQuality",
                            NORMAL_STATE,
                            theCallQoSWindow,
                            &theTX11GC,
                            ButtonW);
  associateFont(theTX11GC,TEXT1_FONT);
  initEvents(theTX11Window,IN_PALETTE);

  XDefineCursor(display,theTX11Window,theButtonCursor);

      XDrawString(display,theCallQoSWindow, theCallQoSGC,
                  x + BUTTON_LEVEL3_WIDTH + DISTANCE,
                  y + BUTTON_LEVEL3_HEIGHT,
                  "Media Quality Parameters",
                  strlen("Media Quality Parameters"));

  y += (BUTTON_LEVEL3_HEIGHT + DISTANCE);

  theTX12Window = openWindow( x,y,
                            BUTTON_LEVEL3_WIDTH,
                            BUTTON_LEVEL3_HEIGHT,
                              NORMAL_WINDOW,
                              "MediaQuality",
                              NORMAL_STATE,
                              theCallQoSWindow,
                              &theTX12GC,
                              ButtonW);
  associateFont(theTX12GC,TEXT1_FONT);
  initEvents(theTX12Window,IN_PALETTE);

  XDefineCursor(display,theTX12Window,theButtonCursor);

  XDrawString(display,theCallQoSWindow, theCallQoSGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Media Relations",
              strlen("Media Relations"));
  XFlush(display);

}/* -- function initTX1TextWindows */


/*
** displayTX1
*/

displayTX1(window,side)
Window window;
int side;

{
  int xt,yt;
  int y;

  y = textHeight(TEXT1_FONT) + 2;

  if (window == theTX1CancelWindow)
    {
      XDrawString(display,window, theTX1CancelGC,
                  10,y,
                  "Quit",
                  strlen("Quit"));
    }

  if (window == theCallQoSWindow)
    {
      xt = DISTANCE;
      yt = DISTANCE;

      initTX1TextWindows(xt,yt);
    }
  if (window == theTX11Window)
    {
      XClearWindow(display,theCallQoSWindow);
      freeTX1Text();

      xt = DISTANCE;
      yt = DISTANCE;

     setColorWithName(theCallQoSGC, "blue");
      XDrawString(display,theCallQoSWindow,
                  theCallQoSGC,
                  DISTANCE, DISTANCE -5,
                  "Media Quality Parameters",
                  strlen("Media Quality Parameters"));

      QoSMediaQuality(xt,yt,side);

      }

  if (window == theTX12Window)
    {
      XClearWindow(display,theCallQoSWindow);
      freeTX1Text();

      xt = DISTANCE;
      yt = DISTANCE;

     setColorWithName(theCallQoSGC, "blue");
      XDrawString(display,theCallQoSWindow,
                  theCallQoSGC,
                  DISTANCE, DISTANCE -5,
                  "Media Relations",
                  strlen("Media Relations"));
      QoSMediaRelations(xt,yt,side);
      }

  XFlush(display);
} /* -- function displayTX1 */

/*
** TX1EventLoop
*/

TX1EventLoop(side)
int side;
{
  int status = (-1);
  XEvent event;
```

```
  XNextEvent(display, &event);

  switch(event.type)
    {
    case ConfigureNotify:
    case Expose:
    case MapNotify:
      displayTX1(event.xany.window,side);
      break;
    case ButtonPress:
       if (event.xbutton.window == theTX1CancelWindow)
        {
          highlightChoice(theTX1CancelWindow,
                          "blue",
                          BUTTON_LEVEL2_WIDTH,
                          BUTTON_LEVEL2_HEIGHT);
          displayTX1(event.xbutton.window,side);
          status = 0;
        }

      if (event.xbutton.window == theTX11Window)
        {
          highlightChoice(theTX11Window,
                          "blue",
                          BUTTON_LEVEL3_WIDTH,
                          BUTTON_LEVEL3_HEIGHT);
          displayTX1(event.xbutton.window,side);
          XClearWindow(display,theCallQoSWindow);
          setColorWithName(theCallQoSGC,"black");
          displayTX1(theCallQoSWindow,side);

        }

      if (event.xbutton.window == theTX12Window)
        {
          highlightChoice(theTX12Window,
                          "blue",
                          BUTTON_LEVEL3_WIDTH,
                          BUTTON_LEVEL3_HEIGHT);
          displayTX1(event.xbutton.window,side);
          XClearWindow(display,theCallQoSWindow);
          setColorWithName(theCallQoSGC,"black");
          displayTX1(theCallQoSWindow,side);
        }

      break;
    }
  return(status);
}/* -- function TX1EventLoop */



freeTX1Windows()
{

  XFreeGC(display, theCallQoSGC);
  XFreeGC(display,theTX1CancelGC);

  XDestroySubwindows(display,theCallQoSWindow);
  XDestroyWindow(display,theCallQoSWindow);
}

freeTX1Text()
{

  XFreeGC(display,theTX11GC);
  XFreeGC(display,theTX12GC);

  XDestroyWindow(display,theTX11Window);
  XDestroyWindow(display,theTX12Window);
}
```

```
/*************************************************************************/
/* Filename: dialogConfigure.c                                           */
/* Purpose : Dialog window for the QoS Configuration Button              */
/* Author  : Klara Nahrstedt                                             */
/* Update  : 06/17/94                                                    */
/*************************************************************************/


#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>
#include <X11/keysymdef.h>

#include "/home/klara/tele.d/include.d/retta.h"


/*  Menu state variable  */

int menu_state;

extern Display  *display;

extern Cursor theArrowCursor;
extern Cursor theTextCursor;
extern Cursor theQuitCursor;
extern Cursor theButtonCursor;
extern Cursor theBusyCursor;

extern Window theRootWindow;
extern Window theTeleroboticsWindow;

#define MAX_CHARS    80

/*
** subwindows of Telerobotics window
** QoS configuration dialog windows
*/

Window theDCWindow;
/* Subwindows of DCWindow */

Window theDCCancelWindow;

/* subwindows of TextWindow */

Window theTX1Window;
Window theTX2Window;
Window theTX3Window;
Window theTX4Window;

GC theDCGC;
GC theDCCancelGC;
GC theTX1GC;
GC theTX2GC;
GC theTX3GC;
GC theTX4GC;

char theDialogText[MAX_CHARS + 5];

QoSConfigurationDialog(x,y)
int x,y;

{
  int theChoice;
  int width,height;

  width = MAIN_WINDOW_WIDTH - 2*DISTANCE;
  height = MAIN_WINDOW_HEIGHT - 100 - DISTANCE;

  initDCWindows(x,y,width,height);

  displayDialog(theDCCancelWindow);

  /* Handle Dialog Window Events */
  theChoice  = (-1) ;

  while (theChoice == -1)
    {
      theChoice = dialogEventLoop();
    }


   freeDialog();

} /* QoSConfigurationDialog*/

initDCWindows(x,y,width,height)
int x,y,width,height;
{
   Window openWindow();
   int ButtonW =1;
   int NoneButtonW =0;

   /* Main DialogConfiguration Box window */

   theDCWindow = openWindow(x,y,width,height,
                            NORMAL_WINDOW,
                            "Configuration",
                            NORMAL_STATE,
                            theTeleroboticsWindow,
                            &theDCGC,NoneButtonW);
   associateFont(theDCGC, TEXT1_FONT);
   initEvents(theDCWindow,IN_PALETTE);


   x = DISTANCE;
   y = DISTANCE;

   width =  width - 2*DISTANCE;
   height = height - 4*DISTANCE ;

   initTextWindows(x,y,width,height);

   x = DISTANCE;
   y = height + 2*DISTANCE;

   theDCCancelWindow = openWindow(x,y,
                                  BUTTON_LEVEL2_WIDTH,BUTTON_LEVEL2_HEIGHT,
                                  NORMAL_WINDOW,
                                  "Configuration",
                                  NORMAL_STATE,
                                  theDCWindow,
                                  &theDCCancelGC,ButtonW);

   associateFont(theDCCancelGC, TEXT1_FONT);
   initEvents(theDCCancelWindow,IN_PALETTE);
```

```
   XDefineCursor(display,theDCWindow, theArrowCursor);
   XDefineCursor(display,theDCCancelWindow, theQuitCursor);


   XFlush(display);
 }/* --function initDialogWindows */
/*
** initTextWindows
*/

initTextWindows(x,y,width,height)
int x,y;
{
  Window openWindow();
  int ButtonW = 1;
  int NoneButtonW =0;

/*   theDCTextWindow = openWindow(x, y,
                                  width,height,
                                  NORMAL_WINDOW,
                                  "Configuration",
                                  NORMAL_STATE,
                                  theDCWindow,
                                  &theDCTextGC,NoneButtonW);

   associateFont(theDCTextGC, TEXT1_FONT);
   initEvents(theDCTextWindow,IN_PALETTE);
*/
  x = DISTANCE;
  y = DISTANCE;

  theTX1Window = openWindow(x,y,
                            BUTTON_LEVEL3_WIDTH,
                            BUTTON_LEVEL3_HEIGHT,
                            NORMAL_WINDOW,
                            "Set/Change QoS",
                            NORMAL_STATE,
                            theDCWindow,
                            &theTX1GC,
                            ButtonW);

  initEvents(theTX1Window, IN_PALETTE);

  XDrawString(display,theDCWindow, theDCGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Configure QoS Parameters at Master Side",
              strlen("Configure QoS Parameters at Master Side"));

  y +=(BUTTON_LEVEL3_HEIGHT + DISTANCE);

  theTX2Window = openWindow(x,y,
                            BUTTON_LEVEL3_WIDTH,
                            BUTTON_LEVEL3_HEIGHT,
                            NORMAL_WINDOW,
                            "Set/Change QoS",
                            NORMAL_STATE,
                            theDCWindow,
                            &theTX2GC,
                            ButtonW);

  initEvents(theTX2Window, IN_PALETTE);

  XDrawString(display,theDCWindow, theDCGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Configure QoS Parameters at Slave Side",
              strlen("Configure QoS Parameters at Slave Side"));

  y +=(BUTTON_LEVEL3_HEIGHT + DISTANCE);

  theTX3Window = openWindow(x,y,
                            BUTTON_LEVEL3_WIDTH,
                            BUTTON_LEVEL3_HEIGHT,
                            NORMAL_WINDOW,
                            "Set/Change QoS",
                            NORMAL_STATE,
                            theDCWindow,
                            &theTX3GC,
                            ButtonW);

  initEvents(theTX3Window, IN_PALETTE);

  XDrawString(display,theDCWindow, theDCGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Change/Show QoS Parameters at Master Side",
              strlen("Change/Show QoS Parameters at Master Side"));


  y +=(BUTTON_LEVEL3_HEIGHT + DISTANCE);

  theTX4Window = openWindow(x,y,
                            BUTTON_LEVEL3_WIDTH,
                            BUTTON_LEVEL3_HEIGHT,
                            NORMAL_WINDOW,
                            "Set/Change QoS",
                            NORMAL_STATE,
                            theDCWindow,
                            &theTX4GC,
                            ButtonW);

  initEvents(theTX4Window, IN_PALETTE);


  XDrawString(display,theDCWindow, theDCGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Change/Show QoS Parameters at Slave Side",
              strlen("Change/Show QoS Parameters at Slave Side"));

   XDefineCursor(display,theTX1Window, theButtonCursor);
   XDefineCursor(display,theTX2Window, theButtonCursor);
   XDefineCursor(display,theTX3Window, theButtonCursor);
   XDefineCursor(display,theTX4Window, theButtonCursor);

