

EECS 123:
**Introduction to
Real-Time Distributed Programming**
Lecture : RMMC

Feb-07	1
--------	---

UCI
DREAM Lab



Introduction

Basic Goals of Multicasts

- Two conceivable needs for group communications in RT distributed computing systems
 - [Server replicas](#) and [tightly coupled heterogeneous servers](#)
 - [News multicast](#) to casual readers
 - Attain high performance in operating tightly coupled servers and/or achieving casual news multicasts
- Two additional purposes
 - [Abstract distributed programming](#)
 - Basic building blocks for fault-tolerant distributed systems
 - Not important at this time because suitable / credible ways of doing this have not matured

Feb-07	2
--------	---

UCI
DREAM Lab



Introduction

- Multicast channels for abstract distributed programming
 - TMOs can use another interaction mode in which messages can be exchanged over **logical message channels**.
 - **Access gates** for such channels are explicitly specified as **data members**, more specifically, **special types of ODSSs**, of involved objects.
- **RMMC** (Real-time Multicast and Memory-replication Channel)
 - One of the most advanced types of multicast channels

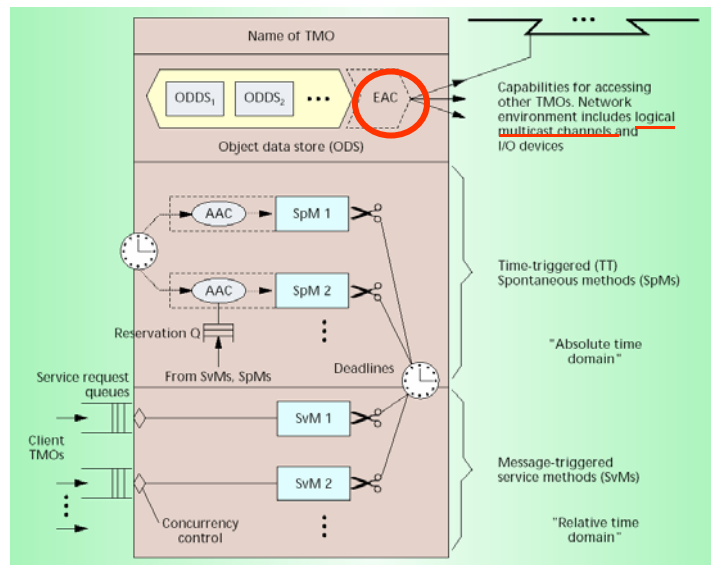
Feb-07 3

UCI
DREAM Lab



Real-time Multicast and Memory-replication Channel (RMMC)

- Access gates for channels explicitly specified as data members of TMO objects



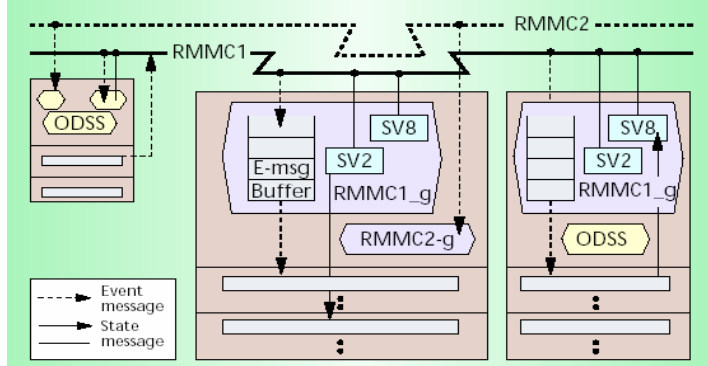
Feb-07 4

UCI
DREAM Lab



Real-time Multicast and Memory-replication Channel (RMMC)

RMMCs accessed by TMO methods



- Access gates for two RMMCs (RMMC1 and RMMC2) can be declared as data members (special types of ODSS's) of each of the 3 TMOs (which may be dispersed widely) during the design time.
- One gate is declared as one subscriber to an RMMC channel. One TMO can have multiple gates for the same RMMC channel.

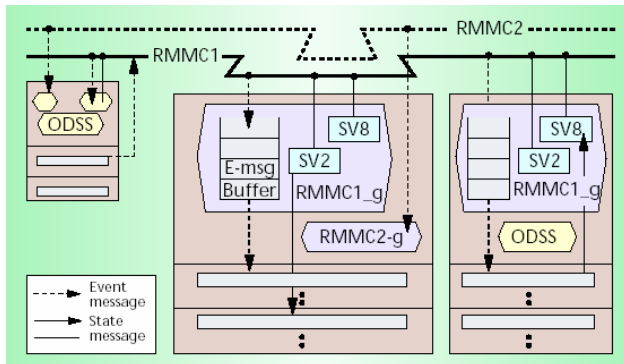
Feb-07 5

UCI
DREAM Lab



RMMC

- An event message sent over RMMC1 will be delivered to the buffer allocated inside the execution engine for each of the subscribing RMMC1-Gates except the RMMC1-Gate through which the message goes out.
- Later, a certain method in each TMO with a receiving RMMC1-Gate can pick up those messages by sending requests through the gate to the exec engine.
- An RMMC can be implemented over point-to-point networks as well as over broadcast-enabled bus networks.



Feb-07 6

UCI
DREAM Lab



Event Messages and State Messages

- RMMC scheme supports not only conventional **event messages** but also **state messages** based on **distributed replicated memory semantics**.
- **Event message**
 - An event message **must be read by every corresponding subscriber** (RMMC-Gate) other than the announcing subscriber.
 - The announcing subscriber cannot obtain its event messages.
 - Every message sent must not be ignored by other subscribers.
 - If a subscriber doesn't read the message quickly enough, an **event message queue overflow** or a **time-out**, both of which are error detection events, can occur.

In principle, the programmer should have the option of specifying that a **message which has waited in the buffer for a maximum residency period** can be simply discarded without raising an error signal.
 - An event message producer **timestamps** the message at message-production time.

Feb-07

7

UCI
DREAM Lab

Event Messages and State Messages

- **State message**
 - A state message carries information to be stored in a **fixed memory location in each subscriber** corresponding to the **ID of the state message**.
 - A state message's ID represents a **group of replicated memory units**, each capable of holding the information carried in the state message and belonging to a different subscriber.
 - A state message producer **timestamps** the message at message-production time.
 - A method in each TMO with a subscriber RMMC1-Gate can **read** the content of its state msg memory through the gate **at its convenient time**.
 - The producer may **update** the contents of the state message memory units **at a higher frequency** than the frequency at which a certain consumer reads the content of one of the state message memory units.
 - A state message is thus typically used to share the periodically observed state information about a dynamic state-varying item, e.g., a car's position.

Feb-07

8

UCI
DREAM Lab

Official Release Time (ORT)

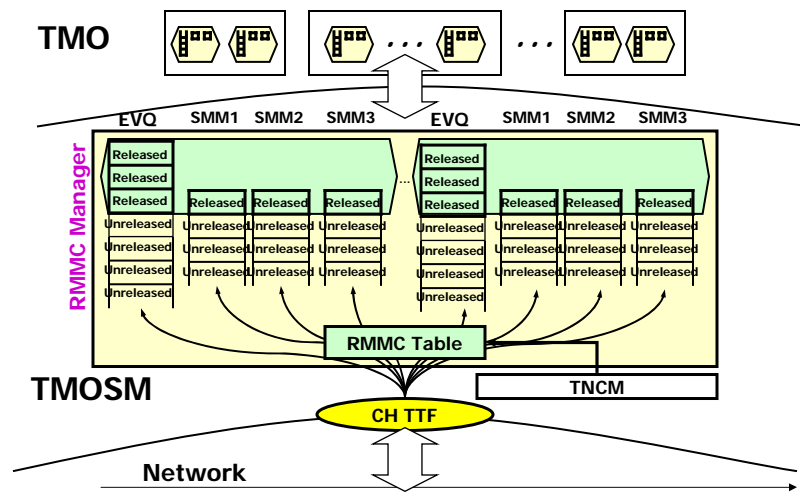
- The **official release time** (ORT) is the time at which the event / state message should become **accessible** through subscriber RMMC-Gates to the consumer methods.
- A (consumer) method in each TMO with a subscriber RMMC-Gate picks up **event messages** from the Q associated with the gate one at a time **in the order of their ORTs**.
- Consumer methods in each TMO with a subscriber RMMC-Gate can read only **the most recent officially released state messages** kept in the state msg memories associated with the gate.
- If **ORT is not given by the sender**, messages are released to receiving subscribers **ASAP**.

