

國立交通大學

電子工程學系 電子研究所碩士班

碩士論文

MPEG-4 IPMPX 系統

在 MPEG-21 測試平台上的設計與實現



MPEG-4 IPMPX Design and Implementation

On MPEG-21 Testbed

研究生：范振韋

指導教授：杭學鳴 博士

中華民國九十三年六月

MPEG-4 IPMPX 系統在 MPEG-21 測試平台上的設計與實現

**MPEG-4 IPMPX Design and Implementation on MPEG-21
Testbed**

研究生：范振韋

Student: Chen-Wei Fan

指導教授：杭學鳴

Advisor: Hsueh-Ming Hang

國立交通大學
電子工程學系 電子研究所碩士班
碩士論文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Electronics Engineering

June 2004

HsinChu, Taiwan, Republic of China

中華民國九十三年六月

MPEG-4 IPMPX 系統在 MPEG-21 測試平台上的設計與實現

研究生：范振韋

指導教授：杭學鳴 博士

國立交通大學

電子工程學系 電子研究所碩士班

摘要

近幾年來，數位化的多媒體內容開始成為趨勢，保護數位化多媒體內容的智慧財產權這一個議題變得越來越受到重視，本論文將介紹並且實現一個由 MPEG 會議發展的數位多媒體內容智慧財產權保護系統 MPEG-4 IPMPX 系統，本論文的目的是研習 IPMP 的標準，接著設計並實現 MPEG-4 IPMPX 系統在 MPEG-21 多媒體傳輸的測試平台上，MPEG-21 測試平台是 MPEG 標準的一部份，其目的是用來測試多媒體的傳輸。我們在把 MPEG-4 IPMPX 系統整合進 MPEG-21 的測試平台上，並在其上測試 MPEG-4 IPMPX 系統的功能，如數位影像的加密，解密，以及浮水印的加入與取出比對等功能。我們設計了 IPMPX 系統的軟體架構，以及其 API，並且，我們完成了一個 MPEG-4 IPMPX 系統與其應用的範例。為了符合 MPEG-21 測試平台，我們使用 C++ 語言實現 IPMP 系統。論文中，我們完成並描述 IPMP 系統與其應用範例。

MPEG-4 IPMPX Design and Implementation on MPEG-21 Testbed

Student: Chen-Wei Fan

Advisor: Dr. Hsueh-Ming Hang

Department of Electronic Engineering &
Institute of Electronics
National Chiao Tung University

Abstract

Distribution of digital media content is a trend in recent years. The issues of protecting the rights of digital content are thus becoming more and more important. In the thesis, the IPMPX system (Intellectual Property Management and Protection Extension system), a Digital Rights Management interface and architecture developed by MPEG is introduced and implemented. The goal of this thesis is to study the IPMP standard and design an MPEG-4 IPMPX system implementation on the MPEG-21 Multimedia Test Bed. MPEG-21 Test Bed is a part of MPEG standard for testing multimedia resource delivery. We integrate the MPEG-4 IPMP system into the MPEG-21 Testbed to perform the functions of IPMP such as en/decryption and watermarking. We design the software architecture of the IPMP system including its APIs. Furthermore, we design and implement an application example which uses the IPMPX system with MPEG-21 Test Bed to show its powerful functionalities. To match the other parts of the MPEG-21 Test Bed, the IPMP subsystem realization is written in C++. The IPMP subsystem together with the application example has been completed and described in this thesis.

誌謝

首先要感謝我的指導教授 杭學鳴 博士這兩年多來的指導，老師的研究態度、謙虛以及實事求是的精神一直是我的典範。在研究的過程中，遇到的問題與困難，老師總是適時的給予幫助與指點方向，讓我克服研究上的瓶頸，並順利的完成。除了課業上的幫助以外，我還要特別感謝老師在我最困難的時候，給予我鼓勵以及協助，老師的關懷以及體諒，讓我感到溫暖受用，讓我度過了心情的低潮，重新的站起來，面對往後的挑戰，這一切都要感謝老師，謝謝！

在這裡也要感謝通訊電子與訊號處理實驗室，提供了極佳研究環境，讓我在研究中有充足的資源可以使用。也感謝實驗室全體成員，營造了一個充滿活力與和諧的環境氣氛，實驗室的溫馨與有條理的環境，一直是我身為實驗室成員自豪的一點。感謝張峰城學長，在我從事論文研究時不吝提供經驗與鼓勵，以及教導我許多軟體開發的智識。也感謝蔡家陽學長在研究上的協助，另外，也要莊孝強、方耀諄、蘇子良等同學，適時提供技術上的支援，以及陪伴我走過兩年的研究生歲月，感謝這一切，使得論文能夠順利的進行。

最後，要感謝的是我的家人，他們讓我能夠心無旁騖的從事研究工作。沒有家人在背後的支持與體諒，也就沒有今天的我，在此，謹獻上最高的謝意與歉意。

謝謝所有陪我走過這一段歲月的師長、同儕與家人，所有愛我的人，我愛的人，謝謝！

Contents

摘要	v
Abstract.....	vi
Chapter 1 Introduction.....	1
Chapter 2 IPMP Extension System.....	4
2.1 IPMP Hook	4
2.2 IPMP Extension Overview	5
2.2.1 IPMP Tool Manager.....	6
2.2.2 IPMP Message Router	7
2.2.3 Terminal.....	7
2.2.4 IPMP Tool.....	7
2.2.5 IPMP Control Point (IPMP Filter).....	8
2.3 MPEG-4 IPMP Extension Specification	8
2.3.1 IPMP Control Information.....	8
2.3.2 IPMP Messages	15
2.3.3 Using IPMP	17
Chapter 3 Overview of MPEG-21 Resource Delivery Test Bed.....	20
3.1 Server components	21
3.2 Client components	22
3.3 Common components	23
3.4 Network	25
Chapter 4 IPMP Reference Software.....	27
4.1 IM1 IPMPX Reference Software	27
4.1.1 IPMPToolInterface	28
4.1.2 MRInterface.....	28
4.1.3 TMInterface	30
4.1.4 IPMPSystem	31
4.2 PSL MPEG-2 IPMP-X Reference Software.....	33
Chapter 5 MPEG-4 IPMP Extension System Implementation on MPEG-21 Test Bed	37
5.1 Architecture Design	38
5.1.1 Context	38
5.1.2 IPMP Control Points (IPMP Filters).....	41

5.2 Design and Implementation of Software	44
5.2.1 Message Router	44
5.2.2 Tool Manager	48
5.2.3 Terminal	50
5.2.4 IPMP Filter	51
5.2.5 IPMP Tool	55
Chapter 6 Demo	59
6.1 Structure	59
6.2 Setup	60
6.3 Execution	64
Chapter 7 Conclusion	69
References	72
自 傳	73



List of Figures

Figure 1 Multimedia Delivery System with IPMP	2
Figure 2 MPEG-4 IPMPX system [1]	6
Figure 3 Sample of IPMP Tool context ID mapping [1]	15
Figure 4 MPEG-4 IPMP basic concept [1].....	18
Figure 5 Architecture of MPEG-21 Testbed [5]	20
Figure 6 Network protocol [6].....	25
Figure 7 IPMPX system in IM1 [3].....	27
Figure 8 IPMPX class hierarchy in IM1.....	28
Figure 9 MPEG-2 IPMPX system software architecture [4].....	33
Figure 10 Class hierarchy of message interface of PSL IPMPX reference software	36
Figure 11 Relationships among modules in the IPMPX system	37
Figure 12 Context hierarchy	38
Figure 13 Links between four level contexts	39
Figure 14 Search the context tree	40
Figure 15 Relationship between IPMP Context and other modules.....	40
Figure 16 Block diagram of IPMPX system at client side	42
Figure 17 Block diagram of IPMPX system at server side	43
Figure 18 Modification of Decoder	44
Figure 19 Relationship between Message Router and other modules.....	45
Figure 20 The messaging procedure.....	46
Figure 21 Relationship between Tool Manager and other modules	48
Figure 22 Relationship between Terminal and other modules	50
Figure 23 Relationship between IPMP Filter and other modules.....	52
Figure 24 The procedure of processing data by IPMP Filter at client side	54
Figure 25 The procedure of processing data by IPMP Filter at server side.....	54
Figure 26 Relationship between IPMP Tool and other modules	55
Figure 27 Flow chart of IPMP DES Tool	57
Figure 28 System diagram of our application	59
Figure 29 Relationship of various pointers in the demonstration system.	63
Figure 30 Messaging routine in the demonstration system.....	64
Figure 31 Screenshot_1 of demonstration system.....	65

Figure 32 Screenshot_2 of demonstration system..... 66
Figure 33 Screenshot_3 of demonstration system..... 67
Figure 34 Screenshot_4 of demonstration system..... 67



Chapter 1

Introduction

Digitization of media content became a trend about twenty years ago. However, following the digitization of media content, protecting the rights of media content becomes an important issue. Digital media contents have the advantages of perfect copy and separation from the media container, but these advantages become disadvantages in the view of rights protection and management. A new component called digital right management becomes an important element in the digital media delivery system.