}/* -- initTextWindows */



/*
** displayDialog
*/

displayDialog(window)
Window  window;
{
```

```
  int xt,yt;
  int y,width,height;

  y = textHeight(TEXT1_FONT) + 2;


  if (window == theDCCancelWindow)
    {
     XDrawString(display, window, theDCCancelGC,
                 10,y,
                 "Quit",
                 strlen("Quit"));
    }

  if (window == theDCWindow)
    {

     xt = DISTANCE;
     yt = DISTANCE;

     width = MAIN_WINDOW_WIDTH - 4*DISTANCE;
    height = MAIN_WINDOW_HEIGHT - 100 -5*DISTANCE;

    initTextWindows(xt,yt,width,height);
    }

  if (window ==theTX1Window)
    {
     XClearWindow(display,theDCWindow);
     freeDCText();
     xt = DISTANCE;
     yt = DISTANCE;

     setColorWithName(theDCGC, "blue");
     XDrawString(display, theDCWindow,theDCGC,
                 DISTANCE,DISTANCE-5,
                 "Configure QoS Parameters at Master Side",
                 strlen("Configure QoS Parameters at Master Side"));

     QoSsetUp(xt,yt,MASTER);
    }

  if (window == theTX2Window)
    {
     XClearWindow(display,theDCWindow);
     freeDCText();
     xt = DISTANCE;
     yt = DISTANCE;

     setColorWithName(theDCGC, "blue");
     XDrawString(display, theDCWindow,theDCGC,
                 DISTANCE,DISTANCE-5,
                 "Configure QoS Parameters at Slave Side",
                 strlen("Configure QoS Parameters at Slave Side"));

    QoSsetUp(xt,yt,SLAVE);
   }

  if (window ==theTX3Window)
     {
     XClearWindow(display,theDCWindow);
     freeDCText();
     xt = DISTANCE;
     yt = DISTANCE;

     setColorWithName(theDCGC, "blue");
     XDrawString(display, theDCWindow,theDCGC,
                 DISTANCE,DISTANCE-5,
                 "Change/Show QoS Parameters at Master Side",
                 strlen("Change/Show QoS Parameters at Master Side"));

     QoSsetUp(xt,yt,MASTER);
    }

    if (window == theTX4Window)
    {
      XClearWindow(display,theDCWindow);
      freeDCText();
      xt = DISTANCE;
      yt = DISTANCE;

      setColorWithName(theDCGC, "blue");
      XDrawString(display, theDCWindow,theDCGC,
                  DISTANCE,DISTANCE-5,
                  "Change/Set QoS Parameters at Slave Side",
                  strlen("Change/Set QoS Parameters at Slave Side"));

      QoSsetUp(xt,yt,SLAVE);
    }
  XFlush(display);
}/* -- function displayDialog */

/*
** dialogEventLoop handles all the dialog events
*/

dialogEventLoop()

{
  int   status = (-1);
  XEvent    event;

  XNextEvent(display, &event);

  switch(event.type)
    {
    case ConfigureNotify:
    case Expose:
    case MapNotify:
      displayDialog(event.xany.window);
      break;
    case ButtonPress:
      if (event.xbutton.window == theDCCancelWindow)
        {
          highlightChoice(theDCCancelWindow,"blue",BUTTON_LEVEL2_WIDTH,
                          BUTTON_LEVEL2_HEIGHT);

          displayDialog(event.xbutton.window);
          status = 0;
        }

      if (event.xbutton.window == theTX1Window)
        {
          menu_state = CONFIGURE;
          highlightChoice(theTX1Window,"blue",BUTTON_LEVEL3_WIDTH,
                          BUTTON_LEVEL3_HEIGHT);
          displayDialog(event.xbutton.window);
          XClearWindow(display,theDCWindow);
```

```
          setColorWithName(theDCGC,"black");
          displayDialog(theDCWindow);
        }

      if (event.xbutton.window ==theTX2Window)
        {
          menu_state = CONFIGURE;
          highlightChoice(theTX2Window,"blue",BUTTON_LEVEL3_WIDTH,
                          BUTTON_LEVEL3_HEIGHT);
          displayDialog(event.xbutton.window);
          XClearWindow(display,theDCWindow);
          setColorWithName(theDCGC,"black");
          displayDialog(theDCWindow);
        }

      if (event.xbutton.window == theTX3Window)
        {
          menu_state = CHANGE_SHOW;

          highlightChoice(theTX3Window,"blue",BUTTON_LEVEL3_WIDTH,
                          BUTTON_LEVEL3_HEIGHT);
          displayDialog(event.xbutton.window);
          XClearWindow(display,theDCWindow);
          setColorWithName(theDCGC,"black");
          displayDialog(theDCWindow);
        }

      if (event.xbutton.window == theTX4Window)
        {
          menu_state = CHANGE_SHOW;
          highlightChoice(theTX4Window,"blue",BUTTON_LEVEL3_WIDTH,
                          BUTTON_LEVEL3_HEIGHT);
          displayDialog(event.xbutton.window);
          XClearWindow(display,theDCWindow);
          setColorWithName(theDCGC,"black");
          displayDialog(theDCWindow);
        }

      break;
    case KeyPress:
      dialogKeyPress(&event);
      break;
    }
  return(status);
} /* -- function dialogEventLoop */


/*
** dialogKeyPress handles keybord input inot the stringDialog
*/

dialogKeyPress (event)
XKeyEvent   *event;
{
  int   length, l,i;
  int   theKeyBufferMaxLen = 64;
  int   theKeyBuffer[65];
  KeySym  theKeySym;
  XComposeStatus   theComposeStatus;

   for (i=0; i<65; i++)
   theKeyBuffer[i] =0;
  length = XLookupString(event,
                         theKeyBuffer,
                         theKeyBufferMaxLen,
                         &theKeySym,
                         &theComposeStatus);

  printf("KeyBuffer is %s \n", theKeyBuffer);

  l = strlen(theDialogText);

  if ((theKeySym >= ' ') &&
      (theKeySym <= '~') &&
      (length > 0))
    {
      if ((l+strlen(theKeyBuffer)) < MAX_TEXT_LENGTH)
        {
          strcat(theDialogText, theKeyBuffer);
          displayDialog(theDCWindow);
        }
    }
  else
    {
      switch(theKeySym)
        {
        case XK_BackSpace:
        case XK_Delete:
          if (l>=1)
            {
              XClearWindow(display, theDCWindow);
              l--;
              theDialogText[l] = '\0';
              displayDialog(theDCWindow);
              XFlush(display);
            }
          break;
        default:;
        }
    }
}/* -- function dialogkeyPress */



/*
** freeDialog
*/

freeDialog()
{
  XFreeGC(display, theDCGC);
  XFreeGC(display, theDCCancelGC);
  XFreeGC(display, theTX1GC);
  XFreeGC(display, theTX2GC);
  XFreeGC(display, theTX3GC);
  XFreeGC(display, theTX4GC);


/*
** Destroy all windows
*/

  XDestroySubwindows(display, theDCWindow);
  XDestroyWindow(display, theDCWindow);
  XFlush(display);
}/* -- function freeDialog */
```

```
/*
** freeDCText
*/

freeDCText()
{
 XFreeGC(display,theTX1GC);
 XFreeGC(display,theTX2GC);
 XFreeGC(display,theTX3GC);
 XFreeGC(display,theTX4GC);

 XDestroyWindow(display,theTX1Window);
 XDestroyWindow(display,theTX2Window);
 XDestroyWindow(display,theTX3Window);
 XDestroyWindow(display,theTX4Window);

}
refreshDCText(x,y)
int x,y;
{


  XDrawString(display,theDCWindow, theDCGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Configure QoS Parameters at Master Side",
              strlen("Configure QoS Parameters at Master Side"));


  y +=(BUTTON_LEVEL3_HEIGHT + DISTANCE);

  XDrawString(display,theDCWindow, theDCGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Configure QoS Parameters at Slave Side",
              strlen("Configure QoS Parameters at Slave Side"));

  y +=(BUTTON_LEVEL3_HEIGHT + DISTANCE);

  XDrawString(display,theDCWindow, theDCGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Change/Show QoS Parameters at Master Side",
              strlen("Change/Show QoS Parameters at Master Side"));

  y +=(BUTTON_LEVEL3_HEIGHT + DISTANCE);

  XDrawString(display,theDCWindow, theDCGC,
              x + BUTTON_LEVEL3_WIDTH + DISTANCE,
              y + BUTTON_LEVEL3_HEIGHT,
              "Change/Show QoS Parameters at Slave Side",
              strlen("Change/Show QoS Parameters at Slave Side"));


}
```

```
/*************************************************************************/
/* Filename : dialogUtilities.c                                          */
/* Purpose  : Help Functions used in dialog procedures                   */
/* Author   : Klara Nahrstedt                                            */
/* Update   : 07/01/94                                                   */
/*************************************************************************/

#include "/home/klara/tele.d/include.d/defs.h"
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include "/home/klara/tele.d/include.d/qos.h"
#include "/home/klara/tele.d/include.d/retta.h"
extern Display *display;
extern Window theTeleroboticsWindow;
extern Cursor theErrorCursor;

Window theErrorWindow;
GC     theErrorGC;

int getintVideoRate(charRate)
char charRate[2];
{
  return(atoi(charRate));
}

int getintRobotRate(charRate)
char charRate[4];
{
  return(atoi(charRate));
}

int getcharVideoRate(intRate,charRate)
int intRate;
char charRate[2];
{
  if (intRate ==0)
    strcpy(charRate,"0");
  if (intRate==60)
    strcpy(charRate,"60");
  if (intRate==2)
    strcpy(charRate,"2");
  if (intRate==3)
    strcpy(charRate,"3");
  if (intRate==4)
    strcpy(charRate,"4");
  if (intRate==5)
    strcpy(charRate,"5");
  if (intRate==6)
    strcpy(charRate,"6");
  if (intRate==10)
   strcpy(charRate,"10");
  if (intRate ==12)
    strcpy(charRate,"12");
  if (intRate==15)
    strcpy(charRate,"15");
  if (intRate == 20)
    strcpy(charRate,"20");
  if (intRate == 30)
    strcpy(charRate,"30");
  if (intRate == 120)
    strcpy(charRate,"120");
  if (intRate==180)
    strcpy(charRate,"180");
  if (intRate==240)
    strcpy(charRate,"240");
  if (intRate==300)
    strcpy(charRate,"300");


}

int getcharYesNo(intValue,charValue)
int intValue;
char charValue[4];
{
  if (intValue == TRUE)
    strcpy(charValue, "yes");
  if (intValue == FALSE)
    strcpy(charValue,"no");
}
int getcharRobotRate(intRate,charRate)
int intRate;
char charRate[4];
{

  if (intRate == 50)
    strcpy(charRate,"50");
  if (intRate == 100)
    strcpy(charRate,"100");
  if (intRate == 150)
    strcpy(charRate,"150");
  if (intRate == 200)
    strcpy(charRate,"200");
  if (intRate == 250)
    strcpy(charRate,"250");
  if (intRate == 300)
    strcpy(charRate,"300");
  if (intRate == 350)
    strcpy(charRate,"350");
  if (intRate == 400)
    strcpy(charRate,"400");
  if (intRate == 450)
    strcpy(charRate,"450");
  if (intRate == 500)
    strcpy(charRate,"500");