Feb-07 9

UCI
DREAM Lab



RMMC and TMOSM



Feb-07 10

UCI
DREAM Lab



Official Release Time (ORT)

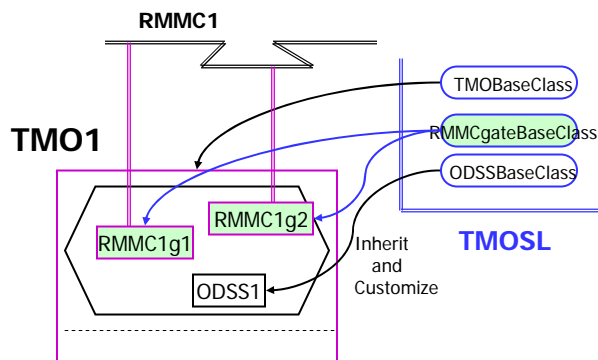
- **Toshiba** has a patent on the idea of using ORT !
 - Invented in 1996 and US Patent was granted in April 2001 but we learned the invention only in February 2005 although we have been using it since mid-1999.
 - As will be discussed later, ORT can also be emulated at the application level .

Feb-07 | 11

UCI
DREAM Lab



Creation of an RMMC Access Gate & an Enclosing TMO



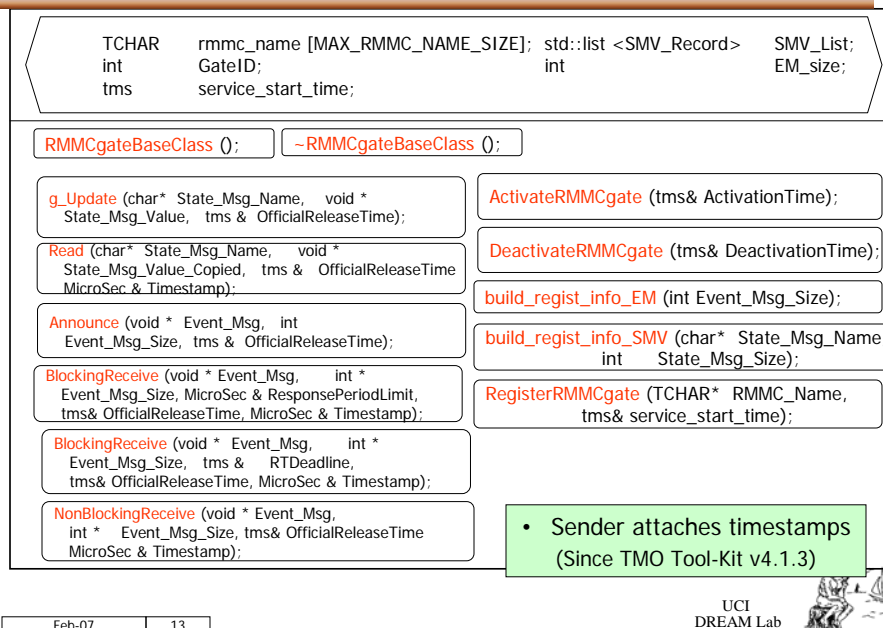
- The RMMC-Gate, not the TMO, is the unit for subscribing to an RMMC.
- Multiple RMMC-Gates for the RMMC may be used within a TMO.

Feb-07 | 12

UCI
DREAM Lab



RMMCgateBaseClass



API's

- Common APIs for establishing an RMMC-Gate

```
// Initialize the data members of RMMCgateBaseClass
- RMMCgateBaseClass::RMMCgateBaseClass ()

// Remove the RMMC gate from the list of gates which connect to
// the RMMC
- RMMCgateBaseClass::~~RMMCgateBaseClass ()

/* Prepare a copy of the subject State Message variable in a form for
registration inside a local (node) instantiation of TMOSM.
The first parameter, which is a character string, will be used as the
globally recognized name of the subject SM variable. */
- void RMMCgateBaseClass::build_regist_info_SMV (
TCHAR * SM_name, int SM_size);
```

Feb-07	14
--------	----



API's

- Common APIs for establishing a RMMC

```
// Prepare the size information of the Event Message
// in a form for Registering within a local (node) instantiation
// of TMOSM
```

- **void RMMCgateBaseClass::build_regist_info_EM** (int EM_size)

```
// Create one support record inside TMOSM
// for a gate to the specified RMMC.
```

- **BOOL RMMCgateBaseClass::RegisterRMMCgate** (
TCHAR* RMMC_Name, tms & service_start_time)

Feb-07	15
--------	----

UCI
DREAM Lab



API'S (cont.)

- Event Message API's

```
/* Announce an "event_msg" to all subscribers (RMMC-Gates) of the
subject RMMC except for the announcing subscriber.
```

Each of the TMOSM nodes hosting subscribers will create a copy of this message for each local subscriber as the message comes in.

The "OfficialReleaseTime" indicates when this event message should be released to each subscriber.

If "OfficialReleaseTime" is not provided or if it is set to "0", this event message will be released to subscribers ASAP.

```
Returns SUCCESS or FAIL (when failure signals have been raised by
TMOSM). */
```

- int **RMMCgateBaseClass::Announce** (void* event_msg,
int Msg_size, tms OfficialReleaseTime)

Feb-07	16
--------	----

UCI
DREAM Lab



API's (cont.)

- Event Message API's

/* Block until an "event_msg" is received or the amount of time indicated by "ResponsePeriodLimit" or "RTDeadline" passes by.

Returns SUCCESS, FAIL (when failure signals have been raised by TMOSM), or TIMEOUT indicating inability to complete the message pickup operation within ResponsePeriodLimit while no other failure signals have been raised by TMOSM.

After completion of this function, the result parameter "OfficialReleaseTime" contains the ORT that was attached to "event_msg". In case "OfficialReleaseTime" is 0, this event message is delivered to subscribers ASAP.

The result parameter "Timestamp" contains the "send-timestamp" created by the announcing subscriber. */

- int **RMMCGateBaseClass::BlockingReceive** (void* event_msg, int * Received_Msg_size, microsec & ResponsePeriodLimit, tms & OfficialReleaseTime, MicroSec & Timestamp);
- int **RMMCGateBaseClass::BlockingReceive** (void* event_msg, int * Received_Msg_size, tms & RTDeadline, tms & OfficialReleaseTime, MicroSec & Timestamp);

Feb-07

17

UCI
DREAM Lab

API's (cont.)

- Event Message API's

/* Checks if an "event_msg" has arrived and if it has, picks it up. Returns SUCCESS, FAIL (when failure signals have been raised by TMOSM), or NO_VALUE without blocking.

After completion of this function, the result parameter "OfficialReleaseTime" contains the ORT that was attached to "event_msg".

In case "OfficialReleaseTime" is 0, this event message is delivered to subscribers ASAP.

The result parameter "Timestamp" contains the "send-timestamp" created by the announcing subscriber. */

- int **RMMCGateBaseClass::NonBlockingReceive** (void* event_msg, int * Received_Msg_size, tms & OfficialReleaseTime, MicroSec & Timestamp)

Feb-07

18

UCI
DREAM Lab

API'S (cont.)

- State Message API's

```
/* Perform a global update of the State Message variable.
   I.E., Update all distributed replicas of the SM variable.
   Note that an SM variable is referenced by a character string.
   All TMOSM nodes hosting subscribers will update their copies of the SM
   variables.
```

```
The "OfficialReleaseTime" indicates when the new value provided by
"State_Msg_Value_To_Be_Stored" should be released to each subscriber.
If "OfficialReleaseTime" is not provided, this state message will be released
to subscribers ASAP.
```

```
Returns SUCCESS or FAIL (when failure signals have been raised by TMOSM)
*/
```

```
- int RMMCgateBaseClass::g_Update (char* StateMsg_Name,
  void * StateMsg_Value_To_Be_Stored,
  tms OfficialReleaseTime)
```

Feb-07	19
--------	----

UCI
DREAM Lab



API'S (cont.)

```
/* Read the local copy (kept in the local TMOSM supporting the subject
   RMMC-Gate) of the State Message variable.
```

```
After completion of this function, the result parameter "OfficialReleaseTime"
contains the ORT attached to the value of the StateMsg variable which will
be copied into the result parameter "State_Msg_Value_Copied".
In case "OfficialReleaseTime" is 0, this state message is delivered to
subscribers ASAP.
```

```
The result parameter "Timestamp" contains the "send-timestamp" created
when this state message was sent out from the updating subscriber.
```

```
Note that this is a non-blocking function call.
Returns SUCCESS or FAIL (when failure signals have been raised by
TMOSM). */
```

```
- int RMMCgateBaseClass::Read (char* StateMsg_Name,
  void * StateMsg_Value_Copied,
  tms & OfficialReleaseTime,
  MicroSec & Timestamp)
```

