Chapter 4 IPMP-X Software Reference

In this chapter, we will describe IPMP-X Software Reference for its implementation including a few key components. The major key components contain Message Router, Tool Manager, Control Point and Tool Framework

The IPMP-X Terminal Software is mostly developed by Craig A.Schutz and based on IM1 system. IM1,the AHG on System Reference Software Implementation , is a group that is responsible for the development and integration of the MPEG-4 system software.

4.1 Building the Reference Software

The reference software codes are developed with Microsoft Visual C++. To build the application, users could use Microsoft Visual Studio to open the workspace file IM1.dsw under _WorkSpace directory. This will result in projects being opened. We will compile projects to build the application.

To run the application, users need to take the following actions.

(1) Open the workspace IM1.dsw in the directory \IM1_Workspaces.

(2) To add the tool for the IPMP reference software program, the IPMP-X software reference implementation will load the IPMP Tools present in the directory \IM1\Debug on the basis of a string of bytes appended at the end of their DLL file after these are built (add the command AppendBin.bat, refer to example IPMP_MasterTool\AppendBin.bat). For more information on how to compose this string of bytes have a look at IPMPServicesFull::CheckDllForTool method in the file \IPMPXFull\IPMPXFull.cpp

(3) Select the necessary modules to build. The basic necessary modules contain

IM1_2D , Bifsenc,Mux ,IPMPXFull and your tool module.

(4) Edit the file \IM1_Registry\im1.reg . If is not present the necessary entry,(Ex : “IPMPSystem =”IPMPFull.dll”), add and update your registry

(2) Copy the necessary decoder module from IM1\IM1Decoders\*.dll to the the IM1\Debug directory

(3) Prepare a multiplexed MPEG-4 IPMPX transport stream

The steps are as below:

1. Open the DOS window and change cd to the \IM1\Debug
2. Type BifsEnc(has created BifsEnc.exe in build module) IPMPX.txt
3. Type Mux(has created Mux.exe in build module) IPMPX.scr
4. After step 3 finishes, IPMPX.trif file is created.