}
ShowError(error,type,param)
int error;
int type;
int param;
{
  int x,y,yh;
  int NoneButtonW = 0;


  x = MAIN_WINDOW_WIDTH - ERROR_WINDOW_WIDTH - 10;
  y = ERROR_WINDOW_HEIGHT + 30;
  theErrorWindow = openWindow(x,y,
                              ERROR_WINDOW_WIDTH,
                              ERROR_WINDOW_HEIGHT,
                              NORMAL_WINDOW,
                              "Error Window",
                              NORMAL_STATE,
                              theTeleroboticsWindow,
                              &theErrorGC,
                              NoneButtonW);
  associateFont(theErrorGC,TEXT1_FONT);
```

```
initEvents(theErrorWindow,IN_PALETTE);
XDefineCursor(display,theErrorWindow,theErrorCursor);
XClearWindow(display,theErrorWindow);
yh=ERROR_WINDOW_HEIGHT - 5;

switch(type)
  {
  case VIDEO:
    switch(param)
      {
      case S_RATE:
        switch(error)
          {
          case BAD_MAXBOUND:
            XDrawString(display,theErrorWindow,
                        theErrorGC,
                        5,yh,
                        "Rate:No Support (Max.Rate=300/minute)",
                        strlen("Rate:No Support (Max.Rate=300/minute)"));
            break;
          case BAD_VALUE:
            XDrawString(display,theErrorWindow,
                        theErrorGC,
                        5,yh,
                        "Rate:Bad Value (Multiple/Divisible of 60)",
                        strlen("Rate:Bad Value (Multiple/Divisible of 60)"));
            break;
            default :
              break;
          }
        break;
      case  S_COMPRESS:
        switch(error)
          {
          case NOT_SUPPORTED:
             XDrawString(display,theErrorWindow,
                        theErrorGC,
                        5,yh,
                        "Compression: No Support",
                        strlen("Compression: No Support"));
           }
        break;
      case S_TYPE:
        switch(error)
          {
          case NOT_SUPPORTED:
             XDrawString(display,theErrorWindow,
                        theErrorGC,
                        5,yh,
                        "Video: No Support",
                        strlen("Video: No Support"));
           }
        break;
      }
    break;

  case ROBOT:
    switch(param)
      {
      case S_RATE:
        switch(error)
          {
          case BAD_MAXBOUND:
            XDrawString(display,theErro
                        theErrorGC,
                        5,yh,
                        "Rate:No Support (Max.Rate=500)",
                        strlen("Rate:No Support (Max.Rate=500)"));
            break;
          case BAD_MINBOUND:
            XDrawString(display,theErrorWindow,
                        theErrorGC,
                        5,yh,
                        "Rate:No Support (Min.Rate=10)",
                        strlen("Rate:No Support (Min.Rate=10)"));
            break;
          case BAD_VALUE:
            XDrawString(display,theErrorWindow,
                        theErrorGC,
                        5,yh,
                        "Rate:Bad Value (Multiple of 10)",
                        strlen("Rate:Bad Value (Multiple of 10)"));
          }
        break;
      }
    break;
  case AUDIO:
    XDrawString(display,theErrorWindow,
                theErrorGC,
                5,yh,
                "Audio: No Support",
                strlen("Audio: No Support"));
    break;
  case OTHER: /* Errors with Relations Specification */
    switch(param)
      {
      case R_SYNCH:
        switch(error)
          {
          case NOT_SUPPORTED:
            XDrawString(display,theErrorWindow,
                        theErrorGC,
                        5,yh,
                        "Media Synchronization: No Support",
                        strlen("Media Synchronization: No Support"));
            break;
          }
        break;
      case R_INTEG:
        switch(error)
          {
          case NOT_SUPPORTED:
            XDrawString(display,theErrorWindow,
                        theErrorGC,
                        5,yh,
                        "Integration: No Support",
                        strlen("Integration: No Support"));
            break;
          }
        break;
      case R_CONV:
        switch(error)
          {
          case NOT_SUPPORTED:
            XDrawString(display,theErrorWindow,
                        theErrorGC,
                        5,yh,
                        "Conversion: No Suppo
```

```
                        strlen("Conversion: No Support"));
              break;
            }
          break;
        case R_COMM:
          switch(error)
            {
            case NOT_SUPPORTED:
              XDrawString(display,theErrorWindow,
                        theErrorGC,
                        5,yh,
                        "Communication: No Support",
                        strlen("Communication: No Support"));
              break;
            }
          break;
        }
      break;
    }
}
freeError()
{
  XDestroyWindow(display,theErrorWindow);
  XFreeGC(display,theErrorGC);
}
```

```
/*****************************************************************************
*
*  Filename:   bit3_perror.c
*
*      Purpose:    Routine to check the status of the Bit 3 device driver
*                  and print an error message to 'stderr' if there were
*                  any status errors.
*
*      Copyright (c) 1990, 1991 by Bit 3 Computer Corporation.
*      All rights reserved.
*
*****************************************************************************/

static char *revcntrl = "$Revision:   1.4  $";

#include        <stdio.h>
#include        <errno.h>
#include        <sys/btio.h>
#include        "/pkg/bit3/921/v1.6/sys/btlio.h"
#include        "/pkg/bit3/921/v1.6/sys/btio.h"

/*****************************************************************************
*
*      Function:       bit3_perror()
*
*      Purpose:        Check the local status register for errors, print
*                      message to stderr if any errors are detected.
*
*      Args:           chan            File channel device was opened on.
*
*      Returns:        0               No errors.
*                      1               If status errors.
*
*****************************************************************************/

int     bit3_perror(chan)
int     chan;
{
    bt_status_t         data;

    if (ioctl(chan, BIOC_STATUS, &data)) {
        printf("BIOC_STATUS returned errno = %d.\n", errno);
        return(1);
    }

    if (data &= BT_STATUS_MASK) {
        fprintf(stderr,"Status error 0x%2.2x:\n", data>>BT_INTR_ERR_SHFT);
        if (data & BT_INTR_POWER) {
            fprintf(stderr,"\tRemote chassis off or cable disconnected.\n");
        } else {
            if (data & BT_INTR_PARITY)
                fprintf(stderr,"\tInterface parity error.\n");
            if (data & BT_INTR_REMBUS)
                fprintf(stderr,"\tRemote bus error.\n");
            if (data & BT_INTR_TIMEOUT)
                fprintf(stderr,"\tInterface timeout.\n");
        }
        return (1);
    }
    return (0);
}                               /* end of bit3_perror() */
```

```
/*****************************************************************************
*
*      Filename:       bt_devname.c
*
*      Purpose:        Routine to build the requested device name and to figure
*              the correct address to use for the apporpriate device type.
*
*  Arguments:
*      int unit        - base unit number to open.
*      u_long *addr     - pointer to address value. ( value may be modified)
*      int axstype     - requested base access type BT_AXSDP, BT_AXSRR, etc.
*      char *devname   - Device name to open. "/dev/bti" "/dev/bte"
*
*  Returns:
*      NULL if failure. PTR if success.
*
*      Copyright (c) 1990, 1991 by Bit 3 Computer Corporation.
*      All rights reserved.
*
*****************************************************************************/

static char *revcntrl = "$Revision:   1.8  $";

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/file.h>

#include <sys/btio.h>

static char devstrng[FILENAME_MAX];

/* bt device driver name */

char *bt_devname(unit, addr, axstype, devname)
int unit; u_long *addr; int axstype; char *devname;
{
    /****  Figure the address needed to provide the access desired      */

    if (addr != NULL) {
        if (*addr > (u_long) INT_MAX) {
            if ((axstype == BT_AXSRR) || (axstype == BT_AXSRE)) {
                axstype = BT_AXSRE;
                *addr -= ((u_long) INT_MAX) +1;
            } else {
                return ((char *) NULL);
            }
        } else {
            if (axstype == BT_AXSRE)
                axstype = BT_AXSRR;
        }
    }
    printf(" allocated address %x \n",*addr);

    /****   Build the name based on parameters issued.                  */

    unit += (axstype << BT_AXS_SHFT);

    sprintf(devstrng, "%s%d", devname, unit);
    printf("device name %s \n",devstrng);
    return(devstrng);
}
```

```
/*****************************************************************/
/* Filename: rtnp.c                                              */
/* Purpose : entites which support real-time transport/network   */
/*           protocol                                            */
/* Author :  Klara Nahrstedt                                     */
/* Update :  2/3/95                                             */
/*****************************************************************/

#include "/home/klara/tele.d/include.d/defs.h"
#include "/home/klara/tele.d/include.d/comm.h" /* network constants and structures */
#include "/home/klara/tele.d/include.d/atm_dd.h"



/*****************************************************************/
/* Module : init_send_atm                                        */
/* Purpose : open atm host interface for sending data            */
/* Input :  vci identifier                                       */
/* Output : connection identifier (file dscriptor) of the device*/
/*          this connection identifier is used to send data      */
/*****************************************************************/

init_send_atm(vci,connection_id,count,mode)
int vci;
int *connection_id;
int count;
int mode;
{
/************** atm parameters ******************************/

  char *dev_name;
  struct session_blk start_data;
  int conid;
  struct status_block sb;
  int i;
  int mid;
  dev_name = DEFAULT_DEVICE;
  if ((conid=open(dev_name,O_WRONLY)) == -1)
    {
      fprintf(stderr, "Device %s:",dev_name);
      perror("Couldn't open");
      exit(-1);
    }
  bzero(&start_data,sizeof(start_data));
  start_data.status = CIO_OK;
  if (ioctl(conid,CIO_START,&start_data) < 0)
    {
      perror("CIO_START");
      exit(1);
    }
  sb.code = CIO_NULL_BLK;
  while(sb.code != CIO_START_DONE)
    {
      if (ioctl(conid,CIO_GET_STAT, &sb) < 0)
        {
          perror("CIO_GET_STAT");
          exit(1);
        }
      if (sb.code!=CIO_START_DONE && sb.code != CIO_NULL_BLK)
        printf("expected status block: code = %d\n", sb.code);
    }

    ioctl(conid,SPECIFY_OUT_VCI,vci);
/* check if cell mode needs mid = 0 */
/* mid =1 is for datagrams */

  if (mode == DATAGRAM_MODE)
    {
      mid = 1;
      ioctl(conid,SPECIFY_OUT_MID,mid);
    }
  if (mode == CELL_MODE)
    {
      mid =0;
      ioctl(conid,SPECIFY_OUT_MID,mid);
    }

  ioctl(conid,SPECIFY_OTHER,DEFAULT_OTHER);

  *connection_id = conid;
  printf("init_send_atm: <vci,mid,conid>=<%x, %d,%d> \n",vci,mid,conid);

/*************** control the count parameter **********************/

      if (count > MOBY_SIZE)
        {
          fprintf(stderr,"Count too big: reducing to %d bytes. \n", MOBY_SIZE);
          count=MOBY_SIZE;
        }
      if (count % 4)
        {
          count &=0xfffffffc;
          fprintf(stderr,"Count should be word aligned; reducing to %d \n",count);
        }
      if (count % 44 == 4 && vci & 0x4000)
        {
          count -=4;
          fprintf(stderr,"Count cannot be (n*44)+4; reducing to %d \n",
                  count);
        }
}

/*****************************************************************/
/* Module: init_recv_atm()                                       */
/* Purpose : open atm device driver for receiving data           */
/* Input: vci identifier                                         */
/* Output: connection identifier (file descriptor of the device) */
/*****************************************************************/



init_recv_atm(vci,connection_id)
int vci;
int *connection_id;
{

/************** atm parameters ******************************/

  char *dev_name;
  struct session_blk start_data;
  int conid;
  struct status_block sb;

  dev_name = DEFAULT_DEVICE;
  if ((conid=open(dev_name,O_RDONLY)) == -1)
    {
      fprintf(stderr, "Device %s:",dev_name);
      perror("Couldn't open");
```