Designing and implementing a Digital Rights Management system is very challenging. A DRM system must provide an open, safe, and trusted environment for media content delivery and must provide interoperable services. In digital media content market, the industry and users are now demanding that standards be developed to allow the interoperability of the DRM system.

Since DRM becomes more and more important and are demanded to be standardized recently, there are several groups begin to standardize the DRM system such as OpenEBook Forum, Moving Picture Experts Group committee, the Internet Engineering Task Force (IETF), and the World Wide Web Consortium (W3C). In this thesis, we will focus on the standardized DRM system developed by the MPEG group, the MPEG-4 IPMP Extension [1] system.

IPMPX (Intellectual Property Management and Protection Extension) system is a Digital Rights Management interface and architecture specifications developed by the MPEG group. There are now several versions of IPMPX reference software. One is the MPEG-4 IPMPX system included in IM1 and is developed by Craig A. Schultz. Another one is called MOSES [8] IPMPX implementation. The third IPMPX system is developed by Panasonic Singapore Laboratories Pte Ltd (PSL) [4].

The IPMPX system defined by the MPEG group focuses mostly on the terminal (client) side. It does not clearly specify the server side. In a media streaming system, there is not only client side but also server side, and both sides need the IPMPX system so as to deliver the media content in an open, secure, and trusted environment. In order to construct an

implementation of IPMPX system and test its functionalities in a streaming environment, we need a multimedia delivery test bed. We choose the MPEG-21 [7] Testbed [5] [6], a multimedia test bed for resource delivery developed by MPEG group, as our platform to design and implement the IPMPX system. Furthermore, we need test the functionalities of IPMPX system of MPEG-21 Testbed.

The goal of this thesis is to study the standard of MPEG-4 and design an implementation of MPEG-4 IPMPX system on the MPEG-21 Testbed. The system diagram is shown in Figure 1.

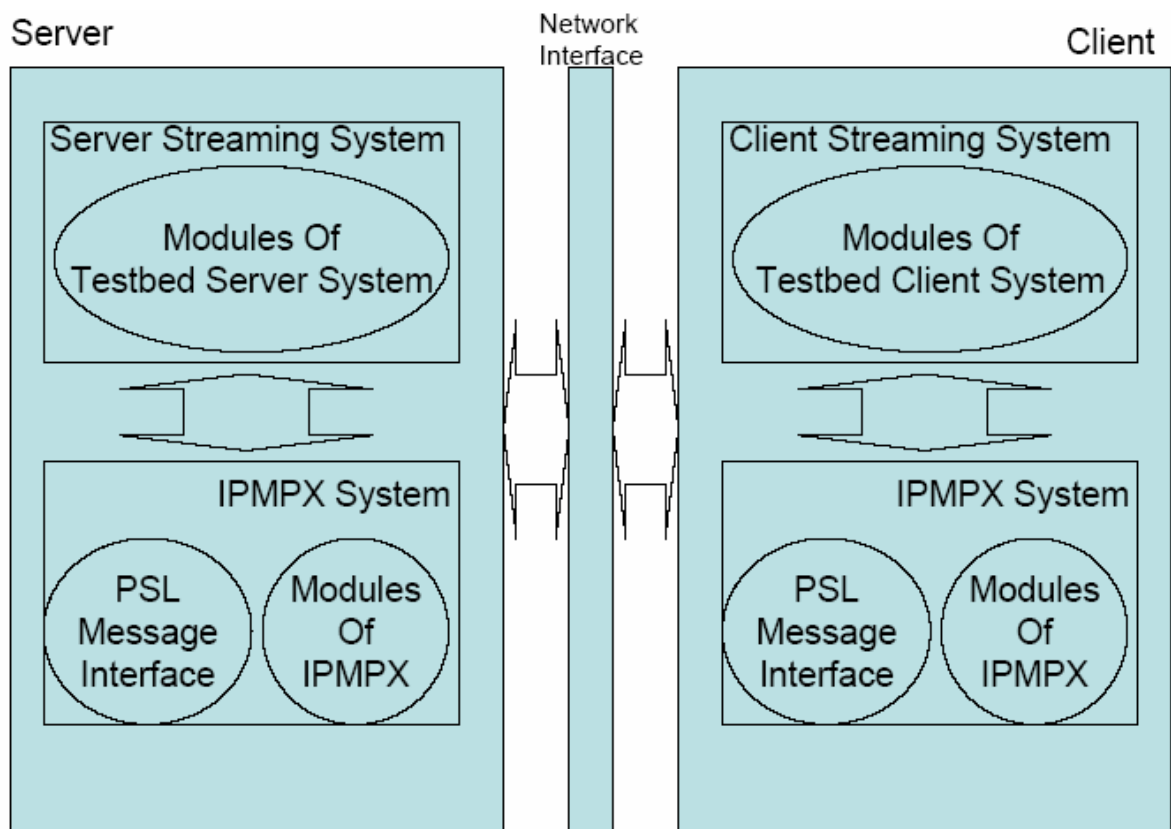


Figure 1 Multimedia Delivery System with IPMP

We design an MPEG-4 IPMPX system and integrate it to the MPEG-21 Testbed to perform the IPMP functions such as en/decryption and watermarking. The designed MPEG-4 IPMP system integrates the Message Interface developed by PSL within. There are some benefits of using the IPMPX system such as security, interoperability, renewability, flexibility, and dynamic operation. Therefore, we design and implement a MPEG-4 IPMPX application system on MPEG-21 resource delivery test bed to demonstrate its functionalities and the powerfulness. This thesis is mainly divided into five parts. First, we describe the concept and

structure of MPEG-4 IPMPX. Second, the MPEG-21 resource delivery test bed is introduced. Then, we give details of the reference software of the IPMPX system. After that, we describe the details about our design and implementation of the IPMPX system on the MPEG-21 Testbed. Finally, several application examples are designed and implemented, and we give a demonstration at the end.



Chapter 2

IPMP Extension System

In following two chapters, chapter 2 and chapter 3, we describe some related works on designing an implementation of MPEG-4 IPMP system [1]. This implementation of MPEG-4 IPMP system is built in the MPEG-21 Testbed [6], which is designed for media coding and testing in streaming environment. So, we first introduce the development of IPMP in MPEG committee, and we give a review of the MPEG-4 IPMP Extension standard to provide some basic concept about the IPMP Extension system. Finally, we describe that what the MPEG-21 Testbed is and how it works.

The ISO/IEC MPEG, Moving Picture Experts Group, committee started to specify a common interface/protocol for IPMP a few years ago. They try to merge IPMP system into exist MPEG multimedia standards, such as MPEG-2 and MPEG-4. They first focus on the standard, MPEG-4, and MPEG-4 IPMP is now almost finished. The committee hopes that the specified MPEG-4 IPMP system could be integrated into other MPEG standards like MPEG-2 or even MPEG-21 with slight modification.

2.1 IPMP Hook

In MPEG-4, IPMP Hook was the first IPMP system. However, after the development of IPMP Extension system (IPMPX), the original IPMP system in MPEG-4 was replaced and named as IPMP Hook for distinguishing from IPMPX. Why the IPMP should be extended from Hook to IPMPX? In IPMP Hook, there allows multiple IPMP systems to exist in the same one terminal. However, the IPMP Hook does not specify that how does an IPMP system be hooked on the terminal. Furthermore, there is no standardized mechanism for the authentication between the different IPMP systems hooked on the same terminal. Since there allows multiple IPMP systems existing in the same terminal, easy replacement mechanism is important. However, in IPMP Hook, to replace an IPMP system is not very easy. So, the MPEG committee moved from IPMP Hook to IPMP Extension to solve all these problems described above. The IPMP Hook nowadays is out of date, so, we ignore the details about the

IPMP Hook.

2.2 IPMP Extension Overview

There are two versions of IPMPX specifications, MPEG-2 IPMPX [2] and MPEG-4 IPMPX [1]. In this section, we give an overview of the IPMP Extension in MPEG-4 standard. The MPEG-4 IPMP Extension system, or IPMPX system, is now at the stage of Final Draft Amendment. In the 62nd MPEG meeting in October 2002, the MPEG-4 IPMP system is declared to become a new part, part 13, of MPEG-4 standard.

IPMP Extension system has several benefits to the industry and end users. Many Digital Rights Management systems use the same or similar components to perform the functionalities of DRM. However, there are too many redundant designs that waste times and money. Using IPMP extension could reduce the redundant work. For instance, company A designs an AES decryption tool, and different DRM systems adopting the IPMP Extension could use this tool directly without redesigning the same tool.