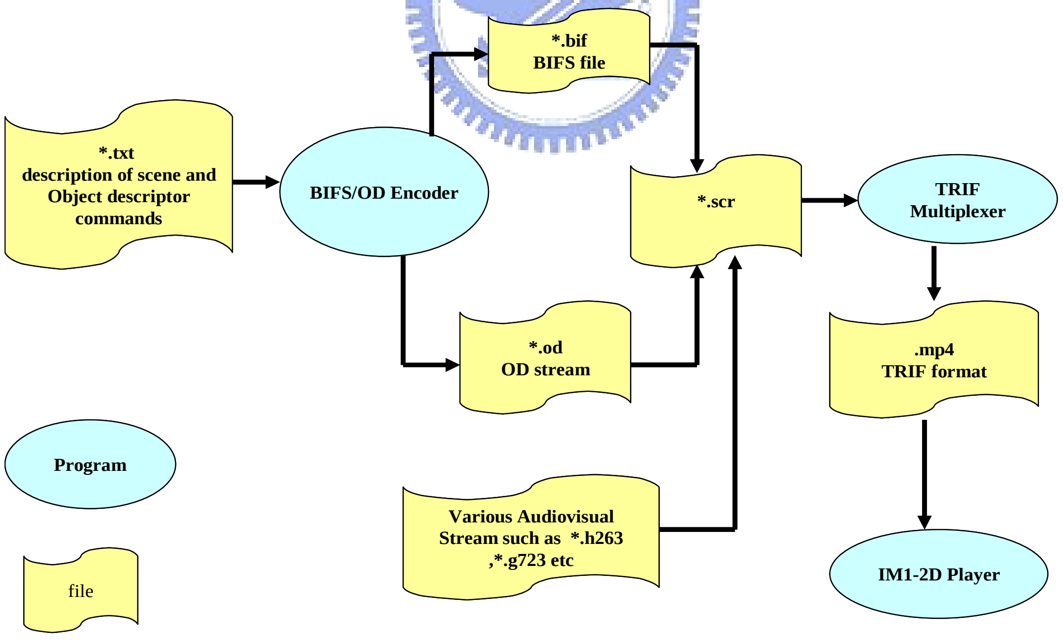


Figure 4-1 IM1 Development Environment

(4) Launch IM1-2D application

4.2 Software Reference Architecture

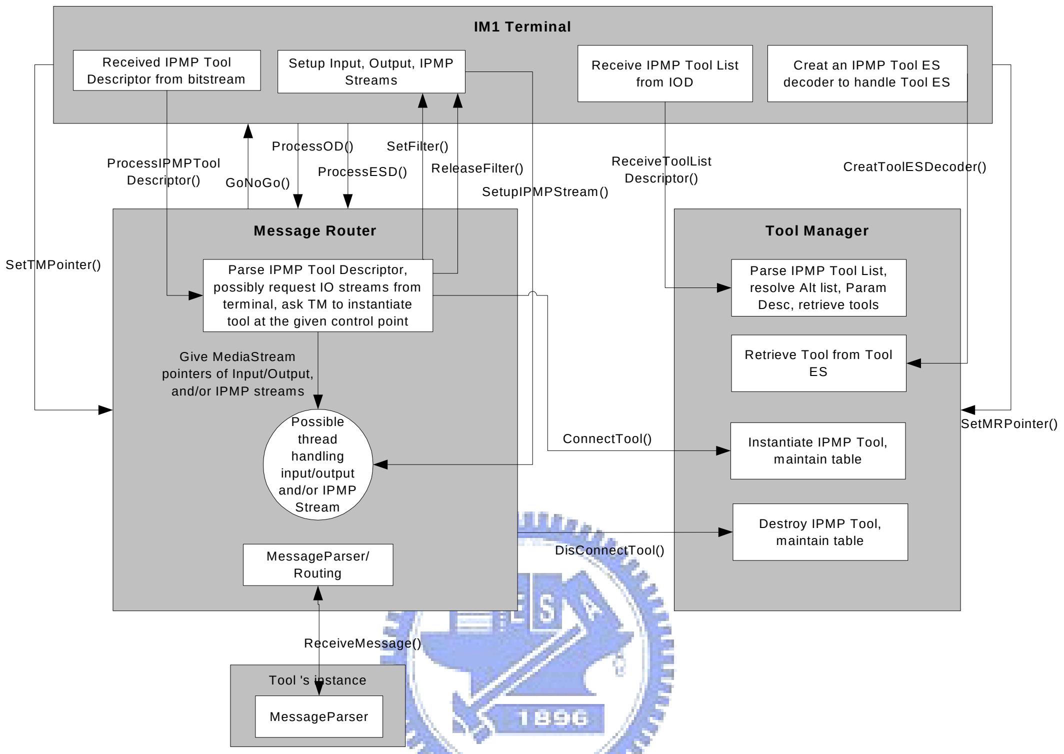

IM1 Terminal
Received IPMP Tool Descriptor from bitstream
Setup Input, Output, IPMP Streams
Receive IPMP Tool List from IOD
Creat an IPMP Tool ES decoder to handle Tool ES
ProcessIPMPTool Descriptor()
GoNoGo()
ProcessOD()
ProcessESD()
SetFilter()
ReleaseFilter()
SetupIPMPStream()
ReceiveToolList Descriptor()
CreatToolESDecoder()
SetTMPointer()
Message Router
Parse IPMP Tool Descriptor, possibly request IO streams from terminal, ask TM to instantiate tool at the given control point
Give MediaStream pointers of Input/Output, and/or IPMP streams
Possible thread handling input/output and/or IPMP Stream
MessageParser/ Routing
Tool Manager
Parse IPMP Tool List, resolve Alt list, Param Desc, retrieve tools
Retrieve Tool from Tool ES
Instantiate IPMP Tool, maintain table
Destroy IPMP Tool, maintain table
ConnectTool()
DisConnectTool()
SetMRPointer()
ReceiveMessage()
Tool 's instance
MessageParser

Figure 4-2 MPEG-4 with IPMP-X Terminal, [3]