Feb-07	20
--------	----

UCI
DREAM Lab



API's (cont.)

```

/*This API will be used to disconnect the RMMC-Gate from the RMMC.
Once an RMMC-Gate is deactivated, methods in the owner TMO can still
access the RMMC-Gate but the gate does not provide a connection to the
RMMC and thus can neither receive and announce Event messages nor
read and g_update State messages communicated through the RMMC.

Until Deactivation_Time is reached, the RMMC gate remains connected to
the RMMC. For receiving Event messages, those messages of which
official release times (ORTs) are later than Deactivation time cannot be
received. Announcement of Event messages can succeed if the
messages pass through the RMMC-Gate before Deactivation_Time.

For reading State messages, those messages of which ORTs are later than
Deactivation_Time cannot be read. g_Update of State messages can
succeed if the messages pass through the RMMC-Gate before
Deactivation_Time. */

```

```

- int RMMCgateBaseClass::DeactivateRMMCgate (
    tms & Deactivation_Time)

```

Feb-07

21

UCI
DREAM Lab

API's (cont.)

```

/* This API will be used to connect the RMMC-Gate to the RMMC from which
the gate was disconnected earlier.
Methods in the owner TMO can receive through the RMMC-Gate those
Event messages of which Official Release Times (ORTs) are after
Reactivation_Time.
Announcement of Event messages can succeed if Announce () is called
after Reactivation_Time.

Similarly, the TMO methods can read State messages communicated through
the RMMC and tagged ORTs which are after Reactivation_Time.
g_Update of State messages can succeed if g_Update () is called after
Reactivation_Time. */

```

```

- int RMMCgateBaseClass::ActivateRMMCgate (
    tms & Reactivation_Time)

```

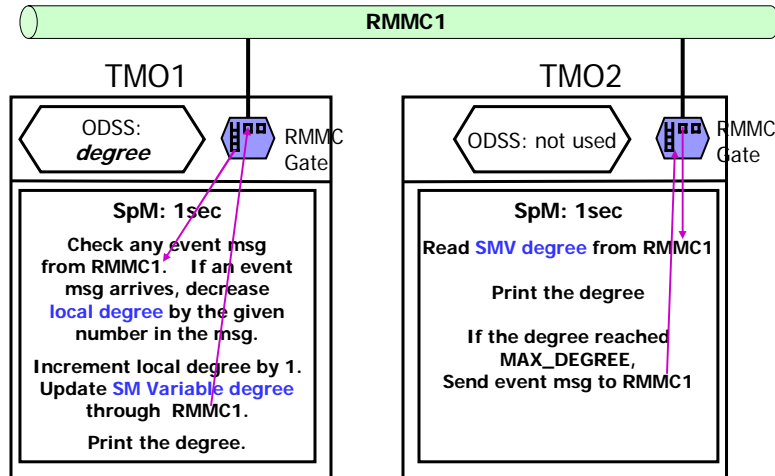
Feb-07

22

UCI
DREAM Lab

Example

- This example consists of two TMOs which share one RMMC



Feb-07

23

UCI
DREAM Lab

Example (cont.)

- RMMC.h

```
#define MAX_DEGREE 5

// Struct for State Msgs
struct SM1struct_RMMC1
{
    int degree;
};

// Struct for Event Msg (also can be defined dynamically)
struct EMstruct_RMMC1
{
    int degree;
    TCHAR Message[30];
};
```

Feb-07

24

UCI
DREAM Lab

Example (cont.)

- RMMC.h (cont.)

```
// RMMC for state message Comm
class CRMMCGateClass: public RMMCGateBaseClass
{
public:
    CRMMCGateClass (TCHAR* RMMCname)
    {
        // build state message info
        build_regist_info_SMV (_T("SM1"), sizeof (SM1struct_RMMC1));
        // build event message info
        build_regist_info_EM (sizeof (EMstruct_RMMC1));
        // RMMC gate Registration
        RegisterRMMCGate (RMMCname);
    };
};
```

Feb-07	25
--------	----

UCI
DREAM Lab



Example (cont.)

- TMO1.h

```
#include "RMMC.h"
class ODSSClass1: public ODSSBaseClass <ODSSClass1> {
public:
    int Degree;
    ODSSClass1 () { Degree = 0; }
};

class TMO1: public CTMOBase
{
public:
    TMO1 (TCHAR *, TCHAR*, tms);
private:
    ODSSClass1          ODSS1;
    int                 SpM1 ();
    CRMMCGateClass     RMMC_TMO1;
};
```

Feb-07	26
--------	----

UCI
DREAM Lab



Example (cont.)

- TMO1.cpp

```
TMO1::TMO1 (TCHAR * TMO_name, TCHAR* sRMMCName,
           tms TMO_start_time)
:RMMC_TMO1 (sRMMCName)
{
    // register SpM
    AAC aac1 (NULL,           // null for candidate aac label
             tm4_DCS_age (1*1000*1000), // from
             tm4_DCS_age (60*1000*1000), // until
             1 * 1000 * 1000, // every
             0,                // EST
             100*1000,         // LST
             200*1000         // by
            );
```

Feb-07	27
--------	----

UCI
DREAM Lab



Example (cont.)

- TMO1.cpp

```
SpM_RegistParam spm_spec;
spm_spec.build_regist_info_AAC (aac1);
spm_spec.build_regist_info_ODSS (ODSS1.GetId(), RW);
spm_spec.build_regist_info_ODSS (RMMC_TMO1.GetId(), RW);
RegisterSpM ( (PFSpMBody) SpM1, & spm_spec);

// register TMO
TMO_RegistParam tmo_spec;
_tcscopy (tmo_spec.global_name, TMO_name);
tmo_spec.start_time = TMO_start_time;
RegisterTMO (& tmo_spec);
}
```

Feb-07	28
--------	----

UCI
DREAM Lab



Example (cont.)

- TMO1.cpp (cont.)

```
int TMO1::SpM1 ()
{
    // Variables for RMMC
    int          degree;
    tms          OfficialReleaseTime;
    int          sizeofEM;
    MicroSec     Timestamp;
    static SM1struct_RMMC1    SM1_RMMC1;
    static EMstruct_RMMC1     EMPara;

    // Start SpM
    degree = ODSS1.Degree;
```

Feb-07	29
--------	----

UCI
DREAM Lab



Example (cont.)

- TMO1.cpp (cont.)

```
// RMMC check any event message
if ( RMMC_TMO1.NonBlockingReceive (&EMPara, &sizeofEM,
    OfficialReleaseTime, Timestamp) == SUCCESS){
    degree -= EMPara.degree;
    TMOSLprintf (_T("Message from TMO2:"));
    TMOSLprintf (EMPara.Message);
}
SM1_RMMC1.degree = degree;
// Update State Message in RMMC1
OfficialReleaseTime = later (tm4_DCS_age (GetCurrentDCSage ()), 10000);
// This message is valid, starting in 10 millisecs from now.
RMMC_TMO1.g_Update (_T("SM1"), &SM1_RMMC1,
    sizeof(SM1struct_RMMC1), OfficialReleaseTime);
TMOSLprintf (_T("1.Current Degree: %d\n"),degree);
// Increment Degree
ODSS1.Degree = ++degree;
// End SpM
return 1;
```

Feb-07	30
--------	----

UCI
DREAM Lab



Example (cont.)

- TMO2.h

```
class TMO2: public CTMOBase
{
public:
    TMO2 (TCHAR *, TCHAR *, tms);
private:
    ODSSClass1          ODSS1;
    int                 SpM2 ();
    CRMMCGateClass     RMMC_TMO2;
};
```

Feb-07	31
--------	----

UCI
DREAM Lab



Example (cont.)

- TMO2.cpp

```
TMO2::TMO2 (TCHAR * TMO_name, TCHAR * sRMMCName,
            tms TMO_start_time)
    :RMMC_TMO2 (sRMMCName)
{
    // register SpM
    AAC aac1 (NULL,          // null for candidate aac label
             tm4_DCS_age (1*1000*1000),    // from
             tm4_DCS_age (60*1000*1000),   // until
             1 * 1000 * 1000,              // every
             0,                             // EST
             100*1000,                      // LST
             200*1000                       // by
    );
```

Feb-07	32
--------	----

UCI
DREAM Lab



Example (cont.)

- TMO2.cpp

```
SpM_RegistParam  spm_spec;
spm_spec.build_regist_info_AAC (aac1);
spm_spec.build_regist_info_ODSS (ODSS1.GetId (), RW);
spm_spec.build_regist_info_ODSS (RMMC_TMO2.GetId (), RW);
RegisterSpM ( (PFSpMBody) SpM1, & spm_spec);
```

Feb-07	33
--------	----

UCI
DREAM Lab



Example (cont.)