IPMP Extension also provides some mechanisms for the secure communication between components, for instance, Tool to Tool or Terminal to Tool. Mutual Authentication is introduced to perform the secure communication with the authenticated channel. Furthermore, IPMP Extension provides the interoperability by clearly identifying what is provided by the architecture and what must be provided by a DRM system. And, the renewability is also an advantage using IPMP Extension. Because there is an elementary stream called IPMP ES in IPMP Extension in MPEG-4, the whole IPMP Extension system can change the parameters dynamically. The user can choose whatever tools, encryption, decryption, audio watermarking, or video watermarking, to plug them into the IPMP Extension system. This results in IPMP Extension being more flexible.

MPEG-4 IPMP system is a message based system. The standard specifies the interface and protocol for these messages. Such that, tools, such as encryption, decryption, or watermarking, developed by many different companies could be plug into the system through the common interface/protocol. This makes the development of IPMP tools be free from the whole IPMP system.

The basic concept of MPEG-4 IPMP system is a Virtual Terminal, which contains mainly two parts, Tool Manager (TM) [1] and Message Router (MR) [1]. The Message Router, as the

name, is to pass the message to the corresponding receiver. All IPMP Tool Messages are routed through this terminal. However, Tool Manager is to manage and connect all the IPMP Tools to perform the IPMPX function. The architecture of MPEG-4 IPMP Extension is illustrated in Figure 2. All the details will be described as follows.

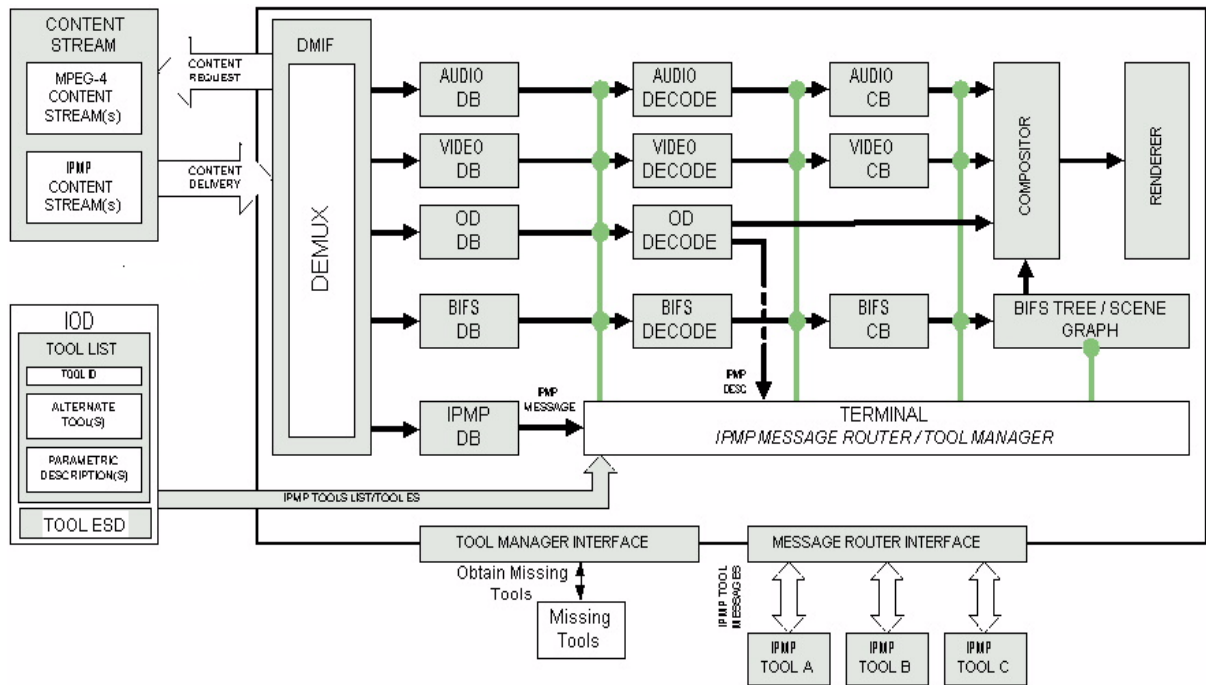


Figure 2 MPEG-4 IPMPX system [1]

2.2.1 IPMP Tool Manager

The Tool Manager [1] here is a conceptual entity to perform some functionalities of managing all the IPMP Tools within the Terminal. Where the conceptual entity means that there may be no real instance of IPMP Tool Manager, but only be some functions or methods to achieve the function. An IPMP Tool Manager within an IPMP system should parse the IPMP Tool List, which is a list of IPMP Tools that are required for content consumption. Receiving the IPMP Tool List that dispatched from the Terminal, Tool Manager resolves the alternative list, and parametric description within the IPMP Tool List. If there are any IPMP Tools that are not available at the local side, the IPMP Tool Manager has to retrieve these IPMP Tools remotely, from the media content delivery server or from the website. Because it is allowed in the IPMP Extension system that IPMP Tool could be carried within a elementary stream called IPMP Tool ES, the IPMP Tool Manager has to retrieve the IPMP Tools from the IPMP Tool ES. Since the IPMP Tool Manager has to manage all the IPMP Tools in the

Terminal, it should also instantiate the IPMP Tool instances required for content consumption, and then maintain the mapping table of each IPMP Tool and context.

2.2.2 IPMP Message Router

The IPMP Message Router [1], as the IPMP Tool Manager is a conceptual entity to perform the message routing. There are many IPMP messages that can be divided into two groups, IPMP Tool Message, and IPMP Device Message. We will give the details about these two kinds of IPMP messages in section 2.3.2. Except to routing the IPMP messages, the IPMP Message Router should also parse the IPMP Tool Descriptor dispatched from the Terminal. The IPMP Tool Descriptor contains some information for IPMP Tool initialization. Also, the IPMP Message Router handles the IPMP elementary stream, which contains the dynamic IPMP information for updating the IPMP system.

2.2.3 Terminal

The Terminal [1] is an environment where the IPMP system performs its functionalities. For example, it may be an IM1 terminal or any compatible terminals that could perform content consumption combining with the IPMP system. The Terminal should receive the IPMP Tool Descriptor, IPMP elementary stream, and the IPMP Tool Descriptor Pointer from the bitstream and dispatch them to the IPMP Message Router. And, it receives the IPMP Tool List from bitstream and dispatches this to the IPMP Tool Manager. The description above is the work of Terminal related with IPMP. There still some work done by the Terminal such as decoding the bitstream data, displaying, etc.

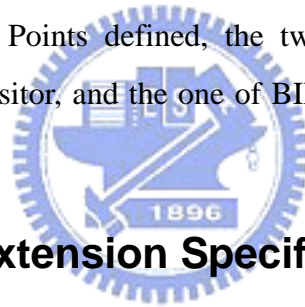
2.2.4 IPMP Tool

The IPMP Tool [1] is the component that really performs the functionalities of IPMP. Such functionalities include encryption, decryption, watermarking insertion, watermarking extracting, authentication, etc. The IPMP Tools may be instantiated in the IPMP Control Points to perform the data processing or may be instantiated outside all the IPMP Control Points to perform other functionalities that are not related with the data, for instance, mutual authentication. The IPMP Tools within the IPMP Extension system should be able to receive

the IPMP messages that routed by IPMP Message Router. These messages may come from other IPMP Tools or from the bitstream. When an IPMP Tool receives the IPMP message, the process of the message is implementation dependant and is not defined in the specification.

2.2.5 IPMP Control Point (IPMP Filter)

IPMP Control Points [1] are the place where the IPMP Tools perform its function. IPMP Control Points are just like the Filters with one or more IPMP Tools plugged inside. For example, there is one IPMP Control Point between the decoder and decoding buffer, and one between the decoder and the composition buffer. Then, the IPMP Tool that is to decrypt the stream data will be inserted into the IPMP Control Point between the decoder and decoding buffer. And, the IPMP Tool that is to extract the watermarking will be plugged into the IPMP Control Point between the decoder and composition buffer. There allows multiple IPMP Tools to be plugged into one IPMP Control Point. In the MPEG-4 IPMP Extension specification, there are four IPMP Control Points defined, the two described above, the one between composition buffer and compositor, and the one of BIFS tree. There reserves some positions for user defining.



2.3 MPEG-4 IPMP Extension Specification

This chapter gives the details about the MPEG-4 IPMP Extension specification. We first give the details about the IPMP control information that defined in the specification. Because there are numerous IPMP message defined in the MPEG-4 IPMP Extension specification, we only take some for example and describe them in detail. Finally, we give a possible scenario of the process of IPMP.