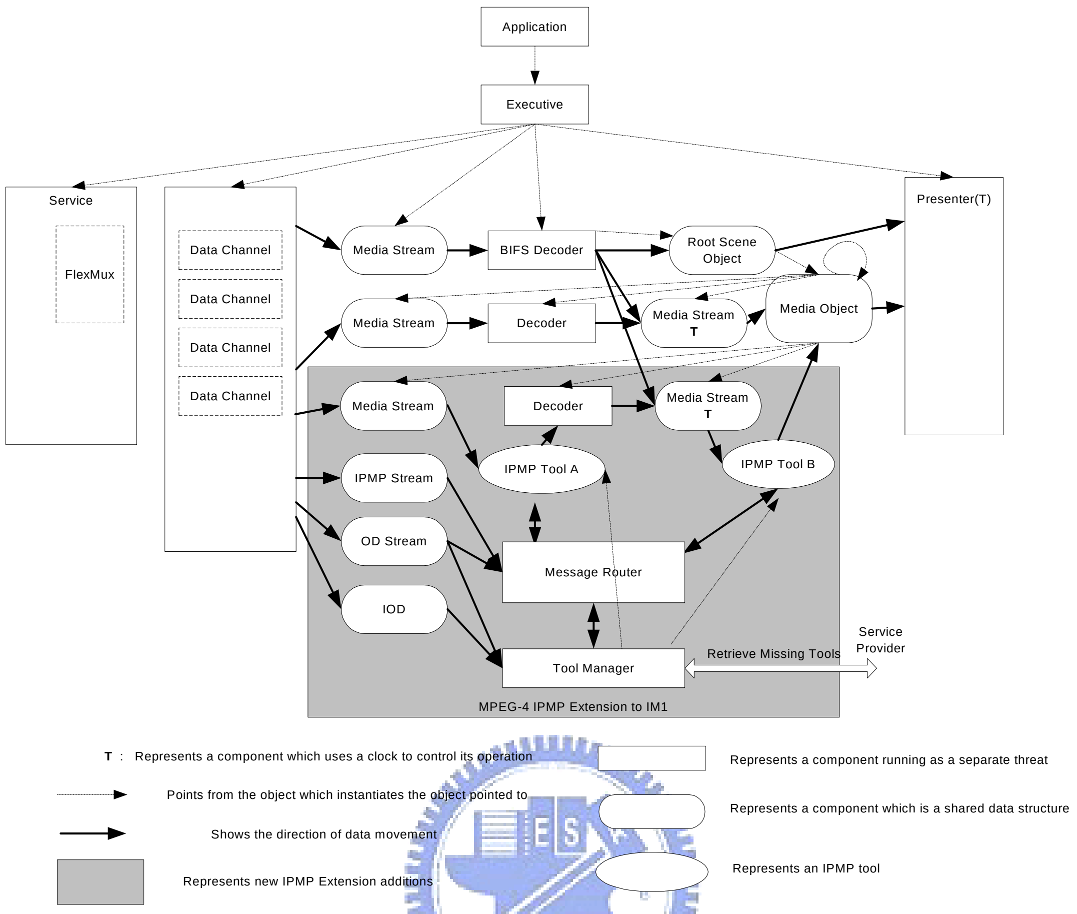


Figure 4-3 Architecture of IM1 with IPMP-X, [8]

The figure 4-1and figure 4-2 are supported by IM1.We can roughly understand flow control of the architecture of IM1 with IPMP-X from Fig 4-1. We can find that it is the multithread architecture(the symbol □ of this figure represents the thread). We must point out here that IPMP-X Box including Message Router, Tool Manager , Tool is the same as the concept of the IPMP-X Specification we described in chapter 3.

Fig 4-1 emphasizes the implementation of IPMP-X Terminal supported by IM1. We trace this reference code and describe it clearly in next section.