- TMO2.cpp (cont.)

```
// register TMO
TMO_RegistParam  tmo_spec;
_tcscopy (tmo_spec.global_name, TMO_name);
tmo_spec.start_time = TMO_start_time;
RegisterTMO (& tmo_spec);
}

int TMO2::SpM2 ()
{
    // Variables for RMMC
    MicroSec      Timestamp;
    tms          OfficialReleaseTime;
    static SM1struct_RMMC1      SM1_RMMC1;
    static EMstruct_RMMC1      EMPara;
    // SpM started
```

Feb-07	34
--------	----

UCI
DREAM Lab



Example (cont.)

- TMO2.cpp (cont.)

```
// RMMC
if ( RMMC_TMO2.Read (_T("SM1"), &SM1_RMMC1, OfficialReleaseTime, Timestamp)
    == SUCCESS)
{
    TMOSLprintf (_T(" 2.Current Degree: %dWn"), SM1_RMMC1.degree);
    if ( SM1_RMMC1.degree >= MAX_DEGREE )
    {
        // Send TMO1 Event Message to decrease the degree
        EMPara.degree = MAX_DEGREE;
        _stprintf (EMPara.Message, _T("Please reduce the degree by %dWn"),
            EMPara.degree);
        OfficialReleaseTime = later (tm4_DCS_age (GetCurrentDCSage ()),15000);
        // This message is valid, starting in 15 millisecs from now.
        RMMC_TMO2.Announce (&EMPara, sizeof (EMPara), OfficialReleaseTime );
    };
}
return 1;
}
```

Feb-07

35

UCI
DREAM Lab

Example (cont.)

- TestTMO.cpp

```
int _tmain (int argc, _TCHAR* argv[])
{
    // Start TMO support Middleware
    StartTMOengine ();

    tms TMO_start_time1 = tm4_DCS_age (2*1000*1000);
    TMO1 T1 (_T("TMO1"), _T("RMMC1"), TMO_start_time1);

    tms TMO_start_time2 = tm4_DCS_age (2*1000*1000);
    TMO2 T2 (_T("TMO2"), _T("RMMC1"), TMO_start_time2);

    MainThrSleep ();

    return 0;
}
```

Feb-07

36

UCI
DREAM Lab

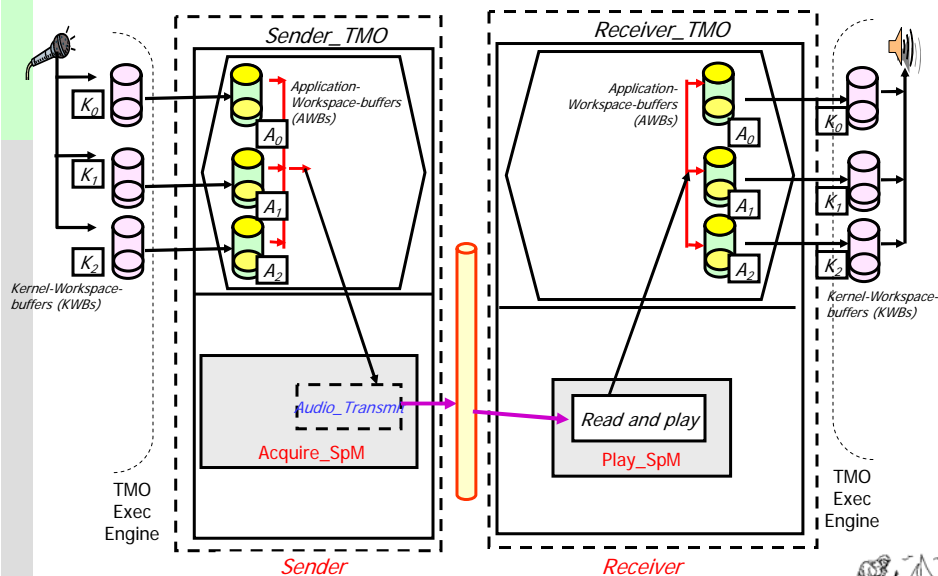
Tele-Audio Application

- This is a multimedia application designed as a TMO network.
- There are two nodes in the system:
 - One records voice with microphone and sends data to the receiver;
 - The other receives the data from the sender and plays back the sound with a speaker.
- Sender has one SpM which reads sampled data from the audio device and sends data packets to the receiver.
- Receiver has one SpM which receives data packets from the sender and writes the received voice data into audio device driver.
- There is one RMMC channel between sender and receiver which facilitates message exchanges.

Feb-07 37



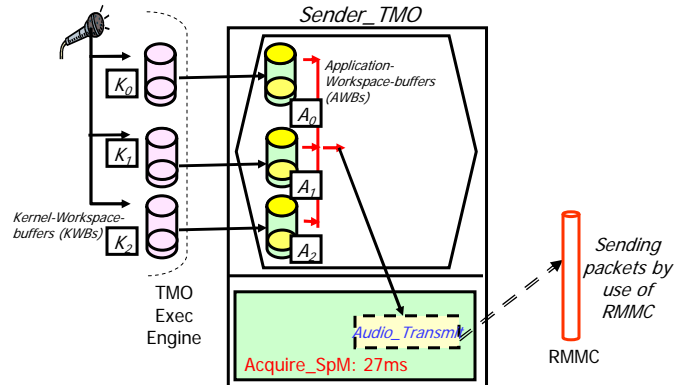
TMO Network for Tele-Audio Application



Feb-07 38



Design of Sender



- In the initialization phase of audio capturing, three application-workspace-buffers (AWBs) are registered into the OS kernel.
- For each AWB, a corresponding kernel-workspace-buffer (KWB) with the same size is allocated in the kernel.
- When the capturing starts, audio device driver saves audio packets into KWBs in a round-robin fashion, i.e., $K_0, K_1, K_2, K_0, K_1, K_2, K_0, \dots$
- To obtain an audio packet from a KWB, its corresponding AWB should be provided.
- In order to obtain audio packets sequentially, the sequence of providing AWBs is $A_0, A_1, A_2, A_0, \dots$

Feb-07 39

UCI DREAM Lab

Wrapper Class for Audio Input Device

Mic_Wrapper_Class (Inherited from ODSSBaseClass)

```

char    AWB [IN_BUFFER] [WAVEIN_AUDIO_BLOCK_SIZE];
        // Application workspace buffer
WAVEFORMATEX  wave_format;    // WaveIn Device Setting Parameter
HWAVEIN      wavein_handle;   // The handle of WaveIn device.
WAVEHDR      wavein_wave_hdr[IN_BUFFER];
        // The headers for each WaveIn buffer.
int    wavein_open_result;    // Status flag on WaveIn open operation.
BOOL   AudioIn_start;        // Flag for microphone start.
    
```

```

Mic_Wrapper_Class ();        // constructor
void    OpenMic ();          // Open the audio input device.
BOOL    IsMicStart ();       // Is it ready to use?
MMRESULT Read (int buffer_number); // Read audio data from device driver.
Char *  GetBufferData (int buffer_number); // Get a pointer to AWB.
    
```

Feb-07 40

UCI DREAM Lab

Wrapper Class for Audio Output Device

Spk_Wrapper_Class (Inherited from ODSSBaseClass)

```

char AWB [OUT_BUFFER] [WAVEIN_AUDIO_BLOCK_SIZE];
    // Application workspace buffer
WAVEFORMATEX wave_format; // WaveOut Device Setting Parameter
HWAVEOUT waveout_handle; // The handle of WaveOut device.
WAVEHDR waveout_wave_hdr[OUT_BUFFER];
    // The headers for each WaveOut buffer.
int waveout_open_result; // Status flag on WaveOut open operation.
BOOL AudioOut_start; // Flag for speaker start.

Spk_Wrapper_Class (); // constructor
void OpenSpk (); // Open the audio output device.
BOOL IsSpkStart (); // Is it ready to use?
void Play (char * waveout_dataPtr, int waveout_size);
    // Method for Playing Audio Frames

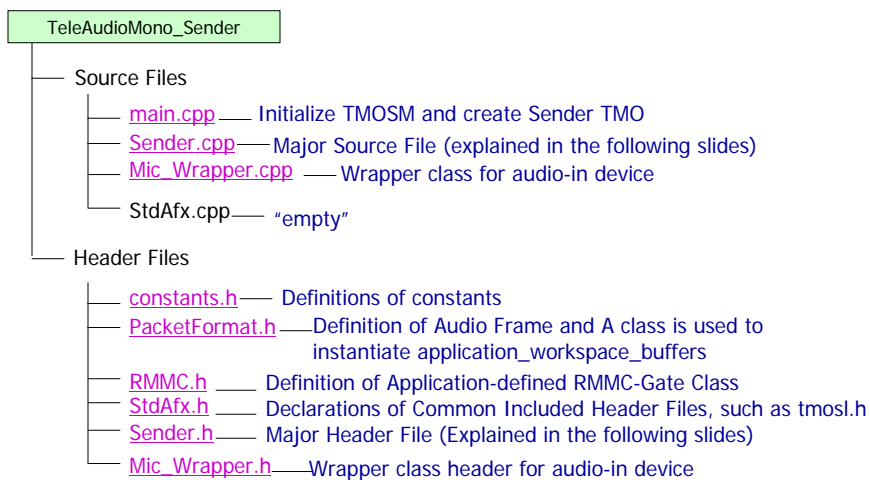
```