2.3.1 IPMP Control Information

The IPMP control information may be contained in the Initial Object Descriptor (IOD) [1], Object Descriptor update command [1], or IPMP elementary stream [1], and is to specify how the IPMP system works. The IPMP control information is composed of IPMP Tool List Descriptor [1], IPMP Tool Descriptor [1], and IPMP Tool Descriptor Pointer [1]. We give the detail about them as later. Before describing these, we first give the detail about the IOD.

Initial Object Descriptor

This is an extension of the `InitialObjectDescriptor` being extended by adding `IPMP_ToolListDescriptor` and `IPMP_ToolDescriptor` and `IPMP Tool Descriptor Pointer`. The syntax of an `InitialObjectDescriptor` is illustrated as follows [1].

```
class InitialObjectDescriptor extends ObjectDescriptorBase : bit(8) tag=  
InitialObjectDescrTag  
{  
    bit(10) ObjectDescriptorID;  
    bit(1) URL_Flag;  
    bit(1) includeInlineProfileLevelFlag;  
    const bit(4) reserved=0b0000;  
    if (URL_Flag) {  
        bit(8) URLLength;  
        bit(8) URLstring[URLLength];  
    } else {  
        bit(8) ODPProfileLevelIndication;  
        bit(8) sceneProfileLevelIndication;  
        bit(8) audioProfileLevelIndication;  
        bit(8) visualProfileLevelIndication;  
        bit(8) graphicsProfileLevelIndication;  
        ES_Descriptor esDescr[1 .. 255];  
        OCI_Descriptor ociDescr[0 .. 255];  
        IPMP_DescriptorPointer ipmpDescrPtr[0 .. 255];  
        IPMP_ToolListDescriptor toolListDescr[0 .. 1];  
        IPMP_ToolDescriptor ipmpToolDescriptor[0 .. 255];  
    }  
    ExtensionDescriptor extDescr[0 .. 255];  
}
```

At the beginning, the Terminal must access IOD for the initialization of IPMP system. Here, we only concern about the stuffs associated with IPMP system, IPMP Tool List Descriptor, IPMP Tool Descriptor, and IPMP Tool Descriptor Pointer. Note that there may be multiple IPMP Tool Descriptor and multiple IPMP Tool Descriptor pointers, however, there is only one IPMP Tool List Descriptor within the IOD.

IPMP Tool List Descriptor

The IPMP Tool List Descriptor is to transport the list of the required IPMP Tools for content consumption, and the data structure is shown as follows [1].

```
class IPMP_ToolListDescriptor extends BaseDescriptor :
```

```

    bit(8) tag=IPMPToolListDescrTag
  {
    bit(8) numTools;
    IPMP_Tool ipmpTool[numTools];
  }


```

The IPMP_Tool above is the data structure to describe the IPMP Tool. In the specification, the IPMP Tool List Descriptor allows three modes for describing the required IPMP Tools for content consumption, Unique, Alternates, and Parametric. The structure of IPMP_Tool is shown as follows [1].

```

class IPMP_Tool extends BaseDescriptor :
  bit(8) tag= IPMP_ToolTag
  {
    bit(128)  IPMP_ToolID;
    bit(1)    isAltGroup;
    bit(1)    isParametric;
    const bit(6)    reserved=0b0000.00;
    if(isAltGroup){
      bit(8)    numAlternates;
      bit(128)  specificToolID[numAlternates];
    }
    if(isParametric)
      ByteArray toolParamDesc;
    int(8) numURLs;
    ByteArray ToolURL[numURLs];
  }

```




The IPMP_ToolID is a 128-bit identifier to distinct every IPMP Tool that is required for content consumption in Terminal. Using the unique mode, the IPMP Tool is described by a unique ID, IPMP_ToolID and set both isAlterantes and isParametric as zero. While using the Alternate mode, Terminal is allowed to choose one of the alternate IPMP Tools specified in specificToolID[numAlternates]. When the parametric mode is used, the ByteArray, toolParamDesc, conveys the parametric description of the IPMP Tool with the Tool ID, IPMP_ToolID. The parametric description is used when a used IPMP Tool has the following features.

1. It is based on a popular algorithm.
2. It has many equivalent implementations of the same variable.
3. It will be computationally intensive, leading to platform-specific optimized implementations, from a wide variety of vendors.

The parametric description, or configuration, is used for instantiation of an IPMP Tool instance or for the mode change of an IPMP Tool. In the specification, the schema of the parametric description is not specified, but only a basic infrastructure in the current version. Basically, the parametric description may contain the following information, version of the parametric description, the class of the IPMP Tool, the sub-class of the IPMP Tool, and the sub-class-specific information. The class of IPMP Tool specifies what kind of the IPMP Tool is, decryption, rights description language parser, or others. If the IPMP Tool is specified as a decryption tool, the detail information, such as AES, DES, etc, is described by the sub-class of IPMP Tool. In order to specify some more detail information such as number of bits, block size, etc, we use the sub-class-specific information.

IPMP Tool Descriptor

The IPMP Tool List Descriptor specifies what IPMP Tools are required for the content consumption, and the IPMP Tool Descriptor provides IPMP Tools with the information for initialization. The data structure of the IPMP Tool Descriptor is shown as follows [1].



```

class IPMP_ToolDescriptor() extends BaseDescriptor : bit(8) tag =
IPMP_ToolDescrTag
{
    bit(16) IPMP_ToolDescriptorID;
    bit(128) IPMP_ToolID;
    if (IPMP_ToolID == 0)
        ByteArray URLString;
    else
    {
        bit(1) isInitialize;
        const bit(7) reserved = 0b0000.000
        if(isInitialize)
            IPMP_Initialize init;
        bit(8) numOfData;
        IPMP_Data_BaseClass[numOfData] IPMPX_data;
    }
}

```

The IPMP Tool Descriptor carries the IPMP information for one or more IPMP Tools. It may be conveyed in the IOD, OD update command, or IPMP elementary stream. Within it, the IPMP_ToolDescriptorID is the unique identifier of the IPMP Tool Descriptor, and the generation of the IPMP_ToolDescriptorID is the implementation issue. The IPMP Tool that is

described by the IPMP Tool Descriptor is specified by the IPMP_ToolID. However, if the IPMP_ToolID is zero, the remote IPMP Tool Descriptor is pointed by the URLString. If the IPMP_ToolID is a nonzero value, the following data, IPMP_Initialize, and IPMPX_data are the IPMP information carried by the IPMP Tool Descriptor for IPMP Tools. The data structure if IPMP_Initialize is shown as follows [1].

```
class IPMP_Initialize()
{
    bit(8) controlPointCode;
    bit(8) sequenceCode;
    bit(8) numOfData;
    IPMP_Data_BaseClass[numOfData] IPMPX_data;
}
```

The IPMP Tool Descriptor carries IPMP_Initialize when the isInitialize is set to one. It makes one or more IPMP Tools to be instantiated at the position that is specified by the controlPointCode and sequenceCode. Here the controlPointCode specified in the specification is shown in the following table [1].

controlPointCode	Description
0x00	No control point.
0x01	Control Point between the decode buffer and the decoder.
0x02	Control Point between the decoder and the composition buffer.
0x03	Control Point between the composition buffer and the compositor.
0x04	BIFS Tree
0x05-0xDF	ISO Reserved
0xE0-0xFE	User defined
0xFF	Forbidden

When there are multiple IPMP Tools within the same one IPMP Control Point, the order of processing data is decided by the sequenceCode. The IPMP Tool with the higher sequenceCode processes the data first.

And the instantiated IPMP Tools is provided with the IPMP information contained in IPMPX_data. The data type, IPMP_Data_BaseClass, is an expandable base class, and there are many kinds of IPMP data expanded from it. The definition of IPMP_Data_BaseClass is shown as follows [1].

```
abstract aligned(8) expandable(2^28-1) class IPMP_Data_BaseClass: bit(8) tag=0 ..
255
```

```

{
  bit(8) Version;
}

```

The data types extended from the IPMP_Data_BaseClass is specified by the tag value shown as follows [1].