```
      exit(-1);
    }
  bzero(&start_data,sizeof(start_data));
  start_data.status = CIO_OK;
  start_data.netid = 4;
  if (ioctl(conid,CIO_START,&start_data) < 0)
    {
      perror("CIO_START");
      exit(1);
    }
  sb.code = CIO_NULL_BLK;
  while(sb.code != CIO_START_DONE)
    {
      if (ioctl(conid,CIO_GET_STAT, &sb) < 0)
        {
          perror("CIO_GET_STAT");
          exit(1);
        }
      if (sb.code!=CIO_START_DONE && sb.code != CIO_NULL_BLK)
        printf("expected status block: code = %d\n", sb.code);
    }

    ioctl(conid,SPECIFY_IN_VCI,vci);

  ioctl(conid,SPECIFY_IN_MID,1);
  *connection_id = conid;
  printf("init_recv_atm: <vci,conid> = <%x,%d> \n",vci,conid);

}

int connect_s(vci,conid,mode,size)
int vci;
int *conid;
int mode;
int size;
{

  int count;

  if (mode == CELL_MODE)
    {
      count = CELL_SIZE;
    }
  if (mode == DATAGRAM_MODE)
    {
      count = size;
    }

  init_send_atm(vci,conid,count,mode);

}

int connect_r(vci,conid)
int vci;
int *conid;
{

  init_recv_atm(vci,conid);

}

int send_cell(connid,data,size,err)
int connid;
char *data;
int size;
FEC_FLAGS err;
{
  char buf[48];
  int count=48;

  if (size > CELL_SIZE)
    {
      return(WRONG_SIZE);
    }
  else
    {
      bcopy(data,buf,size);
      write(connid,buf,count);
/* send a copy over the additional connection - FEC */
      if (err.err_flag == TRUE)
        write(err.connid,buf,count);
    }

}

int recv_cell(connid,data,size,err)
int connid;
char *data;
int size;
FEC_FLAGS err;
{
  char buf[48];
  int count=48;

  int read_ORIGIN_OK;
  int read_COPY_OK;

  read_ORIGIN_OK=read(connid,buf,count);

/************ check for read problems on ATM interface ********/

  if (read_ORIGIN_OK <= 0)
    {
      perror("read problems ");
/********** read 0 bytes (non-blocking I/O and data were no read
  to be read                                          **************/

      while (errno == EWOULDBLOCK)
        {
          read_ORIGIN_OK=read(connid,buf,count);
        }
      if (read_ORIGIN_OK <=0)
        read_ORIGIN_OK = FALSE;
    }
  if (read_ORIGIN_OK > 0)  /* data were read  O.K. */
    {
      bcopy(buf,data,size);
    }
/******** if FEC wanted get the copy packet ****************/

    if (err.err_flag == TRUE)
    {
      read_COPY_OK=read(err.connid,buf,count);
      if (read_ORIGIN_OK==FALSE && read_COPY_OK > 0)
        {
          bcopy(buf,data,size);
          return(OK);
        }
```

```
      if(read_ORIGIN_OK == FALSE && read_COPY_OK <=0)
        {
          while (errno == EWOULDBLOCK)
            {
              read_COPY_OK=read(err.connid,buf,count);
            }
          if (read_COPY_OK <=0)
            {
              read_COPY_OK = FALSE;
              return(ERROR_DATAGRAM);
            }
          else
            {
              bcopy(buf,data,size);
              return(OK);
            }
        }

    }
    else  /* notification that receive of datagram is errorful */
      {
        if (read_ORIGIN_OK == FALSE)
          return(ERROR_DATAGRAM);
        else
          return(OK);
      }
}

int send_pkt(connid,data,size,err)
int connid;
char *data;
int size;
FEC_FLAGS err;
{

/************* send original packet *********************************/

  if (size > DATAGRAM_SIZE)
    {
      return(WRONG_SIZE);
    }
  else
    {
      write(connid,data,size);

/************* if Forward Error Correction is wanted, i.e. err_flag=TRUE */
      if (err.err_flag == TRUE)
        write(err.connid,data,size);
    }
}


int recv_pkt(connid,data,size,err)
int connid;
char *data;
int size;
FEC_FLAGS err;
{
  char buf[MOBY_SIZE];
  int count=MOBY_SIZE;
  int read_ORIGIN_OK;
  int read_COPY_OK;

  read_ORIGIN_OK=read(connid,data,size);

/*********** check for read problems on ATM interface ********/

  if (read_ORIGIN_OK <= 0)
    {
      perror("read problems ");
/********** read 0 bytes (non-blocking I/O and data were no read
  to be read                                          **************/

      while (errno == EWOULDBLOCK)
        {
          read_ORIGIN_OK=read(connid,buf,count);
        }
      if (read_ORIGIN_OK <=0)
        read_ORIGIN_OK = FALSE;
    }
/*
  if (read_ORIGIN_OK > 0)
    {
      bcopy(buf,data,size);
    }
*/
/******** if FEC wanted get the copy packet *****************/
/*
    if (err.err_flag == TRUE)
    {
      read_COPY_OK=read(err.connid,buf,count);
      if (read_ORIGIN_OK==FALSE && read_COPY_OK > 0)
        {
          bcopy(buf,data,size);
          return(OK);
        }
      if (read_ORIGIN_OK==FALSE && read_COPY_OK <=0)
        {
          while (errno == EWOULDBLOCK)
            {
              read_COPY_OK=read(err.connid,buf,count);
            }
          if (read_COPY_OK <=0)
              return(ERROR_DATAGRAM);
          else
            {
              bcopy(buf,data,size);
              return(OK);
            }
        }
    }
    else */ /* notification that receive of datagram is errorful */
/*
      {
        if (read_ORIGIN_OK == FALSE)
          return(ERROR_DATAGRAM);
        else
          return(OK);
      }
*/
}
```

```
/*************************************************************************/
/* Filename : connectionManagement.c                                     */
/* Purpose : Functions for connection management                         */
/* Author : Klara Nahrstedt                                              */
/* Update : 06/30/95                                                     */
/*************************************************************************/

#include "/home/klara/tele.d/include.d/defs.h"
#include "/home/klara/tele.d/include.d/retta.h"
#include "/home/klara/tele.d/include.d/comm.h"
#include "/home/klara/tele.d/include.d/systemQoS.h"

int connSetup(inout)
int inout;
{
  int i,k;
  NET_QOS_TABLE NTable;
  APP_QOS AParam;
/************************ IF NOTIFICATION SUCCESSFUL- START CONNECT ********/
  getNetQoS(&NTable,inout);
  getAppQoS(&AParam,inout);

  for (i=1; i< MEDIA_NUMBER; i++)
    {
      for (k=0;k < CONNECTION_NUMBER; k++)
        {
          switch(inout)
            {
            case INPUT:
              if (NTable.status[k] == TAKEN &&
                  NTable.medium[k] == AParam.stream[i].type)
                {
                  connect_s(NTable.vci[k],
                            &AParam.stream[i].medium.net_spec.conid[k],NTable.connection
[k].load.id,
                            AParam.stream[i].medium.app_spec.sample_size);
                  printf(" connect_s:<vci[k],conid[k],k,inout> = <%x, %d, %d, %d> \n",
                         NTable.vci[k],
                         AParam.stream[i].medium.net_spec.conid[k],
                         k, inout);
                }
              else
                {
                  k=CONNECTION_NUMBER;
                }
              break;
            case OUTPUT:
              if (NTable.status[k] == TAKEN &&
                  NTable.medium[k] == AParam.stream[i].type)
                {
                  connect_r(NTable.vci[k],
                            &AParam.stream[i].medium.net_spec.conid[k]);
                  printf(" connect_r:<vci[k],conid[k],k,inout> = <%x, %d, %d, %d> \n",
                         NTable.vci[k],
                         AParam.stream[i].medium.net_spec.conid[k],
                         k, inout);
                }
              else
                {
                  k=CONNECTION_NUMBER;
                }
              break;
            }
        }
                }
              setAppQoS(&AParam,inout);
            }
```

```
/*************************************************************************/
/* Filename: mmInstall.c                                                 */
/* Purpose : Install multimedia devices                                  */
/* Author  : Klara Nahrstedt                                             */
/* Update  : 07/01/94                                                    */
/*************************************************************************/

#include "/home/klara/tele.d/include.d/defs.h"
#include "/home/klara/tele.d/include.d/qos.h"
#include "/home/klara/tele.d/include.d/retta.h"

installDevices(dev,side,inout)
MM_DEVICES *dev;
int side;
int inout;
{

  switch(side)
    {
    case MASTER:
      switch(inout)
        {
        case INPUT:
          dev->dev_support[AUDIO].type_of_data = AUDIO;
          dev->dev_support[AUDIO].device_descr=  MICROPHONE;
          dev->dev_support[AUDIO].support = FALSE;
          dev->dev_support[VIDEO].type_of_data = VIDEO;
          dev->dev_support[VIDEO].device_descr = CAMERA;
          dev->dev_support[VIDEO].support = FALSE;
          dev->dev_support[ROBOT].type_of_data = ROBOT;
          dev->dev_support[ROBOT].device_descr=ROBOT_HAND;
          dev->dev_support[ROBOT].support = TRUE;
          dev->spec[ROBOT].quality = HIGH;
          dev->spec[ROBOT].sample_size = 64;
          dev->spec[ROBOT].sample_rate =  50; /* max.50 samples per second */
          dev->spec[ROBOT].comp_spec.name=NONE;
          dev->spec[ROBOT].comp_spec.ratio= 0;
          break;
        case OUTPUT:
          dev->dev_support[AUDIO].type_of_data = AUDIO;
          dev->dev_support[AUDIO].device_descr=  SPEAKER;
          dev->dev_support[AUDIO].support = FALSE;
          dev->dev_support[VIDEO].type_of_data = VIDEO;
          dev->dev_support[VIDEO].device_descr = SCREEN;
          dev->dev_support[VIDEO].support = TRUE;
          dev->spec[VIDEO].quality = MOTION_VIDEO;
          dev->spec[VIDEO].sample_size = 38400; /* frame size, widthxheight*/
          dev->spec[VIDEO].sample_rate = 300; /* max.frames /minute */
          dev->spec[VIDEO].comp_spec.name = NONE;
          dev->spec[VIDEO].comp_spec.ratio = NONE;
          dev->dev_support[ROBOT].type_of_data = ROBOT;
          dev->dev_support[ROBOT].device_descr=ROBOT_HAND;
          dev->dev_support[ROBOT].support = TRUE;
          dev->spec[ROBOT].quality = HIGH;
          dev->spec[ROBOT].sample_size = 64;
          dev->spec[ROBOT].sample_rate =  50; /* max.50 samples per second */
          dev->spec[ROBOT].comp_spec.name=NONE;
          dev->spec[ROBOT].comp_spec.ratio= NONE;
          break;
        }
      break;
    case SLAVE:
      switch(inout)
        {
        case INPUT:
          dev->dev_support[AUDIO].type_of_data = AUDIO;
          dev->dev_support[AUDIO].device_descr=  MICROPHONE;
          dev->dev_support[AUDIO].support = FALSE;
          dev->dev_support[VIDEO].type_of_data = VIDEO;
          dev->dev_support[VIDEO].device_descr = CAMERA;
          dev->dev_support[VIDEO].support = TRUE;
          dev->spec[VIDEO].quality = MOTION_VIDEO;
          dev->spec[VIDEO].sample_size = 38400; /* frame size, widthxheight*/
          dev->spec[VIDEO].sample_rate = 300; /* frames /minute */
          dev->spec[VIDEO].comp_spec.name = NONE;
          dev->spec[VIDEO].comp_spec.ratio = NONE;
          dev->dev_support[ROBOT].type_of_data = ROBOT;
          dev->dev_support[ROBOT].device_descr=ROBOT_SIMULATOR;
          dev->dev_support[ROBOT].support = TRUE;
          dev->spec[ROBOT].quality = HIGH;
          dev->spec[ROBOT].sample_size = 64;
          dev->spec[ROBOT].sample_rate = 50; /* max.50 samples per second */
          dev->spec[ROBOT].comp_spec.name=NONE;
          dev->spec[ROBOT].comp_spec.ratio= NONE;
          break;
        case OUTPUT:
          dev->dev_support[AUDIO].type_of_data = AUDIO;
          dev->dev_support[AUDIO].device_descr=  SPEAKER;
          dev->dev_support[AUDIO].support = FALSE;
          dev->dev_support[VIDEO].type_of_data = VIDEO;
          dev->dev_support[VIDEO].device_descr = SCREEN;
          dev->dev_support[VIDEO].support = FALSE;
          dev->dev_support[ROBOT].type_of_data = ROBOT;
          dev->dev_support[ROBOT].device_descr=ROBOT_SIMULATOR;
          dev->dev_support[ROBOT].support = TRUE;
          dev->spec[ROBOT].quality = HIGH;
          dev->spec[ROBOT].sample_size = 64;
          dev->spec[ROBOT].sample_rate = 50; /* max.50 samples per second */
          dev->spec[ROBOT].comp_spec.name=NONE;
          dev->spec[ROBOT].comp_spec.ratio= NONE;
          break;
        }
      break;
    }
}
```