Feb-07

41

UCI
DREAM Lab

Visual Studio Project (VSP) Structures



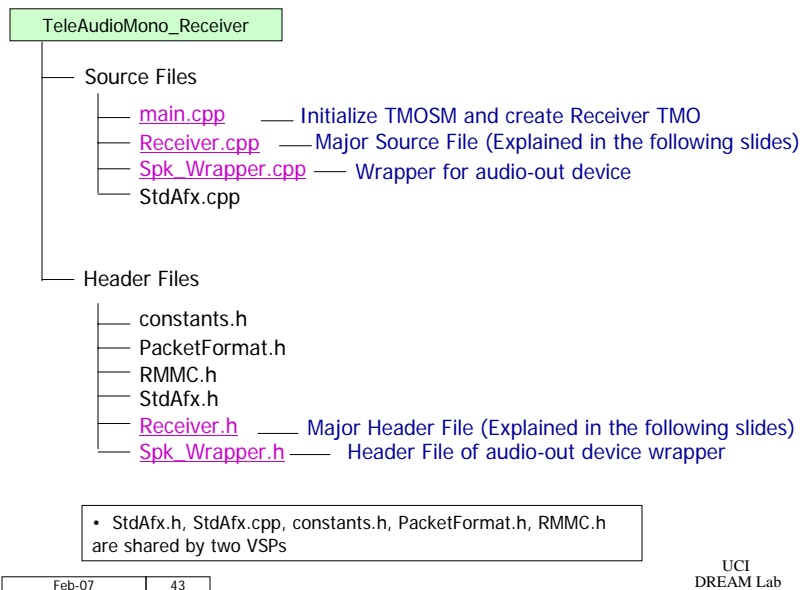
• StdAfx.h, StdAfx.cpp, constants.h, PacketFormat.h, RMMC.h are shared by two VSPs

Feb-07

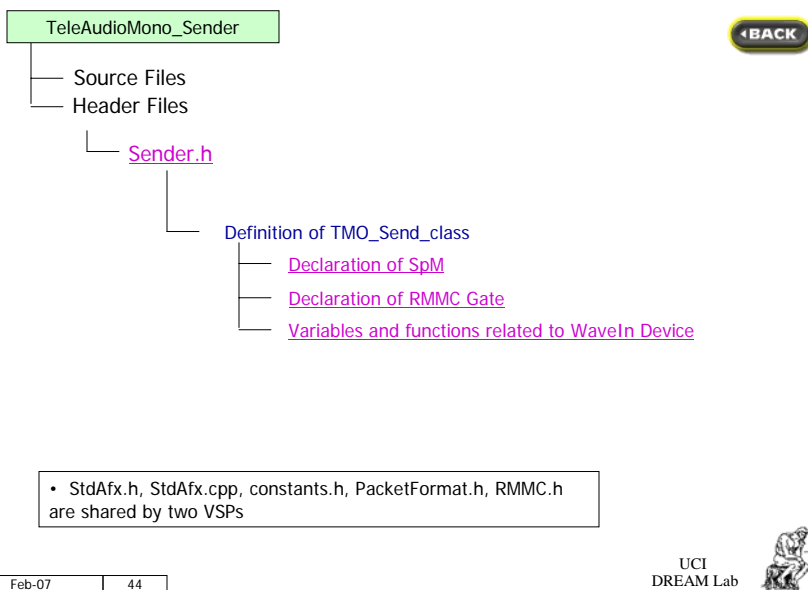
42

UCI
DREAM Lab

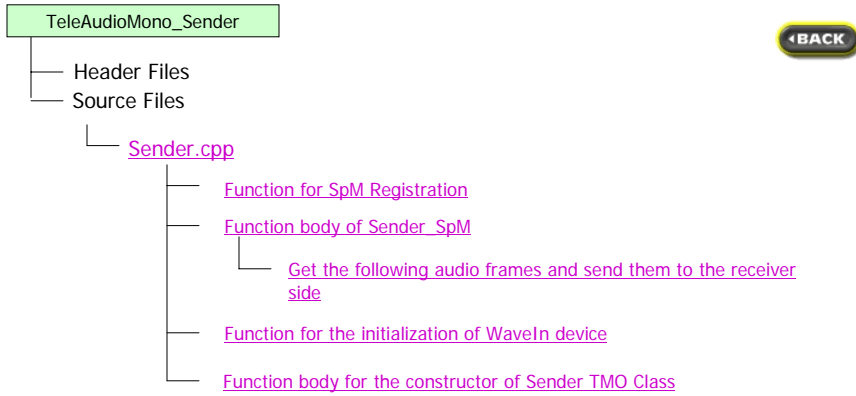
Visual Studio Project (VSP) Structures



TeleAudioMono_Sender::Sender.h



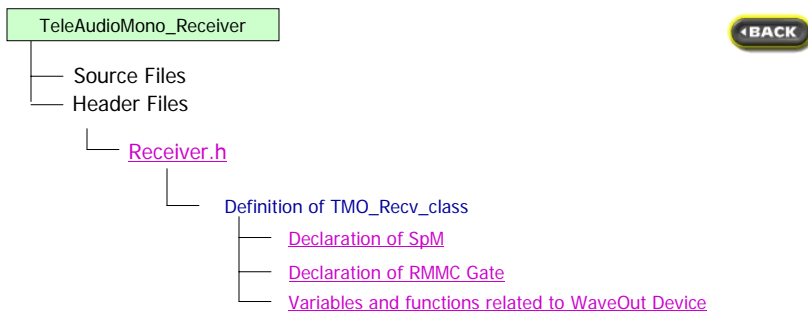
TeleAudioMono_Sender::Sender.cpp



Feb-07 45



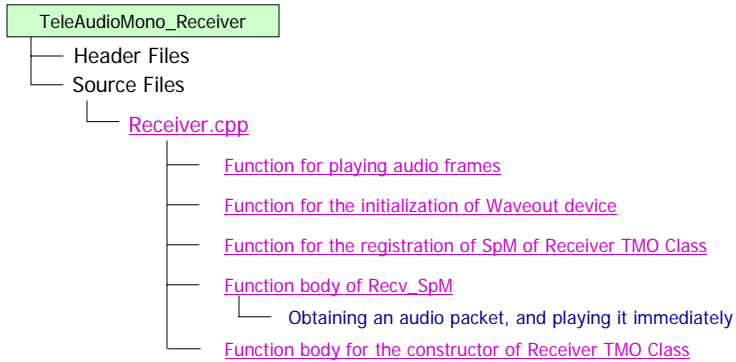
TeleAudioMono_Receiver::Receiver.h



Feb-07 46



TeleAudioMono_Receiver::Receiver.cpp



◀BACK

Feb-07 47

UCI
DREAM Lab



constants.h

```
#ifndef constants_h
#define constants_h

// SpM-related constants

#define WARMUP_DELAY_SECS 5
#define SYSTEM_LIFE_HOURS 24
#define DEALINE_MSEC_RECEIVER_SVM 3

#define SENDER_SPM_PERIOD 27
#define RECEIVER_SPM_PERIOD 27
#define SENDER_SPM_DEADLINE 18
#define RECEIVER_SPM_DEADLINE 18
#define LASTEST_SPM_START_TIME 10
```

◀BACK

Feb-07 48

UCI
DREAM Lab



constants.h (cont')

```
#define SAMPLE_RATE      8000
#define SAMPLE_SIZE      1
#define CHANNEL_NUM      1
#define WAVEIN_AUDIO_BLOCK_SIZE  ((CHANNEL_NUM * SAMPLE_SIZE
 * SAMPLE_RATE * SENDER_SPM_PERIOD)/1000)
    // WaveIn audio block/buffer size.
#define IN_BUFFER        3        // The number of WaveIn buffers.
#define WAVEOUT_AUDIO_BLOCK_SIZE  ((CHANNEL_NUM * SAMPLE_SIZE
 * SAMPLE_RATE * RECEIVER_SPM_PERIOD)/1000)
    // WaveOut audio block/buffer size.
#define OUT_BUFFER 3        // The number of WaveOut buffers
```