8-bit Tag Value	Symbolic Name
0x00	Forbidden
0x01	IPMP_OpaqueData_tag
0x02	IPMP_AudioWatermarkingInit_tag
0x03	IPMP_VideoWatermarkingInit_tag
0x04	IPMP_SelectiveDecryptionInit_tag
0x05	IPMP_KeyData_tag
0x06	IPMP_SendAudioWatermark_tag
0x07	IPMP_SendVideoWatermark_tag
0x08	IPMP_RightsData_tag
0x09	IPMP_Secure_Container_tag
0x0A	IPMP_AddToolNotificationListener_tag
0x0B	IPMP_RemoveToolNotificationListener_tag
0x0C	IPMP_InitAuthentication_tag
0x0D	IPMP_MutualAuthentication_tag
0x0E	IPMP_UserQuery_tag
0x0F	IPMP_UserQueryResponse_tag
0x10	IPMP_ToolParamCapabilitiesQuery_tag
0x11	IPMP_ToolParamCapabilitiesResponse_tag
0x12	IPMP_GetTools_tag
0x13	IPMP_GetToolsResponse_tag
0x14	IPMP_GetToolContext_tag
0x15	IPMP_GetToolContextResponse_tag
0x16	IPMP_ConnectTool_tag
0x17	IPMP_DisconnectTool_tag
0x18	IPMP_NotifyToolEvent_tag
0x19	IPMP_CanProcess_tag
0x1A – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

In the specification, all the data structure of the types listed above is illustrated. Take the IPMP_Opaque_data for example, the data structure of it is shown below [1].

```

class IPMP_OpaqueData extends IPMP_Data_BaseClass
: bit(8) tag = IPMP_OpaqueData_tag
{
  ByteArray opaqueData;
}

```

The opaqueData is the opaque IPMP information that is conveyed to IPMP Tools. This is

the simplest example that is extended from the IPMP_DataBaseClass. The definitions of all the others can be found in the specification.

IPMP Tool Descriptor Pointer

The IPMP Tool Descriptor Pointer exists in the ipmpToolDescrPtr section of an OD or ESD structure. It is a pointer that points to an IPMP Tool Descriptor by specifying an IPMP Tool Descriptor ID within it. The IPMP Tool Descriptor Pointer existing in an object descriptor indicates that all streams referred to by embedded elementary descriptor are subject to protection and management by the IPMP Tool specified in the pointed IPMP Tool Descriptor. And, the IPMP Tool Descriptor Pointer that is contained in an elementary stream descriptor indicates that the stream associated with the ES descriptor is subject to protection and management by the IPMP Tool specified in the referenced IPMP Tool Descriptor. Every IPMP Tool Descriptor pointer results in a unique instance of the corresponding IPMP Tool. The data structure shown below is the IPMP Tool Descriptor Pointer [1].

```
class IPMP_DescriptorPointer extends BaseDescriptor :  
bit(8) tag = IPMP_DescrPtrTag  
{  
    bit(8) IPMP_DescriptorID;  
    if (IPMP_DescriptorID == 0xff){  
        bit(16) IPMP_ToolDescriptorID;  
        bit(16) IPMP_ES_ID;  
    }  
}
```

The IPMP_Descriptor above is to back compatible with the IPMP Hook, so we ignore this. The IPMP_ES_ID specified in the IPMP Tool Descriptor Pointer is to indicate the identifier of the IPMP elementary stream that carries the IPMP information for the referenced IPMP Tool indicated by the IPMP Tool Descriptor with its identifier as IPMP_ToolDescriptorID. Figure 3 gives a simple example about the mapping relationship of OD, ESD, IPMP Tool Descriptor, and IPMP Tool Descriptor Pointer.

In this example, the OD_UPDATE part has two ODs with one ESD and one IPMP Tool Descriptor Pointer contained within each. And, the IPMP UPDATE part has two IPMP Tool Descriptor that are going to pointed by the IPMP Tool Descriptor Pointer within the OD_UPDATE part. The right half part of Figure 3 shows the resolved diagram of the left half

part. The IPMP Tool Descriptor Pointer within the OD A containing an ESD C indicates that there is one IPMP Tool at the AUDIO ES. It is the same for the VIDEO ES. There are many cases of use and scope of the IPMP Descriptors and declaration, and they are illustrated at the Annex F in the MPEG-4 IPMP Extension specification [1].

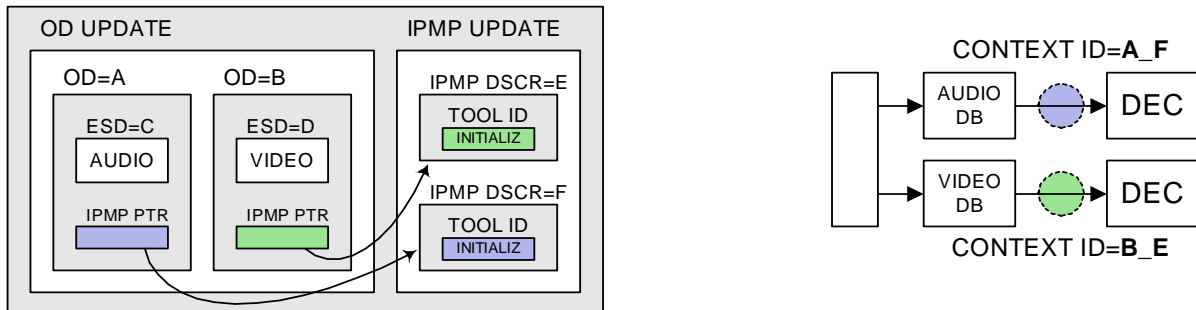


Figure 3 Sample of IPMP Tool context ID mapping [1]

2.3.2 IPMP Messages

The IPMP Messages is a very important part of the MPEG-4 IPMP Extension specification, because the MPEG-4 IPMP Extension is a message framework. The IPMP Messages enable the dynamic updating of the IPMP system. These messages carry the time-varying IPMP information for the IPMP Tools or Terminal to change the configuration simultaneously with the content consumption. The IPMP Messages can be divided into two types, IPMP Tool Message, and IPMP Device Message.

IPMP Tool Message

This kind of message is to carry the IPMP information for interaction between the IPMP Tools or to carry the IPMP information from the content for updating the IPMP Tools. The basic data structure of IPMP Tool Message is shown as follows, and there are two kinds of IPMP Messages extending from it, IPMP Message From Bitstream, and IPMP Tool Descriptor From Bitstream [1].

```

Aligned(8) abstract expandable(228-1)class IPMP_ToolMessageBase: bit(8) tag = 0 {
    bit(8) Version;
    bit(32) Msg_ID;
    bit(32) sender;
    bit(32) recipient;
}

```


The addressing of the IPMP Tool Message is specified by the Context ID of the IPMP Tools. When we instantiating an IPMP Tool, the IPMP Tool must be assigned a unique Context ID by the Terminal or IPMP Tool Manager for addressing. The Context ID is 32-bit long. In the IPMP Tool Message, the sender and recipient of this message is clearly specified by this unique Context ID. However, the generation of the Context ID is not specified in the specification, it is an implementation issue.

The 32-bit `Msg_ID` is assigned to the message when the message is generated. This ID is assigned by the generator of the message and all messages sent in response to the message shall include the ID of the original message.

```
class IPMP_MessageFromBitstream extends IPMP_ToolMessageBase :
    bit(8) tag = IPMP_MessageFromBitstream_tag
{
    bit(8) numMessages;
    IPMP_StreamDataUpdate message[numMessages];
}
```

Above, we show the data structure of the IPMP Message From Bitstream [1]. The IPMP Message From Bitstream is extended from the `IPMP_ToolMessageBase` with the tag value, 0x01. It is to deliver `IPMP_StreamDataUpdates` [1] received in the content to the IPMP Tools specified in the `IPMP_StreamDataUpdate`. `IPMP_StreamDataUpdate` could be viewed as a container for the `IPMP_DataBaseClass` [1]. The data structure of `IPMP_StreamDataUpdate` is shown as follows.

```
aligned(8) expandable(228-1) class IPMP_StreamDataUpdate
{
    bit(16) IPMPS_Type;
    if (IPMPS_Type == 0)
        bit(8) URLString[sizeOfInstance-2];
    else if (IPMPS_Type == 1){
        bit(16) IPMP_ToolDescriptorID;
        bit(8) numOfData;
        IPMP_Data_BaseClass[numOfData] IPMP_ExtendedData
    }else{
        bit(8) IPMP_data[sizeOfInstance-2];
    }
}
```

IPMPS_Type specifies the type of IPMP system. If the IPMPS_Type equals to zero, it is forbidden, and a remote IPMP_StreamDataUpdate is indicated by the URLString. The IPMPS_Type with value 0x0002 to 0x2000 are reserved for future ISO use, and the IPMP_StreamDataUpdate conveys the opaque data, IPMP_data to IPMP Tools. IPMPS_Type with value 0x0001 indicates a MPEG-4 IPMP Extension system, and the IPMP_StreamDataUpdate carries the IPMP_ExtendedData for IPMP Tools.