```
/*************************************************************************/
/* Filename: rtap.c                                                      */
/* Purpose : Application Tasks and Supervisory control for the           */
/*           Telemanufacturing Application                               */
/* Author  : Klara Nahrstedt                                             */
/* Update  : 06/07/95                                                    */
/*************************************************************************/


/***********************************************/
/* Omega includes                              */
/***********************************************/

#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <math.h>


#include "/home/klara/tele.d/include.d/comm.h"
#include "/home/klara/tele.d/include.d/robnet.h"


/***********************************************/
/* Telerobotics application variables          */
/***********************************************/

#define LENGTH  sizeof(ROBOT_IO)
#define CIRCULAR_BUFFER_SIZE 100*sizeof(ROBOT_IO)
#define DUMP_SIZE_POSN 1000
int sample_num_posn;
int sample_posns[DUMP_SIZE_POSN];
struct Dump_Posn {  /* posn history of last dump posn sampling period */
int sample_num;
float pos_T6[4][3]; /*[n,o,a,p][x,y,z]*/
};
struct Dump_Posn   dump_posn[DUMP_SIZE_POSN];
FILE *fforce;

/*************************************************************************
 * Function declarations
 *************************************************************************/


/* Function prototypes */

#ifdef PROTO

int     bit3_perror(int chan);
char    *bt_devname(int unit, u_long *addr, int axxstype, char *devname);

#else

int bit3_perror();
char   *bt_devname();

#endif


/* test - main program to read robotics data from trace file */


application(robot_pkt,j)
ROBOT_IO *robot_pkt;
int j;
{

if (j==0)
  ReadTraceData();

robot_pkt->kind=ROBOT;
robot_pkt->seq = 1;
bcopy(dump_posn[j].pos_T6,&robot_pkt->data[0], NUMPOINTS * sizeof(float));
}


ReadTraceData()
{
int i,j;
char header[100];
printf("Reading  trace data file  : \n");
fforce=fopen("force_data","r");


for(j=0; j<20; j++) {

  fgets(header, 100, fforce);
  sscanf(header,"\t %f %f %f %f",&dump_posn[j].pos_T6[0][0],
  &dump_posn[j].pos_T6[1][0],&dump_posn[j].pos_T6[2][0],&dump_posn[j].pos_T6[3][0]);
  fgets(header, 100, fforce);
  sscanf(header,"\t %f %f %f %f",&dump_posn[j].pos_T6[0][1],
  &dump_posn[j].pos_T6[1][1],&dump_posn[j].pos_T6[2][1],&dump_posn[j].pos_T6[3][1]);
  fgets(header, 100, fforce);
  sscanf(header,"\t %f %f %f %f",&dump_posn[j].pos_T6[0][2],
  &dump_posn[j].pos_T6[1][2],&dump_posn[j].pos_T6[2][2],&dump_posn[j].pos_T6[3][2]);

  fgets(header,100,fforce);

}

return(0);
}

ReadSinglePosn(j)
int j;
{

 char header[100];

 if(fgets(header, 100, fforce) == NULL) return(0);
  sscanf(header,"\t %f %f %f %f",&dump_posn[j].pos_T6[0][0],
  &dump_posn[j].pos_T6[1][0],&dump_posn[j].pos_T6[2][0],&dump_posn[j].pos_T6[3][0]);
  fgets(header, 100, fforce);
  sscanf(header,"\t %f %f %f %f",&dump_posn[j].pos_T6[0][1],
  &dump_posn[j].pos_T6[1][1],&dump_posn[j].pos_T6[2][1],&dump_posn[j].pos_T6[3][0]);
  fgets(header, 100, fforce);
  sscanf(header,"\t %f %f %f %f",&dump_posn[j].pos_T6[0][2],
  &dump_posn[j].pos_T6[1][2],&dump_posn[j].pos_T6[2][2],&dump_posn[j].pos_T6[3][2]);

 return(1);

}

/* this routine is for printing the posn data collected at servo rate by dv  */
/*
OpenForceData()
{
```

```
  printf("[JIFFE] Opening force data file (joint servo rate) : \n");
  if((fforce=fopen("force_data","w")) == NULL) {
     printf("Can't open force_data\n");
     return(0);
   }
  return(1);

}
*/
/*
WriteForceData(tw)
TRSF *tw;
{

  fprintf(fforce,"\t %12.6f %12.6f %12.6f %12.6f \n",tw->n.x,
  tw->o.x,tw->a.x,tw->p.x);
  fprintf(fforce,"\t %12.6f %12.6f %12.6f %12.6f \n",tw->n.y,
  tw->o.y,tw->a.y,tw->p.y);
  fprintf(fforce,"\t %12.6f %12.6f %12.6f %12.6f \n\n",tw->n.z,
  tw->o.z,tw->a.z,tw->p.z);
  return(0);
}
*/
```

```
/*********************************************************************/
/* Filename: rtap.c                                                  */
/* Purpose : Joint Schedule for the  Telemanufacturing Application   */
/* Author  : Klara Nahrstedt                                         */
/* Update  : 07/05/95                                                */
/*********************************************************************/


#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <math.h>
#include     <sys/time.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>

/*************** atm needs following   h-files *************************/

#include    <sys/mode.h>
#include    <sys/errno.h>
#include    <sys/devinfo.h>
#include    <sys/comio.h>
#include    <sys/xmem.h>

/*************** bit 3 need following h-files *************************/

#include     <sys/mman.h>
#include     <sys/btio.h>
#include     "/pkg/bit3/921/v1.6/sys/btio.h"
#include     "/pkg/bit3/921/v1.6/sys/btlio.h"



#include "/home/klara/tele.d/include.d/comm.h"
#include "/home/klara/tele.d/include.d/systemQoS.h"
#include "/home/klara/tele.d/include.d/robnet.h"


/***************************************************/
/* UMS includes                                    */
/***************************************************/


#include <UMSObject.h>
#include <UMSClass.h>
#include <UMSVideoIO.h>
#include <UMSStrings.h>

/***************************************************/
/* UMS variables                                   */
/***************************************************/

UMSVideoIO  video_obj;
Environment *ev;
long colormap_size;

/*****************************************************************************
 * X variables
 *****************************************************************************/



 Display         *dpy;
 Window          wid;
 int                     screen;
 GC                      gc;
 int                     depth;
 int                     pixel_pad;
 Visual          *my_visual;
 Status          result;
 XVisualInfo     info;
 XSetWindowAttributes wa;
 long            colormap_size;

/***********************************************/
/* Telerobotics application variables          */
/***********************************************/

#define LENGTH  sizeof(ROBOT_IO)
#define CIRCULAR_BUFFER_SIZE 100*sizeof(ROBOT_IO)
#define DUMP_SIZE_POSN 1000


int sample_num_posn;
int sample_posns[DUMP_SIZE_POSN];
struct Dump_Posn {  /* posn history of last dump posn sampling period */
int sample_num;
float pos_T6[4][3]; /*[n,o,a,p][x,y,z]*/
};
struct Dump_Posn   dump_posn[DUMP_SIZE_POSN];
FILE *fforce;

/*****************************************************************************
 * Function declarations
 *****************************************************************************/


/* Function prototypes */

#ifdef PROTO

int     bit3_perror(int chan);
char    *bt_devname(int unit, u_long *addr, int axxstype, char *devname);

#else

int bit3_perror();
char   *bt_devname();
#endif



int rtap(side)
int side;
{
/********************** bit3 parameters ********************************/

  char     *device = BT_DEVNAME;
  char     *cp;
  int      unit = 0;
  int      type = BT_AXSDP;
  int      my_index;
  int      nlines;
  int      chan;
  int      stat;
  u_long   status_flags;
  u_long   remote_addr_in = 0;
  u_long   remote_addr_out;
```

```
  u_long   remote_in_ptr;

/********************** System Parameters ****************************/
  GLOBAL_STATE SystemState;
  RATE_MONOTONIC_SCHEDULER Scheduler;
  INFO_STATE WhatInfo;
  long runtime,readtime,sendtime,recvtime,writetime,sendcelltime,recvcelltime;
  long clock,readvideo,writevideo;
  long recvrobdata,sendrobdata;
  long diff;
  BOOLEAN first;
  int conid;
  int t; /* index of number of ticks in scheduler */
  int i; /* index  of tasks in a minimum interval */
  int k,s; /* index of connections */
  int j; /* index of application robot data traces */
  int kin_video,kin_robot,kout_video,kout_robot; /* index of connection id */
  int ii; /* index for fragments in an image */
  int m;
  int pri;
  pid_t p_pid;

  char datagram[DATAGRAM_SIZE];
  char cell[CELL_SIZE];
  int size;
  BOOLEAN done;
  int changedDelay;

/********************** time parameters ******************************/

  struct timeval tv1,tv2;
  struct timeval t1read,t2read;
  struct timeval t1recvf,t2recvf;
  struct timeval t1send,t2send;
  struct timeval t1recv,t2recv;
  struct timeval t1readvideo,t2readvideo;
  struct timeval t1writevideo,t2writevideo;
  struct timeval t1write,t2write;
  struct timeval t1sendcell,t2sendcell;
  struct timeval t1recvcell,t2recvcell;
  struct timeval t1ren,t2ren;
  struct timezone tz;
  struct timeval t1neg;
  int MenuControl;
  int MenuStateFD;
  int renegid; /* renegotiation link id*/
  RENEGOT_INFO r_info;

/********************** video variables ***********************/

  unsigned char   *imagedata;
  unsigned char *data;