// WaveIn & WaveOut
device-related constants

UCI
DREAM Lab



Feb-07

49

PacketFormat.h

```
#ifndef __PACKET_FORMAT_H__
#define __PACKET_FORMAT_H__
```

◀BACK

```
#include "StdAfx.h"
#include "constants.h"
using namespace TMO;
typedef struct
```

// The data structure of a frame

```
{
    unsigned int    aFrameID; // ID of a frame
    MicroSec        aICT;     // Imaginary Capture Timestamp of a frame (not used)
    MicroSec        aTPT;     // Target Play Timestamp of a frame (not used)
    MicroSec        aSend;    // Sending Timestamp of a frame (not used)
    MicroSec        aRecv;    // Receiving Timestamp of a frame (recorded at the
                             // receiver side (not used))
    unsigned char   aData [WAVEIN_AUDIO_BLOCK_SIZE]; // Audio Data
} SAudioFrame;
#endif
```

UCI
DREAM Lab



Feb-07

50

RMMC.h

```

#ifndef RMMC_H
#define RMMC_H
#include "stdafx.h"
#include "PacketFormat.h"

class RMMCGateClass: public RMMCgateBaseClass // RMMCGateClass is inherited
                                                    from RMMCgateBaseClass
{
public:
    RMMCGateClass (TCHAR* RMMC_name)
    {
        // build event message info
        build_regist_info_EM (sizeof (SAudioFrame));

        // RMMC gate Registration
        RegisterRMMCGate (RMMC_name);
    };
};
#endif // RMMC_H

```



Feb-07	51
--------	----

UCI
DREAM Lab



stdAfx.h

```

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently

```



```

// The following part in this slide is automatically generated by Visual Studio

```

```

#ifndef defined
(AFX_STDAFX_H__B1409B15_75F8_11D3_BF6B_00A0CC3FBC6E__INCLUDED_)
#define
AFX_STDAFX_H__B1409B15_75F8_11D3_BF6B_00A0CC3FBC6E__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

```

Feb-07	52
--------	----

UCI
DREAM Lab



stdAfx.h (cont')

// Additional headers program requires

```

#define AUDIO_RMMC_to_SvM
#include <tchar.h>
#include <windows.h>
#ifdef _WIN32_WCE
#include <vfw.h>
#else
#include <Mmsystem.h>
#endif

```

// TMOsl header

```

#include <tmosl.h>
#ifdef _WIN32_WCE
#pragma comment ( lib, "winmm" )
#endif
#endif

```

Feb-07	53
--------	----

UCI
DREAM Lab



Mic_Wrapper.h

```

#ifdef __MIC_WRAPPER_H__
#define __MIC_WRAPPER_H__

```

◀BACK

```

#include "StdAfx.h"
#include "constants.h"
#pragma pack(4)

```

using namespace TMO;

// The class is for microphone, which is inherited from ODSSBaseClass

```

class Mic_Wrapper_Class : public ODSSBaseClass <Mic_Wrapper_Class>
{
public:
    Mic_Wrapper_Class () {};
    ~Mic_Wrapper_Class () {};

```

Feb-07	54
--------	----

UCI
DREAM Lab



Mic_Wrapper.h

```

// Initialize the WaveIn device.
void          OpenMic ();
BOOL         IsMicStart ()      { return AudioIn_start; }
MMRESULT     Read (int  buffer_number);
char*        GetBufferData (int  buffer_number) { return AWB[buffer_number];}

private:
char  AWB [IN_BUFFER] [WAVEIN_AUDIO_BLOCK_SIZE];
WAVEFORMATEX      wave_format;      // WaveIn Device Setting Parameter
HWAVEIN           wavein_handle;     // The handle of WaveIn device.
WAVEHDR           wavein_wave_hdr[IN_BUFFER];
// The headers for each WaveIn buffer.

int               wavein_open_result; // Status flag on WaveIn open operation.
BOOL              AudioIn_start;     // Flag for microphone start.
};
#endif

```

Feb-07	55
--------	----

UCI
DREAM Lab



Mic_Wrapper.cpp

```

#include "Mic_Wrapper.h"

// read kernel buffer
MMRESULT Mic_Wrapper_Class::Read (int  buffer_number)
{
    MMRESULT      result;
    WAVEHDR       *pWaveHdr;

    pWaveHdr = (WAVEHDR*) & wavein_wave_hdr[buffer_number];
    pWaveHdr->dwBufferLength = WAVEOUT_AUDIO_BLOCK_SIZE;

    // Send the input buffer to the input device.
    result = waveInAddBuffer (wavein_handle,
                              pWaveHdr,
                              sizeof (WAVEHDR));

    return result;
}

```

Feb-07	56
--------	----

UCI
DREAM Lab



Mic_Wrapper.cpp

```

/*
 * AudioIn_Open opens the waveform-audio input device for reading input data.
 * It also initializes the buffers used for receiving data from the audio
 * input device.
 */
void Mic_Wrapper_Class::OpenMic ()
{
    // Settings for the WaveIn device.
    memset ( (void *) & wave_format, 0, sizeof (wave_format));
    wave_format.wFormatTag      = WAVE_FORMAT_PCM;
    wave_format.nChannels       = CHANNEL_NUM;
    wave_format.nSamplesPerSec  = SAMPLE_RATE;
    wave_format.nAvgBytesPerSec = SAMPLE_RATE;
    wave_format.nBlockAlign     = 1;
    wave_format.wBitsPerSample  = 8 * SAMPLE_SIZE;
    wave_format.cbSize          = 0;

```

Feb-07

57

UCI
DREAM Lab

Mic_Wrapper.cpp

```

/* Open the waveform-audio input device.*/
wavein_open_result = waveInOpen (
    (HWAVEIN *) & wavein_handle,
    WAVE_MAPPER,
    (WAVEFORMATEX *) & wave_format,
    (DWORD) 0,
    (DWORD) 0,
    CALLBACK_NULL);

/* If device open operation fails, print error message.*/
if (wavein_open_result != MMSYSERR_NOERROR)
{
    wavein_handle = NULL;
    TMOStprintf (_T("*** WaveIn device open failed.***\n"));
    return ;
}

```

Feb-07

58

UCI
DREAM Lab

Mic_Wrapper.cpp

```

/* Initialize the waveform-audio input buffer fields.*/
for (int i = 0; i < IN_BUFFER; i++)
{
    wavein_wave_hdr[i].lpData          = (char *) AWB[i];
    wavein_wave_hdr[i].dwBufferLength  = WAVEIN_AUDIO_BLOCK_SIZE;
    wavein_wave_hdr[i].dwUser          = 1;
    wavein_wave_hdr[i].dwFlags         = 0;

    /* Prepare a waveform-audio input.*/
    waveInPrepareHeader (wavein_handle, & wavein_wave_hdr[i], sizeof
(WAVEHDR) );
}

/* Starts input on the given waveform-audio input device.*/
MMRESULT result = waveInStart (wavein_handle);

```

Feb-07

59

UCI
DREAM Lab

Mic_Wrapper.cpp

```

/* Check for the return code.*/
switch (result) {
    case MMSYSERR_INVALIDHANDLE:
        TMOSLprintf ( _T("waveInStart: MMSYSERR_INVALIDHANDLE"));
        break;
    case MMSYSERR_NODRIVER:
        TMOSLprintf ( _T("waveInStart: MMSYSERR_NODRIVER"));
        break;
    case MMSYSERR_NOMEM:
        TMOSLprintf ( _T("waveInStart: MMSYSERR_NOMEM"));
        break;
    case MMSYSERR_NOERROR:
        break;
}
AudioIn_start = true;
return;
}

```

Feb-07

60

UCI
DREAM Lab

Sender.h

```
#ifndef __SENDER_H__
#define __SENDER_H__

#include "StdAfx.h"
#include "PacketFormat.h"
#include "RMMC.h"
#include "Mic_Wrapper.h"

using namespace TMO;
```



Feb-07

61

UCI
DREAM Lab

Sender.h (con't)

```
// class : TMO_Class
class TMO_Send_Class: public CTMOBase
{
private:
    int          Sender_SpM (); // SpM body
    void         Sender_SpM_Register_Init (); // SpM Initialization Method

    RMMCGateClass    RMMC_1; // RMMC gate
    Mic_Wrapper_Class m_MIC; // wrapper object for audio-in device
```



Feb-07

62

UCI
DREAM Lab

Sender.h (con't)

```

__int64    A_nMessageID;    //Frame counter
__int64    SpM_iteration; // Counter for SpM iteration

unsigned char    LastBufferCount; // Auxiliary variables

public:
    TMO_Send_Class (TCHAR *, tms &);
};

#endif