4.3 Description of the Key Components

4.3.1 Instantiation of the IPMP System

The top level object of the IPMP-X DLL Instantiated into an IM1 terminal by the following code :

static ZRegistry registry ("IPMPX\System");

m_pIPMPX= (IPMPSystem *)registry.CreateInstance("IPMPSystem", m_hIPMPX);

The pathname of an IPMPX DLL must be defined in the registry as the value of the key [KEY_CURRENT_USER\Software\MPEG-4\Im1\IPMPX]

The IPMPX DLL must include the following code :

```
extern "C" IPMPSystem* CreateInstance ()
      {
            return new IPMPSystem
   }
```

4.3.2 The Description of the Message Router Implementation

The software reference implements the Message Router function use the many polymorphism techniques of Object Orientated Language. The Message Router interface is an abstract class that is instantiated by IPMPServicesFull object. We highlight the class relationship of Message Router and interface of Message Router in scheme 4A as below.

(1) Class Relationship of The Message Router

$$\begin{tikzcd}
 & \text{IPMPX\_Tool} \arrow[r] \arrow[rd] & \text{Tool A} \\
 & & \text{Tool B} \\
\text{IPMPTool\_Interface} \arrow[ruu] \arrow[r] \arrow[rd] & \text{MessageDistributor} & \\
 & \text{MR\_Interface} \arrow[r] & \text{IPMPService}
\end{tikzcd}$$

(2) the Message Router Interface

The Message Router Interface is defined in class MR_Interface.

Scheme 4A:

```
class MR_Interface : public IPMPTool_Interface
    {
    public:
        /*This constants enumerates the possible access permission of an Elementary Stream
        */
        enum ACCESS_PERMISSION
        {
        ALLOWED = 1,   // terminal can proceed processing the stream
        DENIED = 2,            // don't process and discard the stream
        PROTECTED = 3        //wait for a GO
        };
        virtual bool SetTMPointer( TM_Interface * tmPointer) = 0;
       virtual bool ProcessObjectDescriptor( ObjectDescriptor* pOD ) = 0;
        virtual bool RemoveObjectDescriptor( ObjectDescriptor* pOD ) = 0;
        virtual bool ProcessIPMPDescriptor(IPMP_Descriptor* descriptor) =0;
        virtual bool RemoveIPMPDescriptor(IPMP_Descriptor*descriptor)= 0;
        ProcessESDescriptor(ES_Descriptor* pESD) = 0;
         virtual bool RemoveESDescriptor( ES_Descriptor* pESD ) = 0;
        virtual bool SetUpIPMPStream(ObjectDescriptor    *pOD,ES_Descriptor* pESD,
        MediaStream* ipmpStream) = 0;
        virtual bool RemoveIPMPStream(ES_Descriptor *pESD) = 0;
    };
```

After we have described about the class and interface of the Message Router, we will discuss the implementation of Message Router. There are three key points for the Message Router implementation in the Software Reference program.

(1) We know that class MR_Interface is an abstract class. It is instantiated by IPMPServicesFull object. So,the Message Router interface function is implemented by IPMPServicesFull object. The detailed content refers to the Software Reference Program.

(2) After the InstanceTool() function is called by the Message Router to request a tool

be instantiated, the Message Router may transfers IPMP information to these tools.

(3) The Message Router transfers IPMP information.

Scheme 4B:

```
class IPMPTool_Interface
{
public:
        virtual bool ReceiveMessage(int size, unsigned char* message) = 0;
};
class MessageDistributor : public IPMPTool_Interface
{
public:
…
bool ReceiveMessage ( ToolMessage * msg );
….
};

class IPMPX_Tool : public IPMPTool_Interface
{…
virtual bool ReceiveMessage( ToolMessage* ) = 0;
…
}
class IPMP_ToolA : public IPMPX_Tool
{
….
// IPMPTool_Interface interface methods.
bool ReceiveMessage( ToolMessage* );
….
}
```

The ReceiveMessage(ToolMessage *msg) in scheme 4B that is defined in many different classes. How does it work is decided by which object instantiates it. Its parameter may carry various kinds of IPMP Information.

As below, the two different cases describe IPMP information to transfer either from the terminal to the tool or the too to the terminal on the software reference.