  XImage                *image;
  long                  width;
  long                  height;
  int                   number,end;
  int                           num_frames;
  char                  ch;
  long                  error_code;
  int rc;
  int image_size;
  double frame_size;
  double frame_rate;
  long each_buffer_size;
  long number_of_elements;
  _IDL_SEQUENCE_UMSVideoIO_RingBufferElement *ring_buffer;
  UMSVideoIO_RingBufferElement *rbuffer1;
  long index;
  long flag;
  unsigned char  * Address[5];

/********************** Application Protocol Variables ****************/
  APP_QOS InputAppParam;
  APP_QOS OutputAppParam;
  ROBOT_IO robot_pkt;
  FEC_FLAGS err;

  p_pid = getpid();
  pri = 0;
  if (setpri(p_pid,pri) <0)
    perror("setpri");

/*************** Defaults ****************************************/

  ev = somGetGlobalEnvironment();

  frame_rate = 15.00;
  colormap_size = 128;
  width = 240;
  height = 160;
  depth = 8;
  changedDelay=0;
  recvrobdata = 0;
  sendrobdata = 0;
  readvideo =0;
  writevideo=0;
  sendtime=0;
  recvtime=0;
  readtime=0;
  writetime = 0;
  first = TRUE;
/*************** Call Management *************************************/

  connSetup(INPUT);
  connSetup(OUTPUT);

/*************** Open Bit3 Device ************************************/

      /* Open the device */
/*
    if ((cp = bt_devname(unit, &remote_addr_in, type, device)) != NULL) {
      fprintf(stderr, "Opening %s.\n", cp);
           if ((chan = open(cp, O_RDWR)) < 0 ) {
             perror("Can't Open Device Driver");
             exit(errno);
           }
    } else {
      perror("Can't Generate File Name\n");
      exit(0);
    }
*/
    /* Initialize the Adaptor */
/*
    (void) ioctl(chan, BIOC_SETUP, &status_flags);
    if (status_flags & BT_STATUS_MASK) {
      bit3_perror(chan);
      fprintf(stderr, "\nCould not initialize Bit 3 Adaptor.\n");
```

```
      exit(1);
    };
*/

    /* Position to remote address at the beginning of VMA/MCA region  */
/*
    if (lseek(chan, remote_addr_in, SEEK_SET) == -1) {
      perror("Error: lseek failed");
      exit(1);
    }
*/
/*************** Retrieve Scheduler and App QoS ***********************/

  WhatInfo.i_set[0]=SystemStateInfo;
  WhatInfo.i_set[1]=ScheduleInfo;

  RetrieveGlobalState(&SystemState,&Scheduler,WhatInfo);
  getAppQoS(&InputAppParam,INPUT);
  getAppQoS(&OutputAppParam,OUTPUT);

  done = FALSE;

  j=0;
  kin_robot=0;
  kin_video = 0;
  kout_video = 0;
  kout_robot=0;

  image_size = width*height;
  if ((imagedata=(unsigned char *)malloc(width*height)) == NULL)
    {
      exit(-1);
    }
/************* get robotics data ************************************/

  application(&robot_pkt,j);



  switch(side)
    {
    case BUYER:
      if (OutputAppParam.stream[VIDEO].type == VIDEO)
        {
          /******************** setup window ***************************/

          /* Create the video IO object */

          video_obj = UMSVideoIONew();




          /* Create the display window */

          rc = create_window(width, height);
          if (rc == 1)
            {
              error_code = 1;
              return(error_code);
            }
        }
      /*****************************************************************/
      /*********************** SET UP RENEGOTIATION LINK ******************/
      connect_s(RENEG_SIGNAL_VCI,&renegid,CELL_MODE,CELL_SIZE);

      break;
    case SELLER:
      if (InputAppParam.stream[VIDEO].type == VIDEO)
        {

          /******************** setup video device ***************************/
          frame_rate= (double)(InputAppParam.stream[VIDEO].medium.app_spec.sample_rate)/6
0;

          /**** Comment - video frame rate is specifed in terms of frames/min
    need to calculate frames/sec *****/


          /************* Create the video IO objcet ***********/

          video_obj = UMSVideoIONew();

          /************* Open the video device ****************/

          rc = UMSVideoIO_open(video_obj,ev,"/dev/sr0");
          if (rc != UMSVideoIO_Success)
            {
              printf("Cannot open video device. rc = %d\n",rc);
              error_code = 1;
              goto Error;
            }

          /********* Set object parameters *****************/

          /************ Set analog input format for capture card */

          rc = UMSVideoIO_set_analog_video_format(video_obj,ev,"NTSC");
          if (rc != UMSVideoIO_Success)
            {
              printf("Cannot set analog video format. rc = %d\n", rc);
              error_code = 1;
              goto Error;
            }
          /************ Set output digital video format ***********/

          if (depth == 24 )
            rc = UMSVideoIO_set_output_image_format(video_obj,ev,"RGB24");
          else
            rc = UMSVideoIO_set_output_image_format(video_obj,ev,"RGB8Dither");

          if (rc != UMSVideoIO_Success)
            {
              printf("Cannot set output image format. rc =%d \n",rc);
              error_code = 1;
              goto Error;
            }

          /************ Capture digital video as uncompressed frames ***/
          /********** Here we set the dimensions of the frames ********/

          rc = UMSVideoIO_set_uncompressed_image_size(video_obj,ev,&width,&height);
          if (rc != UMSVideoIO_Success)
            {
              printf("Cannot set uncompressed image size. rc = %d \n",rc);
              error_code =1;
              goto Error;
            }
```

```
        /*********** Set up the ring of buffers to receive the frames ***/

        /* Compute the size of one buffer ****************************/

        if (depth == 8)
          each_buffer_size = width*height;
        else
          each_buffer_size = 4*width*height;

        /* number of buffers *****************************************/
        number_of_elements = 4;

        /************* SOM sequence containing an array of ring-buffer structures */

        ring_buffer = (_IDL_SEQUENCE_UMSVideoIO_RingBufferElement*)
          malloc(sizeof(_IDL_SEQUENCE_UMSVideoIO_RingBufferElement));

        if (ring_buffer < 0)
          {
            printf("Cannot malloc ring_buffer \n");
            error_code = 1;
            goto Error;
          }
        ring_buffer->_length  = number_of_elements;
        ring_buffer->_maximum = number_of_elements;

        ring_buffer->_buffer = (struct UMSVideoIO_RingBufferElement *)
          malloc(sizeof(struct UMSVideoIO_RingBufferElement) * number_of_elements);
        if (ring_buffer->_buffer <=0)
          {
            printf("Cannot malloc ring_buffer->_buffer\n");
            error_code = 1;
            goto Error;
          }

        /************** Set up each ring buffer ************************/

        rbuffer1 = ring_buffer->_buffer;

        for (i=0; i < number_of_elements; i++)
          {
            Address[i] = (unsigned char *) malloc(each_buffer_size + 4096);
            if (Address[i] <= 0)
              {
                printf("Cannot malloc ring_buffer %d \n",i);
                error_code = 1;
                goto Error;
              }
            rbuffer1->Address = (((long) Address[i]) + 4096
                                 - (((long) Address[i]) % 4096));
            rbuffer1->AfterHeader = rbuffer1->Address;
            rbuffer1->SizeOfBuffer = each_buffer_size;
            rbuffer1->SizeOfDataInBuffer = 0;
            rbuffer1->InUseByCaller = 0;
            rbuffer1++;       /* point to the next ring buffer structure */

          }


        rc = UMSVideoIO_setup_uncompressed_capture_buffers(video_obj,ev,
                                                           ring_buffer);
        if (rc != UMSVideoIO_Success)
          {
            printf("Cannot setup uncompressed_capture_buffers. rc=%d\n",rc);
            error_code =1;
            goto Error;
          }


        /************ Set capture mode to uncompressed ***************/

        rc = UMSVideoIO_set_uncompression(video_obj,ev,UMSVideoIO_On);
        if (rc != UMSVideoIO_Success)
          {
            printf("Cannot set_uncompression. rc = %d \n", rc);
            error_code =1;
            goto Error;
          }
        rc = UMSVideoIO_set_capture_rate(video_obj,ev,&frame_rate);
        if (rc != UMSVideoIO_Success)
          {
            printf("Cannot get capture rate  rc = %d \n", rc);
            error_code =1;
            goto Error;
          }

      }
      /***************** SET UP RENEGOTIATION LINK *************************/
      connect_r(RENEG_SIGNAL_VCI,&renegid);
      /*********************************************************************/
      break;
    }

/*********************************************************************/
  MenuControl = START;

  openProfile(&MenuStateFD,"STATE");
  lseek(MenuStateFD,0L,0);
  write(MenuStateFD,(char *)(&MenuControl),sizeof(int));
  printf("Menu Control = %d and Side = %d \n",MenuControl,side);

  j=1; /*counter for reading robotics traces*/
  m=0;
  while(MenuControl == START)
/*  while (m < 50) */
    {
      for (t=0;t < Scheduler.number_of_ticks;t++)
        {
/*
          gettimeofday(&tv1,&tz);
*/
          for (i=0;i < MEDIA_NUMBER*NUMBER_OF_TASKS_PER_MEDIUM; i++)
            {
              if(Scheduler.sched[t].sched_queue[i].task_name != NOT_SPECIFIED)
                {
/*
                  printf("Scheduler.sched[t].sched_queue[i].task_name = %d \n",
                         Scheduler.sched[t].sched_queue[i].task_name);
*/
                  switch(Scheduler.sched[t].sched_queue[i].task_name)
                    {
                    case ReadVideoData:

                      /********************** read video *******************************
*/
                      gettimeofday(&t1readvideo,&tz);
                      index = 0;
                      flag = 1;
```

```
    rc = UMSVideoIO_get_uncompressed_frame(video_obj,ev,&index,flag);   */
    if(rc != UMSVideoIO_Success)
      {
        printf("Cannot get_uncompressed_frame. rc = %d \n",rc);
        error_code =1;
        goto Error;
      }

    imagedata = (unsigned char *) ring_buffer->_buffer[index].Address;


    ring_buffer->_buffer[index].InUseByCaller = 0;
    gettimeofday(&t2readvideo,&tz);
    /*****************************/
    getproctime(t1readvideo,t2readvideo,&readvideo);

    printf(" read video frame =%d microseconds \n",readvideo);

    break;
  case ReceiveDatagram:

    gettimeofday(&t1recv,&tz);                                          /*
    err.err_flag = FALSE;
    switch(Scheduler.sched[t].sched_queue[i].medium)
      {
      case  ROBOT:                                                      */
        conid = OutputAppParam.stream[ROBOT].medium.net_spec.conid[kou
t_robot];

        recv_pkt(conid,(char *)(&robot_pkt),sizeof(ROBOT_IO),err);
        if (side == SELLER && robot_pkt.kind == 0)
          {
            MenuControl = STOP;
          }

        break;
      case VIDEO:                                                       /*

        conid = OutputAppParam.stream[VIDEO].medium.net_spec.conid[kou  elltime);
t_video];                                                               */


        recv_pkt(conid,imagedata,image_size,err);
        break;
      }
    gettimeofday(&t2recv,&tz);
    /*********************************************/
    getproctime(t1recv,t2recv,&recvtime);

    printf(" recv datagram =%d microseconds \n",recvtime);
                                                                        obot];
    break;
  case SendCell:

    gettimeofday(&t1sendcell,&tz);

    err.err_flag = FALSE;                                               ideo];
    switch(Scheduler.sched[t].sched_queue[i].medium)
      {
      case ROBOT:
        conid = InputAppParam.stream[ROBOT].medium.net_spec.conid[kin_
robot];
/*
        printf("number of  conn =%d, kin_robot = %d,conid =%d\n",       ",sendtime);
               InputAppParam.stream[ROBOT].medium.net_spec.count_con,
               kin_robot, conid);
```