```

◀BACK

Feb-07

63

UCI
DREAM Lab

main() (Sender)

```

#include "stdafx.h"
#include "TMO_Send.h"
void main ()
{
    // Start TMO support Middleware
    StartTMOengine ();
    tms    start_time = tm4_DCS_age ( 5*1000*1000);

    TMO_Send_Class* pTMO_Send = new
        TMO_Send_Class (_T("TMO_Send"), start_time);

    // Main thread goes to sleep and TMO SM takes the control
    MainThrSleep ();
}

```

An alternative way to instantiate an TMO obj:
 TMO_Send_Class SenderTMO
 (_T("TMO_Send"), start_time);

◀BACK

Feb-07

64

UCI
DREAM Lab

Sender.cpp

```
#include <iostream>

#include "Sender.h"
#include "mmreg.h"
#include "Mmsystem.h"

using namespace TMO;

void TMO_Send_Class::Sender_SpM_Register_Init ()
{
    LastBufferCount = 0;
```



Feb-07	65
--------	----

UCI
DREAM Lab



Sender.cpp (con't)

```
A_nMessageID = 0;
SpM_iteration = 0;
SpM_RegistParam Sender_SpM_spec;
```



//specify time window for SpM

```
MicroSec from = (MicroSec) WARMUP_DELAY_SECS * 1000 * 1000;
MicroSec until = (MicroSec) SYSTEM_LIFE_HOURS * 60 * 60 * 1000 * 1000;
MicroSec every = (MicroSec) SENDER_SPM_PERIOD * 1000;
MicroSec est = 0;
MicroSec lst = est + LASTEST_SPM_START_TIME * 1000;
MicroSec by = SENDER_SPM_DEADLINE * 1000;
```

Fill parameters for AAC

Feb-07	66
--------	----

UCI
DREAM Lab



Sender.cpp (con't)

```
AAC aac1 (
    NULL, // null for candidate aac label
    tm4_DCS_age ((MicroSec)(WARMUP_DELAY_SECS * 1000 * 1000)) ,
    tm4_DCS_age ((MicroSec)(20 * 60 * 1000 * 1000)),
    every,
    est,
    lst,
    by
); // The instantiation of AAC obj
```

```
Sender_SpM_spec.build_regist_info_AAC (aac1);
// Register RMMC gate as an ODSS // Sender SpM Registration
Sender_SpM_spec.build_regist_info_ODSS (RMMC_1.GetId(), RW);
// Register Application-Workspace-buffer as ODSS
Sender_SpM_spec.build_regist_info_ODSS (m_MIC.GetId(), RW);
// register SpM
if (RegisterSpM ((PFSpMBody) Sender_SpM, &Sender_SpM_spec) == FAIL)
    TMOSLprintf (_T("Fail to register Sender_SpM object Wn"));
}
```

Feb-07	67
--------	----

UCI
DREAM Lab



Sender.cpp (con't)

```
int TMO_Send_Class::Sender_SpM ()
```

```
{
```

```
MMRESULT result;
int i;
```

// Temporary
variables used in
SpM

◀BACK

Feb-07	68
--------	----

UCI
DREAM Lab



Sender.cpp (con't)

// If the waveform-audio input devices has been initialized.

if (m_MIC.IsMicStart ())

{

```
for (i = 0; i < IN_BUFFER; i++)
{
    if (LastBufferCount + 1 == IN_BUFFER)    {
        result = m_MIC.Read (LastBufferCount + 1 - IN_BUFFER);
    }
    else {
        result = m_MIC.Read (LastBufferCount + 1);
    }
}
```

// Get an audio frame from a buffer

◀BACK

Feb-07

69

UCI
DREAM Lab



Sender.cpp (con't)

if (result == MMSYSERR_NOERROR) {

```
if (LastBufferCount + 1 == IN_BUFFER) {
    memcpy (AudioFrame.aData, m_MIC.GetBufferData (LastBufferCount + 1 -
    IN_BUFFER), WAVEIN_AUDIO_BLOCK_SIZE);
    LastBufferCount = LastBufferCount + 1 - IN_BUFFER;
} else {
    memcpy (AudioFrame.aData, m_MIC.GetBufferData (LastBufferCount + 1),
    WAVEIN_AUDIO_BLOCK_SIZE);
    LastBufferCount++;
}
AudioFrame.aFrameID = A_nMessageID;
TMOSSLprintf (_T("From Buffer %d for %dWn"), LastBufferCount, A_nMessageID);
```

// Prepare audio frame to be sent out

Feb-07

70

UCI
DREAM Lab



Sender.cpp (con't)

```

    if (!RMMC_1.Announce ((void *)& AudioFrame, sizeof (SAudioFrame)) == SUCCESS)
        TMOSLprintf (_T("Fail to send Audio Signal %d SuccessfullyWn"),
                    AudioFrame.aFrameID);
    else
        TMOSLprintf (_T("Success to Send Audio Signal %dWn"),
                    AudioFrame.aFrameID);
    A_nMessageID++;
}
else
    break;
}
}
SpM_iteration++;

return 1;
}

```

// Send out the frames

Feb-07

71

UCI
DREAM Lab

Sender.cpp (con't)

```

TMO_Send_Class::TMO_Send_Class (TCHAR* TMO_name, tms& start_time)
    :RMMC_1 (_T("RMMC_1"))
{
    /* Initialize the WaveIn device.*/
    m_MIC.OpenMic ();

    //register Video_SpM
    Sender_SpM_Register_Init ();

    //register TMO
    TMO_RegistParam TMO_Send_spec;
    _tcscpy (TMO_Send_spec.global_name, TMO_name);
    TMO_Send_spec.start_time = start_time;
    RegisterTMO (&TMO_Send_spec);
}

```

Feb-07

72

UCI
DREAM Lab

Receiver.h

```
#ifndef __RECEIVER_H__
#define __RECEIVER_H__

#include "PacketFormat.h"
#include "RMMC.h"
#include "Spk_Wrapper.h"

using namespace TMO;
```

◀BACK

Feb-07

73

UCI
DREAM Lab



Receiver.h (con't)

```
// class : TMO_Class
class TMO_Recv_Class: public CTMOBase
{
private:
    int Recv_SpM (); // SpM method
    RMMCGateClass RMMC_1; // RMMC gate
    Spk_Wrapper_Class m_SPK; // wrapper object for audio-out device

    void Recv_SpM_Register_Init (); // SpM initialization method
```

◀BACK

Feb-07

74

UCI
DREAM Lab



Receiver.h (con't)

```
__int64      SpM_iteration;    // Counter for Recv_SpM iterations
unsigned int  A_nMessageID;

// Auxiliary variables
```

```
public:
    TMO_Recv_Class (TCHAR *, tms &);
};
#endif
```

◀BACK

UCI
DREAM Lab



Feb-07

75

Spk_Wrapper.h

```
#ifndef __SPK_WRAPPER_H__
#define __SPK_WRAPPER_H__
```

◀BACK

```
#include "StdAfx.h"
#include "constants.h"
#pragma pack(4)
```

```
using namespace TMO;
```

```
// The class is for speaker, which is inherited from ODSSBaseClass
```

```
class Spk_Wrapper_Class : public ODSSBaseClass <Spk_Wrapper_Class>
{
public:
    Spk_Wrapper_Class () {};
    ~Spk_Wrapper_Class () {};
```

UCI
DREAM Lab



Feb-07

76

Spk_Wrapper.h

```

// Initialize the WaveOut device.
void      OpenSpk ();
BOOL      IsSpkStart () { return AudioOut_start;}
void      Play (char * waveout_dataPtr, int waveout_size);
          // Method for Playing Audio Frames

private:
char  AWB [OUT_BUFFER] [WAVEIN_AUDIO_BLOCK_SIZE];

WAVEFORMATEX  wave_format;    // WaveOut Device Setting Parameter
HWAVEOUT      waveout_handle; // The handle of WaveOut device.
WAVEHDR       waveout_wave_hdr[OUT_BUFFER];
              // The headers for each WaveOut buffer.

int           waveout_open_result; // Status flag on WaveOut open operation.
BOOL         AudioOut_start;      // Flag for speaker start.
};
#endif