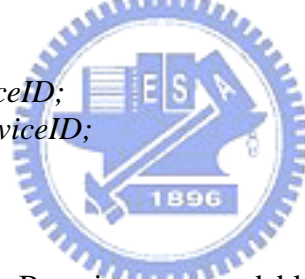
IPMP Device Message

The IPMP Device Message is defined in the annex I in the specification and is to deliver the IPMP information in the distributed environment. In the distributed environment, every IPMP device should be assigned a unique 128-bit Device ID. The data structure of the IPMP Device Message is shown bellow [1].

```

Aligned(8) abstract expandable(228-1)class IPMP_DeviceMessageBase: bit(8) tag = 0
{
    bit(8) Version;
    bit(128) sender_deviceID;
    bit(128) recipient_deviceID;
    bit(32) Msg_ID;
}

```



The IPMP_DeviceMessageBase is an expandable base class and all the tags extended from it are shown bellow [1].

8-bit Tag Value	Symbolic Name
0x00	Forbidden
0x01	IPMP_RequestContent_tag
0x02	IPMP_ResponseToContentRequest_tag
0x03	IPMP_ContentTransfer_tag
0x04	IPMP_RequestTool_tag
0x05	IPMP_ResponseToToolRequest_tag
0x06	IPMP_DeviceID_Broadcasting_tag
0x07	IPMP_DeviceID_Received_tag
0x08	IPMP_InitAuthentication_tag
0x09	IPMP_MutualAuthentication_tag
0x0A	IPMP_SecureMessage_tag
0x0B – 0xCF	ISO Reserved
0xD0 – 0xFE	User Defined
0xFF	Forbidden

2.3.3 Using IPMP

In this section, after describing all the associated concepts, we give a simple example on

using the IPMP Extension. The following chart, Figure 4, shows the basic idea of MPEG-4 IPMP Extension system. We describe the scenario of content consumption step by step as follows.

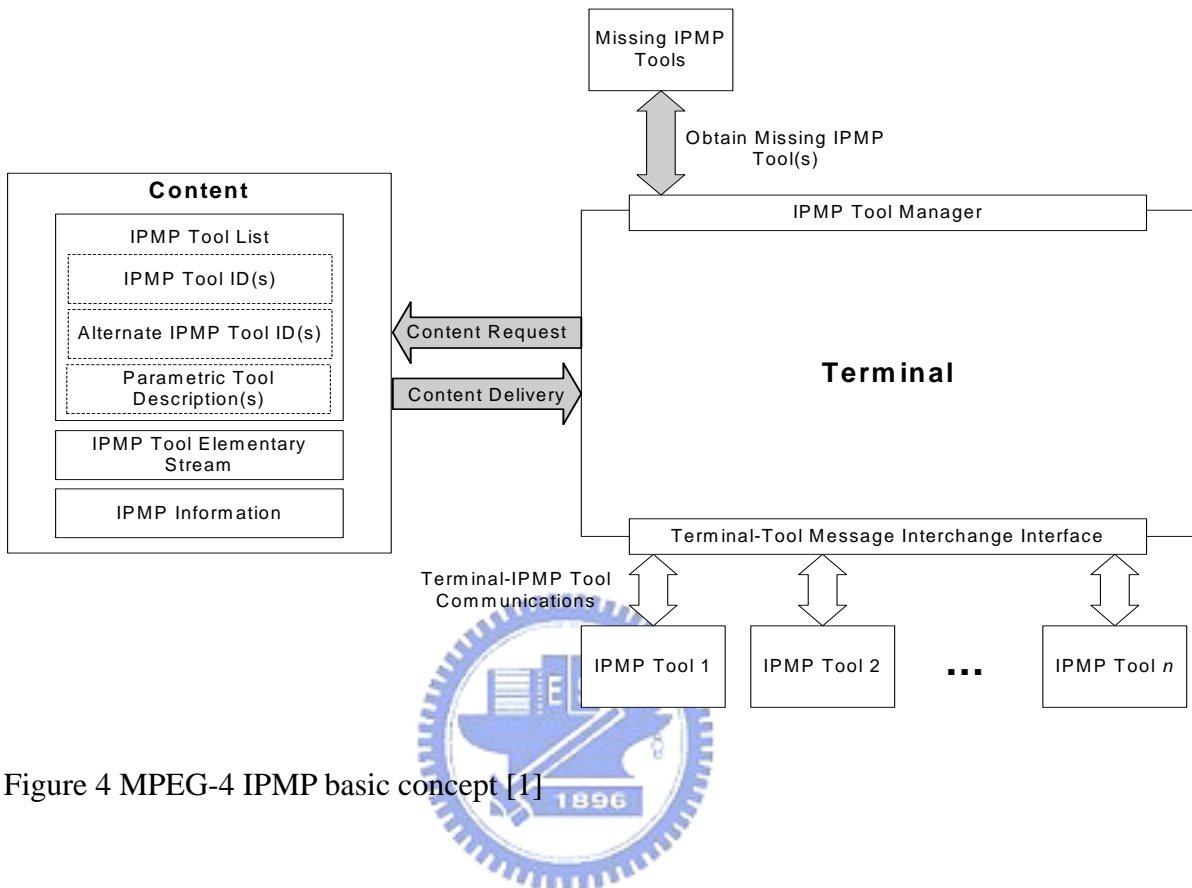


Figure 4 MPEG-4 IPMP basic concept [1]

1. User request specific content

The standard does not specify that how the content is requested. However, the following recommendations are made for the order in which different part of the Content are received and used. First, the IPMP requirement should be placed with or before media requirement. Second, before delivery of the media content, access IPMP information and/or restrictions should be done. For example, if we play a media file that is in the MPEG-4 format and is protected by the IPMP extension, we first access the IPMP requirement and some IPMP information before we really start to access the media content. It is quiet reasonable that the terminal should access and own the information that protects the media content before accessing the media data.

2. IPMP Tool Descriptor access

Before the content consumption, the terminal should access the Initial Object Descriptor

first. So, the terminal accesses the IPMP Tool List Descriptor within the IOD to get the list of the IPMP Tools that are required for the content consumption. And, according to the IPMP Tool List Descriptor, terminal determines the tools that are required to consume the media content. The IPMP Tool Descriptor is also conveyed within the IOD, and should be accessed by the terminal here.

3. IPMP Tool Retrieval

The method to retrieve IPMP Tools is not specified in the standard. However, missing IPMP Tools could be retrieved from a website or other remote device. The missing IPMP Tools may be retrieved from an IPMP Tool Stream if available.

4. Instantiation of IPMP Tools

The IPMP Tools required to consume the Content are instantiated locally or remotely according to the IPMP Tool List Descriptor received before. Then, the IPMP Tool instances are provided with IPMP Tool Descriptors for their initialization. The IPMP Tool Descriptor that results in the instantiation of the IPMP Tool contains the IPMPInitialize information that provides the IPMP Control Point code and the sequence code to inform the Terminal to instantiate the IPMP Tool at the right position.

5. Initialize and update the IPMP system

After setting up the whole IPMP system according to the Initial Object Descriptor, the content consumption begins. With the content consumption, the IPMP information for updating the IPMP system is conveyed within the IPMP ES or the OD update command. The updating information is received and turned into the IPMP messages that are routed by the IPMP Message Router. And, there are some IPMP messages for the negotiation between the IPMP Tools are routed to corresponding IPMP Tools by the Message Router, too. All the steps described above can be requested again during the content consumption, and the request may be implicit within the process or be request by the User.

Overview of MPEG-21 Resource Delivery Test Bed

The main purpose of this platform is to provide a flexible and fair environment for evaluating streaming technologies for MPEG-4 contents over IP network. Because of the modulization of this test bed, users can easily replace the components by their own designs into the system. For example they can replace the media codec to evaluating the decoding algorithm, or replace the streamer to simulate the performance of different streaming methods. With this test bed, users can simulate different channel characteristics of various networks [6].

The overall architecture of the test bed is shown in Figure 5. The entire test bed can be divided into three parts: Server, Client, and the Network Emulator. We give an overview of these parts as follows.