```
      if (InputAppParam.stream[ROBOT].medium.net_spec.count_con == 4)
        {
          switch(kin_robot)
            {
            case 0:
              size=2*sizeof(int)+3*sizeof(float);
              send_cell(conid,&robot_pkt,size,err);
              break;
            case 1:
              size = 3*sizeof(float);
              send_cell(conid,&robot_pkt.data[3],size,err);
              break;
            case 2:
              size = 3*sizeof(float);
              send_cell(conid,&robot_pkt.data[6],size,err);
              break;
            case 3:
              size = 3*sizeof(float);
              send_cell(conid,&robot_pkt.data[9],size,err);
              break;

              printf("%f, %f, %f \n",
                     robot_pkt.data[9],robot_pkt.data[10],
                     robot_pkt.data[11]);


            }
        }
      break;
    }
  kin_robot++;
  gettimeofday(&t2sendcell,&tz);
  /*************************************/
  getproctime(t1sendcell,t2sendcell,&sendcelltime);

  printf(" send cell  delay in one interval=%d microseconds \n",sendc


  sendrobdata = sendrobdata+sendcelltime;
  break;
case SendDatagram:
  gettimeofday(&t1send,&tz);
  err.err_flag = FALSE;
  switch(Scheduler.sched[t].sched_queue[i].medium)
    {
    case ROBOT:
      conid = InputAppParam.stream[ROBOT].medium.net_spec.conid[kin_r


      send_pkt(conid,&robot_pkt,sizeof(ROBOT_IO),err);
      break;
    case VIDEO:


      conid = InputAppParam.stream[VIDEO].medium.net_spec.conid[kin_v


      send_pkt(conid,imagedata,image_size,err);
      gettimeofday(&t2send,&tz);
      /*************************************/
      getproctime(t1send,t2send,&sendtime);


      printf(" send datagram delay in one interval=%d microseconds \n


      break;
```

```
    }
  break;
case ReceiveCell:
  gettimeofday(&t1recvcell,&tz);
  err.err_flag = FALSE;
  switch(Scheduler.sched[t].sched_queue[i].medium)
    {
    case ROBOT:
      conid = OutputAppParam.stream[ROBOT].medium.net_spec.conid[kou
t_robot];
/*

      printf("  number of conn. =%d,kout_robot = %d,conid=%d \n",
             OutputAppParam.stream[ROBOT].medium.net_spec.count_con,
             kout_robot,conid);
*/                                                                    /*
      if (OutputAppParam.stream[ROBOT].medium.net_spec.count_con ==
4)                                                                    */
        {
          switch(kout_robot)
            {
            case 0:
              size=2*sizeof(int)+3*sizeof(float);
              recv_cell(conid,&robot_pkt,size,err);
              break;
            case 1:
              size = 3*sizeof(float);
              recv_cell(conid,&robot_pkt.data[3],size,err);
              break;                                                  f, %f\n",
            case 2:
              size = 3*sizeof(float);
              recv_cell(conid,&robot_pkt.data[6],size,err);
              break;
            case 3:
              size = 3*sizeof(float);
              recv_cell(conid,&robot_pkt.data[9],size,err);
              break;
            }
        }
      break;
    }
  kout_robot++;
  gettimeofday(&t2recvcell,&tz);
  /*********************************************/
  getproctime(t1recvcell,t2recvcell,&recvcelltime);
  if (first == TRUE && side == SELLER)
    {
      recvcelltime = 155;
      first = FALSE;
    }
  recvrobdata = recvrobdata + recvcelltime;
/*
  printf(" recvcell delay in one interval=%d microseconds \n",recvce  /*
lltime);
*/                                                                    */

  break;
case ReadRobotData:
  /*                      printf("ReadRobotData \n"); */
  gettimeofday(&t1read,&tz);
  kin_robot=0;
  application(&robot_pkt,j);

  printf("Read Robot Data:%f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %f
         robot_pkt.data[0],robot_pkt.data[1],
         robot_pkt.data[2],robot_pkt.data[3],
         robot_pkt.data[4],robot_pkt.data[5],
         robot_pkt.data[6],robot_pkt.data[7],
         robot_pkt.data[8],robot_pkt.data[9],
         robot_pkt.data[10],robot_pkt.data[11]);
  j++;
  if (j == 19)
    {
      j=1;
    }
  gettimeofday(&t2read,&tz);
  /*********************************************/
  getproctime(t1read,t2read,&readtime);

  printf(" read robot=%d microseconds \n",readtime);

  break;
case CopyRobotData:
  printf("CopyRobotData \n");
  break;
case WriteRobotData:
  gettimeofday(&t1write,&tz);
  kout_robot = 0;
  /*  if (side == BUYER) { */

  printf("Write Robot Data: %f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %
         robot_pkt.data[0],robot_pkt.data[1],
         robot_pkt.data[2],robot_pkt.data[3],
         robot_pkt.data[4],robot_pkt.data[5],
         robot_pkt.data[6],robot_pkt.data[7],
         robot_pkt.data[8],robot_pkt.data[9],
         robot_pkt.data[10],robot_pkt.data[11]);

  /*  write to bit 3 card }
    if (side  == SELLER)
    {
      if (remote_in_ptr < (remote_addr_in + CIRCULAR_BUFFER_SIZE))
        {
          write(chan,&robot_pkt,LENGTH);
          remote_in_ptr +=LENGTH;
        }
      else
        {
          remote_in_ptr = remote_addr_in;
        }
    }*/
  gettimeofday(&t2write,&tz);
  /*********************************************/
  getproctime(t1write,t2write,&writetime);

  printf(" write robot data =%d microseconds \n",writetime);

  break;
case CopyVideoData:
  break;
case WriteVideoData:

  /**************** Display Image ******************/
  gettimeofday(&t1writevideo,&tz);
  image = XCreateImage(dpy, my_visual, depth, ZPixmap, 0, imagedata,
                       width, height, pixel_pad, 0);
  XPutImage(dpy, wid, gc, image, 0, 0, 0, 0, width, height);
```

```
                    XSync(dpy,FALSE);
                    XFlush(dpy);
                    gettimeofday(&t2writevideo,&tz);
                    /*********************************************/
                    getproctime(t1writevideo,t2writevideo,&writevideo);

                    printf(" display video frame =%d microseconds \n",writevideo);


                    break;
                  case PointerManagementRobotData:
                    break;
                  case PointerManagementVideoData:
                    break;
                  case Renegotiate:
                    err.err_flag = FALSE;
                    gettimeofday(&t1ren,&tz);
                    switch(side)
                      {
                      case BUYER:

                        lseek(MenuStateFD,0L,0);
                        read(MenuStateFD,(char *)(&MenuControl),sizeof(int));
                        r_info.menu_state = MenuControl;
/*
                        printf("BUYER:Menu Control is =%d, r_info.menu_state = %d \n",
MenuControl,r_info.menu_state);
*/
                        if (MenuControl == RENEGOTIATE)
                          {
/*
                            printf("RTAP: RENEGOTIATE \n");
*/
                            lseek(MenuStateFD,sizeof(int),0);
                            read(MenuStateFD,(char *)(&r_info.changed_rate),sizeof(int
));
/*
                            printf("RTAP: changed rate = %d \n",
                                   r_info.changed_rate);
*/
                            MenuControl = START;
                            lseek(MenuStateFD,0L,0);
                            write(MenuStateFD,(char *)(&MenuControl),
                                  sizeof(int));
                          }
                        send_cell(renegid,&r_info,sizeof(RENEGOT_INFO),err);

                        break;
                      case SELLER:
                        recv_cell(renegid,&r_info,sizeof(RENEGOT_INFO),err);
/*
                        printf("SELLER: r_info.menu_state =%d \n",r_info.menu_state);
*/
                        MenuControl = r_info.menu_state;
/*
                        printf("SELLER:Menu Control is =%d \n",MenuControl);
*/
                        if (MenuControl == RENEGOTIATE)
                          {
/*
                            printf("RTAP: RENEGOTIATE \n");
                            printf("RTAP: changed rate=%d \n",
                                   r_info.changed_rate);
*/
                            changedDelay=1000000/r_info.changed_rate;
/*
                            printf("changed delay=%d \n",
                                   changedDelay);
*/
                            MenuControl = START;
                          }
                        break;
                      }
                    gettimeofday(&t2ren,&tz);
                    getproctime(t1ren,t2ren,&clock);
/*
                    printf("Renegotiate overhead=%d [microsec] \n",clock);
*/
                    break;
                  }
              }
            else
              {
                i=MEDIA_NUMBER*NUMBER_OF_TASKS_PER_MEDIUM;
              }
          }
/*        gettimeofday(&tv2,&tz);
          getproctime(tv1,tv2,&runtime);
*/

/*
          printf("<proc time,changed Delay>=<%d,%d>  \n",recvrobdata,changedDelay);
*/
          runtime = recvrobdata+sendrobdata + readtime + writetime +sendtime+recvtime+ wr
itevideo+readvideo;
          recvrobdata =0;
          sendrobdata =0;
          readvideo =0;
          writevideo=0;
          sendtime=0;
          recvtime=0;
          readtime=0;
          writetime = 0;
          if (Scheduler.min_period+changedDelay > runtime)
            {
/*
              printf("I go to sleep \n");
*/
              usleep((Scheduler.min_period + changedDelay) - runtime - diff);
            }
          else
            {
              diff = runtime-Scheduler.min_period;
              printf("miss the deadline by =%d \n",
                     (runtime-Scheduler.min_period));
            }
        }
      printf("counter m is = %d \n",m);

      m++;
    }

  printf("I am done with RTAP \n");

  XFreeGC(dpy,gc);
  XDestroyWindow(dpy,wid);

  free(imagedata);
```

```
  if (InputAppParam.stream[VIDEO].type == VIDEO)
    {
      error_code = 0;

  /* Close the video IO object */

      rc = UMSVideoIO_close(video_obj,ev);
      if (rc !=UMSVideoIO_Success)
        {
          printf("Cannot close video device. rc = % \n", rc);
          error_code = 1;
        }
      for (i=0; i < number_of_elements; i++)
        {
          if (Address[i] != NULL)
            free(Address[i]);
        }
      if (ring_buffer->_buffer != NULL)
        free(ring_buffer->_buffer);
      if (ring_buffer != NULL)
        free(ring_buffer);

      if (video_obj != NULL)
        _somFree(video_obj);
      return(error_code);
    }

  exit(0);

/********** Free buffers, and destroy the video object ********/

    Error:

      for (i=0; i < number_of_elements; i++)
        {
          if (Address[i] != NULL)
            free(Address[i]);
        }
      if (ring_buffer->_buffer != NULL)
        free(ring_buffer->_buffer);
      if (ring_buffer != NULL)
        free(ring_buffer);

      if (video_obj != NULL)
        _somFree(video_obj);
      return(error_code);

}
```

```
/*************************************************************************/
/* Filename: main_slave.c                                                */
/* Purpose : Slave side of communication in retta                        */
/* Author  : Klara Nahrstedt                                             */
/* Update  : 07/01/95                                                    */
/*************************************************************************/