```

Feb-07

77

UCI
DREAM Lab

Spk_Wrapper.cpp

```

#include "Spk_Wrapper.h"

// Function for playing audio frames
void Spk_Wrapper_Class::Play (char *waveout_dataPtr, int waveout_size)
{
    int buf = 0;
    if (!waveout_open_result) return;
    int k=0;
    // get a new free WAVEHDR buffer
    for (buf = 0; buf < OUT_BUFFER; buf++)
    {
        DWORD flag = waveout_wave_hdr[buf].dwFlags;
        // check if a wave out buffer is empty
        if (flag == (WHDR_DONE | WHDR_PREPARED) || flag == WHDR_PREPARED) {
            memcpy ( waveout_wave_hdr[buf].lpData, (char *) (waveout_dataPtr), waveout_size);
            waveout_wave_hdr[buf].dwUser = 1;
            waveout_wave_hdr[buf].dwBufferLength = waveout_size;
            int res = waveOutWrite (waveout_handle, &waveout_wave_hdr[buf], sizeof (WAVEHDR));
        }
    }
}

```

Feb-07

78

UCI
DREAM Lab

Spk_Wrapper.cpp (con't)

```

switch      (res)  {
  case MMSYSERR_INVALIDHANDLE:
    TMOSLprintf (_T("waveOutWrite MMSYSERR_INVALIDHANDLEWn"));
    break;
  case MMSYSERR_NODRIVER:
    TMOSLprintf (_T("waveOutWrite MMSYSERR_NODRIVERWn"));
    break;
  case MMSYSERR_NOMEM:
    TMOSLprintf (_T("waveOutWrite MMSYSERR_NOMEMWn"));
    break;
  case WAVERR_UNPREPARED:
    TMOSLprintf (_T("waveOutWrite WAVERR_UNPREPAREDWn"));
    break;
  case MMSYSERR_NOERROR:
    break;
}
} // if
} // for

```

// All kinds of error messages when failing to play out audio frames

Feb-07

79

UCI
DREAM Lab

Spk_Wrapper.cpp (con't)

```

/*
 * AudioOut_Open opens the waveform-audio output device for playback.
 * It also initializes the buffers used for sending data to the audio
 * output device.
 */
void Spk_Wrapper_Class::OpenSpk ()
{
  int          i;
  int          result;
  WAVEFORMATEX waveout_wave_format;

```

Feb-07

80

UCI
DREAM Lab

Spk_Wrapper.cpp (con't)

/ Settings for waveform-audio output device.*/*

```
memset ( (void*)& waveout_wave_format, 0, sizeof (waveout_wave_format));
waveout_wave_format.wFormatTag    = WAVE_FORMAT_PCM;
waveout_wave_format.nChannels     = CHANNEL_NUM;
waveout_wave_format.nSamplesPerSec = SAMPLE_RATE;
waveout_wave_format.nAvgBytesPerSec = SAMPLE_RATE;
waveout_wave_format.nBlockAlign   = 1;
waveout_wave_format.wBitsPerSample = 8 * SAMPLE_SIZE;
waveout_wave_format.cbSize        = 0;
```

/ Open the waveform-audio output device.*/*

```
result = waveOutOpen (
    (HWAVEOUT *) &waveout_handle,
    WAVE_MAPPER,
    (WAVEFORMATEX *) &waveout_wave_format,
    (DWORD) 0,
    (DWORD) 0,
    CALLBACK_NULL );
```

Feb-07

81

UCI
DREAM Lab

Spk_Wrapper.cpp (con't)

/ If device open operation fails, print error message.*/*

```
if (result != MMSYSERR_NOERROR) {
    waveout_handle = NULL;
    TMOStprintf (_T("WaveOut device open failed.\r\n"));
    return ;
}
```

/ Initialize the waveform-audio output buffer fields.*/*

```
for (i = 0; i < OUT_BUFFER; i++) {
    waveout_wave_hdr[i].lpData          = AWB[i];
    waveout_wave_hdr[i].dwBufferLength  = WAVEOUT_AUDIO_BLOCK_SIZE;
    waveout_wave_hdr[i].dwUser          = 0;
    waveout_wave_hdr[i].dwFlags         = 0L;
    /* Prepare a waveform-audio data block for playback.*/
    waveOutPrepareHeader (waveout_handle, &waveout_wave_hdr[i], sizeof (WAVEHDR));
}
```

```
AudioOut_start = 1; // Indicating the start-up of WaveOut device
return;
```

}

Feb-07

82

UCI
DREAM Lab

main() (Receiver)

```
#include "stdafx.h"
#include "TMO_Recv.h"

void main ()
{
    // Start TMO support Middleware
    StartTMOEngine ();

    tms    start_time = tm4_DCS_age ( 5*1000*1000);

    TMO_Recv_Class* pTMO_Recv = new TMO_Recv_Class (_T("TMO_Recv"), start_time);

    // Main thread goes to sleep and TMOSM takes the control
    MainThrSleep ();
}
```

[◀BACK](#)

Feb-07

83

UCI
DREAM Lab

Receiver.cpp

```
#include <iostream>
#include "Receiver.h"
#include "mmreg.h"
#include "Mmsystem.h"
```

[◀BACK](#)

Feb-07

84

UCI
DREAM Lab

Receiver.cpp (con't)



```
// Initialize SpM
void TMO_Recv_Class::Recv_SpM_Register_Init ()
{
    SAudioFrame          AudioFrame;
    CircleLength =  AUDIO_CIRCLE_LENGTH - 3;
    SpM_iteration = 0;
```

```
    SpM_RegistParam  Recv_SpM_spec;
```

```
    MicroSec every = (RECEIVER_SPM_PERIOD ) * 1000;
    MicroSec est   = 0;
    MicroSec lst   = est + (LASTEST_SPM_START_TIME ) * 1000;
    MicroSec by    = (RECEIVER_SPM_DEADLINE ) * 1000;
```

Fill parameters for AAC

Feb-07	85
--------	----

UCI
DREAM Lab



Receiver.cpp (con't)

```
AAC * aac1 = new AAC (
    NULL, // null for candidate aac label
    tm4_DCS_age (9 * 500 * 1000),
    tm4_DCS_age ((MicroSec) (20 * 60 * 1000 * 1000)),
    every, est, lst, by );
Recv_SpM_spec.build_regist_info_AAC (*aac1);
```

The instantiation
of AAC object

```
//register RMMC gate as an ODSS
Recv_SpM_spec.build_regist_info_ODSS (RMMC_1.GetId(), RW);
//register Application-Workspace-buffer as ODSS
Recv_SpM_spec.build_regist_info_ODSS (m_SPK.GetId(), RW);

// register SpM
if (RegisterSpM ( (PFSpMBody) Recv_SpM, &Recv_SpM_spec) == FAIL)
    TMOSLprintf ("Fail to register Recv_SpM object Wn");
}
```

Recv_SpM
registration

Feb-07	86
--------	----

UCI
DREAM Lab



Receiver.cpp (con't)

```

int TMO_Recv_Class::Recv_SpM ()
{
    int      AudioMsgSize;
    int      result;
    tms      OfficialReleaseTime;
    MicroSec timestamp;

    if (!m_SPK.IsSpkStart ()) return; // wait until speaker init is done
  
```



// Temporary
variables used in
SpM

Feb-07	87
--------	----

UCI
DREAM Lab



Receiver.cpp (con't)

```

while (1)
{
    // Get frames from RMMC channel by calling NonBlockingReceive until returning
    // there is no more frame in the RMMC channel
  
```



```

    result = RMMC_1.NonBlockingReceive ( (void *) &AudioFrame,
                                         &AudioMsgSize, OfficialReleasTime, timestamp);
    if (result != SUCCESS)
    {
        if (result == NO_VALUE)
        {
            TMOSLprintf (_T("NO_VALUEWn"));
        }
        else if (result == FAIL)
            TMOSLprintf (_T("FAILWn"));
        break;
    }
  
```

Feb-07	88
--------	----

UCI
DREAM Lab



Receiver.cpp(con't)

```

else //Successfully get a frame from RMMC channel
{
    TMOSLprintf (_T("TPT %dWn"), AudioFrame.aFrameID);
    m_SPK.Play ( (char*) AudioFrame.aData,
                WAVEOUT_AUDIO_BLOCK_SIZE);
}
}
SpM_iteration++;
return 1;
}

```

play an audio packet.

Feb-07

89

UCI
DREAM Lab

Receiver.cpp (con't)

```

TMO_Recv_Class::TMO_Recv_Class (TCHAR* TMO_name, tms& start_time)
: RMMC_1 (_T("RMMC_1"))
{
    //Initialize audio output device
    m_SPK.OpenSpk ();

    //Initialize SpM
    Recv_SpM_Register_Init ();

    TMO_RegistParam TMO_Recv_spec;
    _tscopy (TMO_Recv_spec.global_name, TMO_name);
    TMO_Recv_spec.start_time = start_time;

    //Register TMO
    RegisterTMO (& TMO_Recv_spec);
};

```

Feb-07

90

UCI
DREAM Lab