(1) m_MessageDistributor.ReceiveMessage(&msg):

The m_MessageDistributor.ReceiveMessage(&msg) indicates that the Message Router will transfer IPMP Information from terminal to the specified tool. The m_MessageDistibutor is instantiated by MessageDistributor object.

(2) m_pMRInterface->ReceiveMessage(&msg):

The m_pMRInterface->ReceiveMessage(&msg) indicates that the Message Router will transfer IPMP Information from the specified tool to terminal. The m_ pMRInterface is instantiated by the specified tool object.

4.3.3 The Description of Tool Manager Implementation

The software reference implements the Tool Manager function using the many polymorphism technique of Object Orientated Language. The Tool Manager interface is an abstract class that is instantiated by IPMPServicesFull object. So, the Tool Manager interface function is implemented by IPMPServicesFull object. The condition is the same as the Message Router. We highlight the class relationship of the Tool Manager and interface of Tool Manager in scheme 4C as below.

(1) Class Relationship for the Tool Manager

$$\begin{tikzcd}
\text{TM\_Interface} \arrow[r] & \text{IPMPService} \arrow[r] \arrow[dr] & \text{IPMPServicesFull} \\
 & & \text{IPMPServicesTriv}
\end{tikzcd}$$

(2) the Tool Manager Interface

The Tool Manager Interface is defined in the class TM_Interface

Scheme 4C:

```
class TM_Interface
{
public:
```

```
virtual bool SetMRPointer( MR_Interface * tmPointer) = 0;
virtual bool DisconnectTool( void* toolPtr) = 0;
virtual void* ConnectTool(ES_Descriptor *pESD, PMPToolDescriptor* toolDescriptor) = 0;
virtual bool ReceiveToolES(MediaStream *tool_ES, ES_Descriptor *tool_ESD) = 0;
virtual bool ReceiveIPMP_ToolListDescriptor( IPMP_ToolListDescriptor* toolList ) = 0;
};
```

After we have described about the class and interface of the Tool Manager in scheme 4C, we will discuss the implementation of Tool Manager. There are two key points for the Tool Manager implementation in the Software Reference program.

(1) The class TM_Interface is an abstract class. It is instantiated by IPMPServicesFull object. The Tool Manager interface function is implemented by IPMPServicesFull object.

(2) The following functions are defined in

.\Craig-IPMP\IPMPXFull\IPMPXFull.cpp

IPMPServicesFull::Parse_IPMPTool(..)

IPMPServicesFull::RetrieveMissingTool(..)

IPMPServicesFull::ReceiveIPMP_ToolListDescriptor()

IPMPServicesFull::ConnectTool(..)

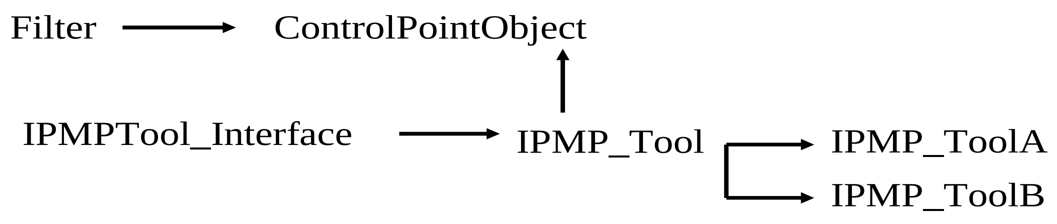
These member functions are the major functions for the Tool Manager to implement as IPMPX specification describes.

4.3.4 The Description of Control Point Implementation

Specifies the IPMP Control Point at which the IPMP Tool resides such that the IPMP Tool knows where it must perform its module. The Software Reference creates thread to implement the Control Point function that receives input stream and transfers the stream to the specified tool. After the tool module is implemented, the Control Point transfers the stream to next level. The next level may be the decoder or the composite or others.

We highlight the class relationship of the Control Point and interface of Control Point as below.