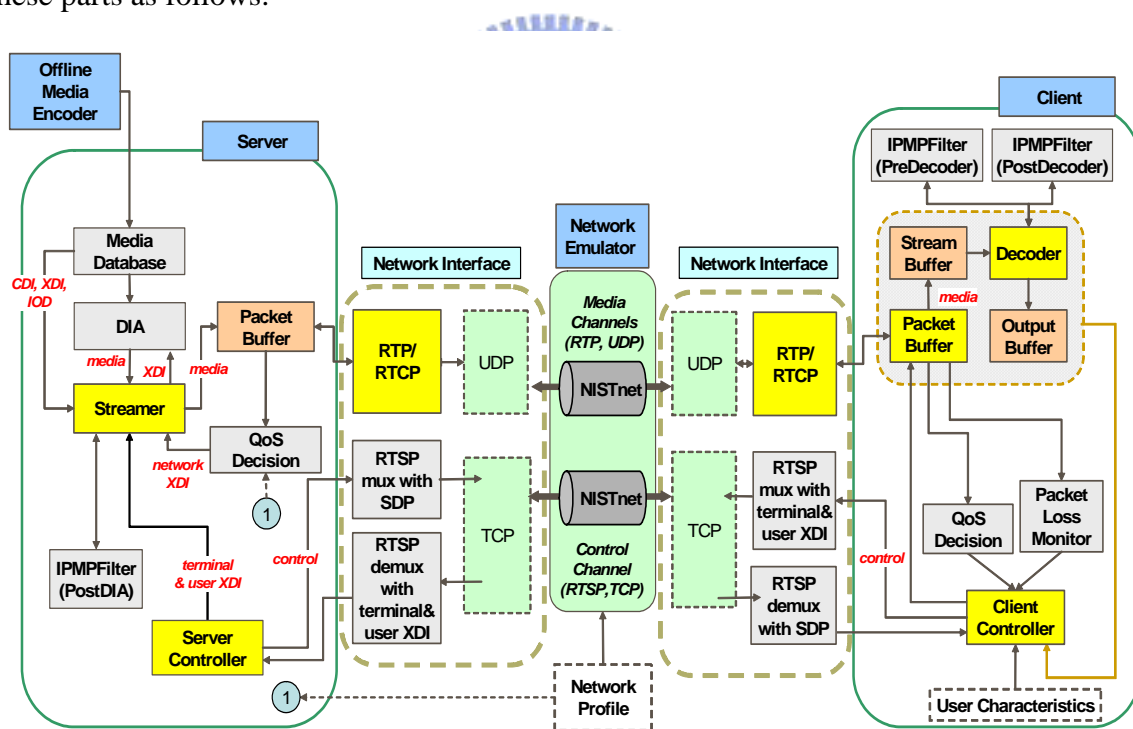


Figure 5 Architecture of MPEG-21 Testbed [5]

The Server system first gets ready to accept the request for specific media content form the Client. A Client is then connected to the streaming Server. It sends a request using the RTSP message, DESCRIBE, to the Server. In this message, the terminal capability, and user

characteristics, etc. are included. Then, the Server prepares the requested file and sets up the streaming system in order to stream out the media content over the Network. There is an acknowledging message called DESCRIBE_ACK from Server to Client to inform the Client that the media data at the Server side is ready. An RTSP message, SETUP, is then sent from Client to Server to request for setting up the transport session for the requested media content. After setting up the RTP channel for content delivery, the Server acknowledged the Client with a message called SETUP_ACK. Receiving this acknowledging message, the Client sends the message, "PLAY" to inform the Server that the Client is ready for content consumption. Then, the content consumption begins after the PLAY_ACK message is sent to the Client from the Server side. During the content consumption, whenever the user stops the procedure of the content consumption, an RTSP message, TEARDOWN, is sent from Client to Server to stop the content delivery [6].

We now give some more details about the modules inside the test bed system. There are mainly three parts: Server, Client, and Network Emulator. There are several common modules in between Server and Client. So, the descriptions below are divided into four parts, Server components, Client components, Common components, and Network Emulator.

3.1 Server components

The Server part acts as a media streaming server to provide digital media content streaming service to the Clients. One Server can provide services to several Clients. There are seven components within the Server: Media Database, DIA (Digital Item Adaptation), Streamer, Packet Buffer, QoS Decision, Server Controller, and IPMP Subsystem.

The functionalities of each component are described below. The IPMP Subsystem and QoS Decision will be discussed later.

Media Database

All the Media Contents are offline encoded and stored in the Media Database. The Media Database is responsible for opening the file of the Media Content that the Client requests. [6]

DIA

DIA (Digital Item Adaptation) performs media resource adaptation by the DIA processing engine. The CDI (Content Digital Item) and static XDI (Context Digital Item) information is required for initialization of DIA. And DIA receives the information from the Server Controller and setups its process with the provided information. [6]

Streamer

When a specific Media Content is requested by the Client, it is moved from the database to the Streamer. The Streamer accepts commands from the Server Controller, and segments a requested bitstream into video packets according to the MPEG-4 specification. [6]

Server Controller

Server Controller handles the control messages, such as REQUEST, SETUP, PLAY, TEARDOWN, that come from the Client side through the RTSP channel. After receiving these messages, the Server Controller processes these messages and set up the system for processing different protocols. [6]

3.2 Client components

The Client part is to consume the digital media content coming from the Server. The Client receives the media data from the server through the Network Interface and playbacks the media content such as video, audio, or both. There are eight components at Client side: Packet Buffer, Stream Buffer, Decoder, Output Buffer, Packet Loss Monitor, QoS Decision, Client Controller, and IPMP Subsystem. The functionalities of each component are described below. The IPMP Subsystem and QoS Decision will be discussed later.

Stream Buffer

The component Stream Buffer is to store temporally the bitstream data unpacketed from the Packet Buffer. And the data saved in the Stream Buffer will be accessed by the decoder for decoding process. The Stream Buffer is designed as a circular buffer. And there is a flag that records the current access location in the Stream Buffer. When there is no data in the Stream Buffer, then Decoder holds the decoding process and the Stream

Buffer gets the data from the Packet Buffer for the following decoding process. The decoding process restarts when the Stream Buffer is fulfilled. [6]

Decoder

The Decoder here is a multimedia decoder component that is to fetch the coded units from the Stream Buffer and to output the decoded data to the Output Buffer for displaying. The real decoding process is done after the coded units fetched by the Decoder. Here the decoding process may be for video or for audio. [6]

Output Buffer

The Output Buffer component is to save the decoded data that comes from the Decoder temporarily. The Decoder should dump the decoded data into this component. Then, the output device could access the decoded data and display. [6]

Packet Loss Monitor

The Packet buffer Monitor here is to handle the lost packet and generates retransmission messages. In the current implementation, the standard RTSP GET_PARAMETER method is used to request for packet retransmission. The component checks the Packet Buffer for the lost packets. The trigger of the checking behavior is the Client-side timer. If there any packet loss has been notified, Packet Loss Monitor informs the Client Controller to issue the retransmission request to the Server side. [6]

Client Controller

The Client Controller is to integrate all the components at the Client side and to controls these components. And the Client Controller watches the GUI user inputs then processes these inputs. When there are any lost packets that detected by the Packet Loss Monitor, the Client Controller is informed to send the retransmission request for the lost packets. [6]

3.3 Common components

In this section we introduce the modules that are shared by both client and server.

Packet Buffer

Real-time Transport Protocol/ Real-time Control Protocol (RTP/ RTCP, RFC-1889) is used as the media transport mechanism. The Packet Buffer implements an RTP packet buffer data structure.

At the Server side, after being packeted by the Streamer, all the video packets are buffered in the Packet Buffer as the RTP payload. Then, the video packets are sent out according to the pre-scheduled time set by the Streamer. However, at the Client side, the Packet Buffer is to buffer all the packets that come from the Network. This component is used at both Server and Client side with different initializations. [6]

QoS Decision

This component estimates the channel condition and provides some QoS information for rate adaptation. Here, the channel condition is the network profile designed in the system. Therefore, users could modify the network profile as they wish to test different channel conditions. [6]



IPMP Subsystem

The IPMP Subsystem exists at both Client and Server sides. It performs the functionalities of intellectual property protection, such as encryption, decryption, watermarking insertion, and extraction. There are several sub-modules inside the IPMP Subsystem. They are Message Router (MR), Tool Manager (TM), IPMP Filters or IPMP Control Points, IPMP Tools, and Terminal. Because the MPEG-4 IPMP system is a message-based infrastructure, we need a module called Message Router to route the messages to corresponding destination. The messages may come from IPMP Tools, Terminal, or other IPMP devices. The Tool Manager manages the IPMP Tools, such as maintaining the Tool mapping table, or retrieving the missing IPMP Tools from a remote site. Then, the IPMP Tools is to perform the IPMP functionalities. There can be several IPMP Tools working within one IPMP system. For instance, at Client side, it can be an DES decryption algorithm, an video watermarking extractor, and an authentication tool. The IPMP Filter is an access point where the IPMP Tools can exercise their functions. The Terminal here plays the interface between the test bed system and the IPMP system.

The description of the IPMP system in the above is very rough. The details about the IPMP system in the test bed will be discussed later.

3.4 Network

In order to connect the Server and the Client and to a simulated transmission channel, a standard RTSP/RTP-based network interface is used. Three categories of network protocols including the network-layer protocol, transport protocol, and session control protocol are adopted as shown in Figure 6.