#include "/home/klara/tele.d/include.d/defs.h"
#include "/home/klara/tele.d/include.d/retta.h"
#include "/home/klara/tele.d/include.d/comm.h"
#include "/home/klara/tele.d/include.d/systemQoS.h"

int pid;

main(argc,argv)
int argc;
int *argv[];
{
  APP_QOS SlaveInputParam,SlaveOutputParam;
  ADD_INFO add_info;
  MM_DEVICES in,out;

  int onalarm,inout;

  int pri;
  pid_t p_pid;

  FEC_FLAGS err;
  int MenuControl;
  int side =SLAVE;
  NOTIFY notification;
  GLOBAL_STATE SystemState;
  RATE_MONOTONIC_SCHEDULER Schedule;
  INFO_STATE WhatInfo;
  NET_QOS_TABLE InputNetParam;
  NET_QOS_TABLE OutputNetParam;
  MenuControl = CALL_SET_UP;



  p_pid = getpid();
  pri = 0;
  if (setpri(p_pid,pri) <0)
    perror("setpri");


  bzero((char *)(&SystemState),sizeof(GLOBAL_STATE));
  bzero((char *)(&Schedule),sizeof(RATE_MONOTONIC_SCHEDULER));
  bzero((char *)(&InputNetParam),sizeof(NET_QOS_TABLE));
  bzero((char *)(&OutputNetParam),sizeof(NET_QOS_TABLE));
  setNetQoS(&InputNetParam,INPUT);
  setNetQoS(&OutputNetParam,OUTPUT);

  WhatInfo.i_set[0] = SystemStateInfo;
  WhatInfo.i_set[1] = ScheduleInfo;
  StoreGlobalState(SystemState,Schedule,WhatInfo);

  init_app_qos(&SlaveInputParam);
  init_app_qos(&SlaveOutputParam);
  init_devices(&in);
  init_devices(&out);

  installDevices(&in,SLAVE,INPUT);
  installDevices(&out,SLAVE,OUTPUT);
  setAppQoS(&SlaveInputParam,INPUT);
  setAppQoS(&SlaveOutputParam,OUTPUT);
  SetTaskParam(SELLER);
/*  QoSBroker(&add_info,&notification,out,SELLER,OUTPUT); */
  WhatInfo.i_set[0] = SystemStateInfo;
  WhatInfo.i_set[1] = NOT_SPECIFIED;

  err.err_flag = FALSE;
  while(MenuControl != QUIT)
    {
      RetrieveGlobalState(&SystemState,&Schedule,WhatInfo);
      printf("Wait for STATE from Master \n");
/* receive Menu Control Command only when control connection was established
   between buyer and seller */

      if (SystemState.net.Aneg_out.status == TAKEN)
        {
          recv_pkt(SystemState.net.Aneg_out.id,&MenuControl,sizeof(int),err);
        }
      switch(MenuControl)
        {
        case CALL_SET_UP:
          printf("Received from master CALL_SET_UP \n");
          QoSBroker(&SlaveOutputParam,
                    &add_info,
                    &notification,
                    SELLER,
                    inout,
                    NEGOTIATE);
          printf("SLAVE: QoS broker for input done \n");
          break;
        case START:
          printf("Received from master START \n");
          if ((pid = fork()) == 0)
            {
              rtap(SELLER);
            }
/*        signal(SIGALRM,onalarm); */
          printf("Leave START \n");
          break;
/*
        case STOP:
          printf("Received from master STOP \n");
          alarm(1);
          break;
*/
        case QUIT:
          printf("Received from master QUIT \n");
          break;
        }

    }
}

onalarm()
{
  kill(pid,SIGKILL);
}
```

```
/*************************************************************************/
/* Filename : QoSutil.c                                                  */
/* Puprose: Utility functions for QoS management                         */
/* Update : 6/29/95                                                      */
/* Author : KLara Nahrstedt                                              */
/*************************************************************************/


#include "/home/klara/tele.d/include.d/defs.h"
#include "/home/klara/tele.d/include.d/retta.h"
#include "/home/klara/tele.d/include.d/comm.h"
#include "/home/klara/tele.d/include.d/systemQoS.h"

/*************************************************************************/
/* Translate Rate between Application and transport Subsystem            */
/*************************************************************************/

double transRate(appPktSize,appPktRate, netPktRate,direction,mode,ceiling)
int appPktSize;
int appPktRate;
int netPktRate;
int direction;
int mode;
double *ceiling;
{
  double rresult;
  double term;

  switch(mode)
    {
    case CELL_MODE:
      term = ((double) appPktSize)/((double) CELL_SIZE);
      break;
    case DATAGRAM_MODE:
      term = ((double) appPktSize)/((double) DATAGRAM_SIZE);
      break;
    }
  *ceiling = ceil(term);

  switch(direction)
    {
    case APP_TO_NET:
      rresult = ceil(term)*((double) appPktRate);
      return(rresult);
      break;
    case NET_TO_APP:
      rresult = ((double) netPktRate)/(ceil(term));
      return(rresult);
      break;
    }

}/*transRate */

/*************************************************************************/
/* Intermedia Delay Translation                                          */
/*************************************************************************/

double transInterDelay(appPktSize,rate,direction,mode)
int appPktSize;
int rate;
int direction;
int mode;
{
  double rresult;
  double term;

  switch(mode)
    {
    case CELL_MODE:
      term = ((double) appPktSize)/((double) CELL_SIZE);
      break;
    case DATAGRAM_MODE:
      term = ((double)  appPktSize)/((double) DATAGRAM_SIZE);
      break;
    }

  switch(direction)
    {
    case APP_TO_NET:
/*************************************************************************/
/* Assume that rate is samples per second and rresult is in miliseconds  */
/*************************************************************************/
      rresult = ((1.0/((double) rate))*1000.0)/(ceil(term));
      return(rresult);
      break;
    case NET_TO_APP:
      rresult = ((1.0/((double) rate))*1000.0)*(ceil(term));
      return(rresult);
      break;
    }
}

/*************************************************************************/
/* Computation of Greatest Common Multiplier                             */
/*************************************************************************/
long mygcdnew(a,b)
long a;
long b;
{
  long helpvar;

  while(b!=0)
    {
      helpvar = a;
      a=b;
      b=helpvar%b;
    }
  return a;
}

long mygcd(a,b)
long a;
long b;
{
  if (b==0)
    {
      return a;
    }
  else
    {

      return mygcd(b,a%b);
    }
}
/*************************************************************************/
/*  Computation of Least Common Multiplier                               */
/*************************************************************************/
```

```
long lcm(x,xn)
long x[2*MEDIA_NUMBER];
int xn;
{
  int i,j,k,m;
  BOOLEAN first;
  long r;
  long result;

/********* number of results from gcd is #xn over 2 ******************/
/********* in our case xn maximal = 2*Number_Of_Media (8)  *************/
  long gcd_results[10];
  bzero((char *)(&gcd_results[0]),sizeof(gcd_results));
  k=0;
  first = TRUE;

  for (i=0; i< xn;i++)
    {
      x[i] = x[i]/1000;
    }

  for (i=0;i < xn;i++)
    {
      for (j=i+1;j < (xn+1);j++)
        {
          r=mygcdnew(x[i],x[j]);
          if (first)
            {
              gcd_results[0] =r;
              first = FALSE;
              k++;
            }
          else
            {
              for (m=0;m<k;m++)
                {
                  if (gcd_results[m] == r)
                    {
                      m=k;
                      gcd_results[k]=1;

                    }
                  else
                    if (m== k-1)
                      {
                        gcd_results[k] = r;
                      }
                }
              k++;
            }
        }
    }
  result =1;
  for (i=0;i<xn;i++)
    {
      result = result*x[i];
    }
  r=1;
  for (i=0;i<k-1;i++)
    {
      r = r*gcd_results[i];
    }
  result = result/r;
  result = 1000*result;
  for (i=0; i<xn;i++)
    {
      x[i]=x[i]*1000;
    }
  return result;
}

/*************************************************************************/
/* GET TASK DURATION PARAMETER                                           */
/*************************************************************************/

int getOneTaskParam(task,name,medium,inout)
TASK *task;
int name;
int medium;
int inout;
{
  TASKS task_i;
  int i;

  GetTaskParam(&task_i,medium,inout);
  for (i=0; i<NUMBER_OF_TASKS_PER_MEDIUM; i++)
    {
      if (name == task_i.app[i].name)
        {
          task->duration = task_i.app[i].duration;
          return;
        }
    }
}

/*************************************************************************/
/* GET THROUGHPUT OF ALL CONNECTIONS IN ONE DIRECTION                    */
/*************************************************************************/

int getThroughput(Param,throughput)
NET_QOS_TABLE *Param;
double *throughput;
{
  int i;
  double help;

  help = 0.0;
  for (i=0; i< CONNECTION_NUMBER;i++)
    {
      if (Param->status[i] == TAKEN)
        {
          help = help + Param->connection[i].load.throughput;
        }
    }
  *throughput = help;
}

/*************************************************************************/
/* TIME CALCULATION                                                      */
/*************************************************************************/

getproctime(tv1,tv2,clock)
struct timeval tv1,tv2;
long *clock;
{
  if (tv1.tv_usec > tv2.tv_usec)
    {
```

```
      tv2.tv_usec +=MILLION;
      tv2.tv_sec -=1;
    }
  *clock = MILLION * (tv2.tv_sec - tv1.tv_sec) +
    (tv2.tv_usec - tv1.tv_usec);

}

/*********************************************************************/
/* RELEASE MEDIUM DESCRIPTION                                        */
/*********************************************************************/

void releaseMedium(Param,medium)
APP_QOS *Param;
int medium;
{


   Param->stream[medium].type = 0;
   Param->stream[medium].medium.app_spec.quality = 0;
   Param->stream[medium].medium.app_spec.sample_size = 0;
   Param->stream[medium].medium.app_spec.sample_rate = 0;
   Param->stream[medium].medium.net_spec.end_to_end_delay = 0;
   Param->stream[medium].medium.net_spec.loss_rate = 0;
   Param->stream[medium].medium.net_spec.importance = 0;
   Param->stream[medium].medium.net_spec.conid[0] = 0;
 }

/*********************************************************************/
/* FREE SCHEDULER  RESOURCE                                          */
/*********************************************************************/

void freeSchedResources(scheduler,interval,queue_no)
RATE_MONOTONIC_SCHEDULER *scheduler;
int interval;
int queue_no;
{
   scheduler->sched[interval].sched_queue[queue_no].medium =0;
   scheduler->sched[interval].sched_queue[queue_no].task_name =0;
   scheduler->sched[interval].sched_queue[queue_no].task_duration = 0;
   scheduler->sched[interval].sched_queue[queue_no].time_begin = 0;
   scheduler->sched[interval].sched_queue[queue_no].time_deadline =0;
}


/*********************************************************************/
/* RELEASE CONNECTION DESCRIPTION                                    */
/*********************************************************************/

releaseConnection(Param,k)
NET_QOS_TABLE *Param;
int k;
{
  Param->status[k] = FREE;
  Param->connection[k].load.id =0;
  Param->connection[k].load.size =0;
  Param->connection[k].load.loss.loss_rate =0;
  Param->connection[k].load.loss.loss_cons_pkt = FALSE;
  Param->connection[k].load.rate = 0.0;
  Param->connection[k].load.throughput = 0.0;
  Param->connection[k].load.end_to_end_delay=0.0;
  Param->connection[k].load.intermediate_delay=0.0;
  Param->connection[k].load.priority = 0;
}
```