(1) Class Relationship for Control Point



(2) Control Point Interface

Scheme 4D:

```
class ControlPointObject : public Filter, public IPMPX_Tool
{
public:…
// Filter interface methods.
void Start ();
void Stop ();
virtual void SetInputStream (MediaStream *pStream){…}
void SetOutputStream (MediaStream *pStream){..}
MediaStream *GetInputStream () {..}
MediaStream *GetOutputStream () {..}
int GetOptimalOutputSize ( int inputSize ) { … }
// IPMPX_Tool interface methods
bool ReceiveMessage( ToolMessage* ) {return false;}
bool ProcessData (..);
bool AddTool( ..);
bool RemoveTool( ..);
void SetControlPoint( ){…}
SDLInt<8> GetControlPointCode(){…}
private: // Methods
….
};
```

After we have described about the class and interface of the Control Point in scheme 4D, we will discuss the implementation of Control Point. There are three key points for the Control Point implementation in the Software Reference program.

(1) The Software Reference executes Control Point function in the thread way.

(2) ProcessData() may be instantiated by ControlPointObject or IPMP_ToolA,and which one is decided by the control flow of the currently execution.

(3) void ControlPointObject::Run()

It is responsible for the major work to reach control point function. The function retrieves units from the input stream, passes them to the first tool in the tool sequence, and dispatched the processed units to the output stream, till the input stream’s end is exceeded

4.3.5 The Description of Tool Module Implementation

When this document was written, there was still no tool available in the reference software to allow the embedding of IPMP elementary streams inside MPEG4 files. So, currently, the Software Reference version is only considered the tool module embedded in the terminal. We highlight the class relationship of the Tool Module and interface of Tool Module in scheme 4E as below.

(1) Class Relationship for the Tool Module

$$\begin{tikzcd}
\text{IPMPTool\_Interface} \arrow[r] & \text{IPMPX\_Tool} \arrow[r] \arrow[dr] \arrow[ddr] & \text{ToolA} \\
 & & \text{ToolB} \\
 & & \text{.....}
\end{tikzcd}$$

(2) the Tool Module Interface

Scheme 4F:

```
class IPMP_DummyTool : public IPMPX_Tool
{
  …// constructor and deconstructor
// IPMPTool_Interface interface methods.
bool ReceiveMessage( ToolMessage* );

// Interface methods added by IPMPX_Tool
bool ProcessData (…);
bool ProcessMessage ( …);
private:
// Member data needed during tool functioning.
IPMPTool_Interface   m_pMRInterface;
__int32   myContextID;
IPMP_Descriptor   m_pMyIPMPDescriptor;
};
```

The Software Reference provides the tool framework for developers to develop application programs. This framework is defined in \Craig-IPMP\IPMP_DummyTool\IPMP_DummyTool.cpp. The tool framework is as below. We only list the key parts and comments.

Scheme 4G:

```
bool   IPMP_ToolA::ReceiveMessage( ToolMessage      msg )
{      ……
// This function calls ProcessMessage() for processing IPMP message.
// The developer can modify this function such that it meets application requirements.
}

bool   IPMP_ToolA::ProcessData ( LPBYTE        pInput, int nInputLength
                                               , DWORD      dwTime )
{      …..
//This function receives media stream and passes the stream to Control Point object.
//The developer can modify this function such that it meets application requirements.
}

bool IPMP_ToolA::ProcessMessage ( ToolMessage*                    base
                                                      , IPMP_Data_BaseClass      *       msg   )
{
//The developer can add code to this function such that it meets application requirements.

        switch (msg->GetTag()){
        case(TAG_IPMP_Secure_Container):
                {
                        return false;
                }
        …….
        case(TAG_User_Initialize):
                {
                // The Software Reference uses this case such that the Tool Message can be transferred
                to terminal .The below codes describes the message transferred to terminal.      …
                Msg.receiver = 0x00;
                Msg.sender = myContextID;
                IPMP_CanProcess      payload;
                payload.canProcess = true;
                Msg.IPMP_MessageFromBitstream += &payload;
                Msg.IPMP_MessageFromBitstream += &anotherPayload;
                m_pMRInterface->ReceiveMessage(&newMsg);
                return true;
                }
        }

        return false;
}
```