Application Control Commands	Layered Video Data	
	Base Layer	Enhancement Layer
RTSP	RTP/RTCP	
TCP	UDP	
IP		
Data Link		
Physical Layer		

Figure 6 Network protocol [6]

The Real-time Transport Protocol (RTP) is to transmit multimedia streams from end to end. And the Real-time Streaming Transport Protocol (RTSP) is used to transmit the control messages reliably. RTSP specifies the messages and procedures to control the media streaming passing through an established channel. There are four basic message types used in the test bed: DESCRIBE, SETUP, PLAY, and TEARDOWN. The DESCRIBE message is sent from Client to Server for requesting a specific media content. It provides the information such as terminal capability or user characteristics to the Server. Then, the SETUP message is to setup a media delivery session between Server and Client with the information contained in the DESCRIBE message. After setting up the channel, Client sends a PLAY message to inform the Server to start transporting the media content via the set up session. Finally, the TEARDOWN message is to end the transport and to close the session.

Here, the test bed adopts an IP network emulator named NISTnet [9] to provide

repeatable network environments. It can emulate practical wide-area heterogeneous network environments. It is a LINUX based IP network emulator developed by the National Institute of Standard Technology, USA. There are many controllable parameters that could be set by user, such as packet loss ratio, jitter, bandwidth variation, and delay.



IPMP Reference Software

In this chapter, we introduce two IPMP reference softwares, IM1, and PSL MPEG-2 IPMP Extension reference software.

4.1 IM1 IPMPX Reference Software

IM1, the AHG on System Reference Software Implementation is a group that is responsible for developing and integrating the MPEG-4 system part reference software for MPEG committee. Here, the software implementation is one of the implementations of MPEG-4 system part. We only focus on the IPMP related parts. Figure 7 shows the software structure of IPMP system in the IM1. As description before, the IPMP system are mainly seperated into three parts, IM1 Terminal, Tool Manager, and Message Router.

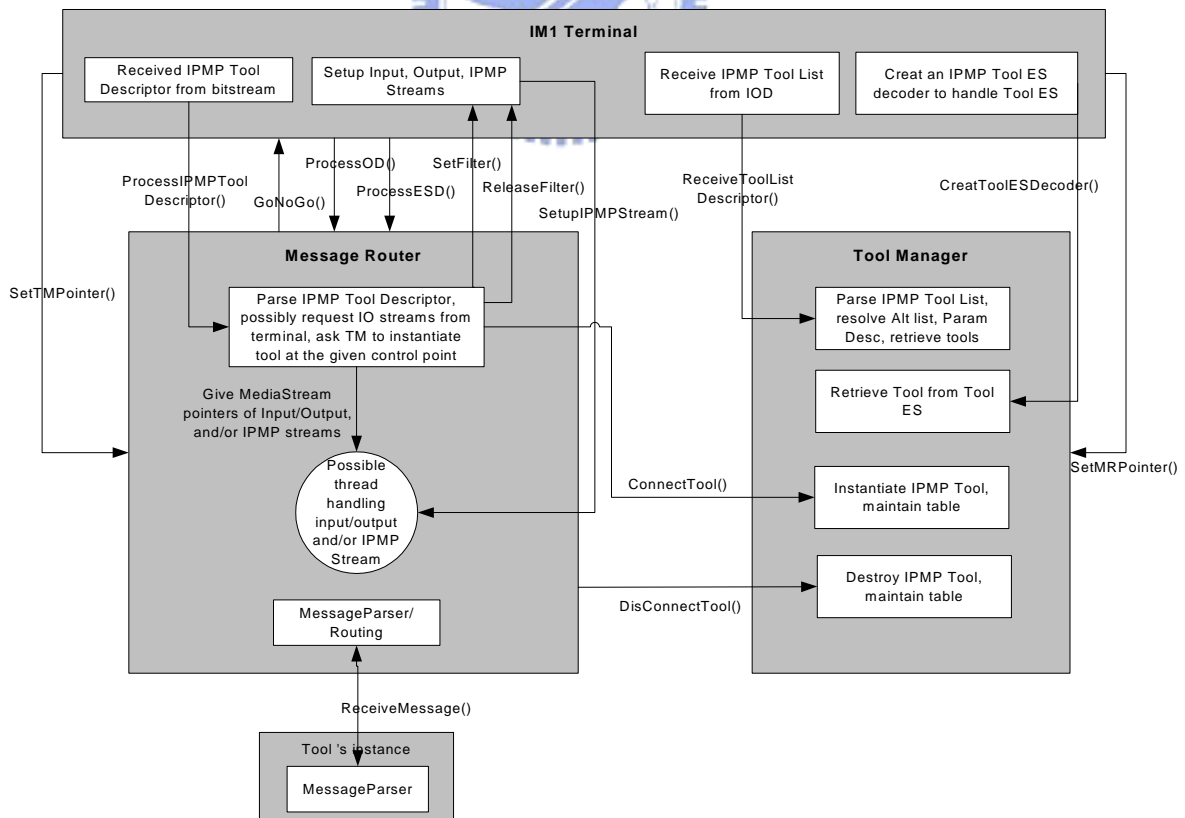


Figure 7 IPMPX system in IM1 [3]

In IM1, the Message Router and the Tool Manager are composed of some method to achieve their function, that is, they only define the interface of the Tool Manager and the Message Router. Most member functions of Message Router and Tool Manager are declared as virtual functions. The class hierarchy is shown in Figure 8, which clearly specifies the relationship between the main classes.

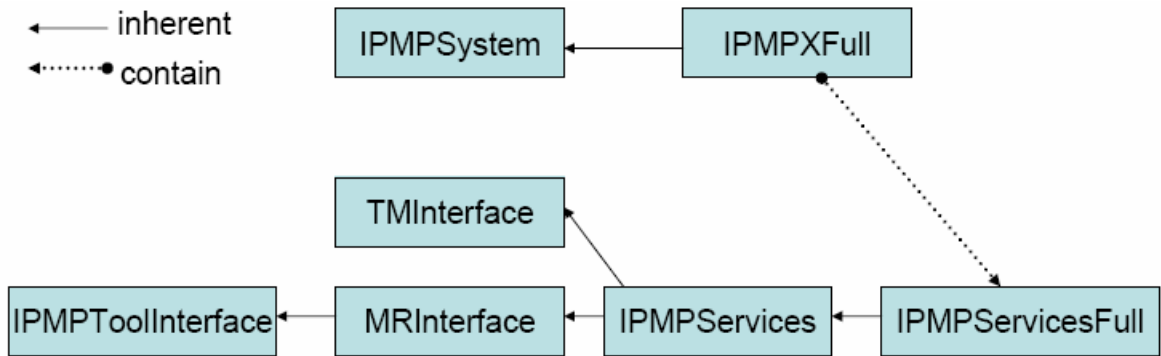


Figure 8 IPMPX class hierarchy in IM1

We can easily obtain that there are mainly four basic classes, IPMP System, TMInterface, MRInterface, and IPMPToolInterface.

4.1.1 IPMPToolInterface

IPMPToolInterface is an abstract class that should be implemented by any IPMP Tools. It has the API as the follows.

```

/**
 * messaging interface.
 * @param size the size of the message
 * @param message one of the messages defined in MPEG-4 IPMP Extension, or a
 * user-defined message
 * @return true if successful
 */
virtual bool ReceiveMessage(ToolMessage*) = 0;
  
```

4.1.2 MRInterface

MRInterface is the main interfaces that implement the functionalities of a Message

Router. Because the MRInterface inherit the class, IPMPToolInterface, MRInterface must implement the API « ReceiveMessage(ToolMessage*) » also. And it is a conceptual entity with its APIs as follows.

```

/**
 * Setting tool manager's pointer, so that the MR can use it to access TM's member
 functions
 * @param tmPointer a pointer to an object derived from TM_interface
 * @return true if successful
 */
virtual bool SetTMPointer( TM_Interface * tmPointer) = 0;

/**
 * Called by the Terminal to process an OD
 * @param pOD pointer to the object descriptor to be processed by the MR
 * @return true if successful
 */
virtual bool ProcessObjectDescriptor( ObjectDescriptor* pOD ) = 0;

/**
 * Called by the Terminal to inform the MR that an OD is not valid anymore.
 * @param pOD pointer to the OD to be removed
 * @return true if successful
 */
virtual bool RemoveObjectDescriptor( ObjectDescriptor* pOD ) = 0;

/**
 * Called by the Terminal to process an IPMPToolDescriptor.
 * @param descriptor the IPMPToolDescriptor to be processed by the MR
 * @return true if successful
 * NOTE: this descriptor object will be deleted after calling this method. The IPMPX
 * system shall take this into account and copy it into its own memory space if necessary
 */
virtual bool ProcessIPMPDescriptor(IPMP_Descriptor* descriptor) =0;

/**
 * Called by the Terminal to inform the MR that an IPMPToolDescriptor is not valid
 anymore.
 * @param descriptor pointer to the IPMPToolDescriptor to be removed
 * @return true if successful
 */
virtual bool RemoveIPMPDescriptor(IPMP_Descriptor* descriptor) = 0;

/**
 * Called by the Terminal to process an Elementary Stream Descriptor.
 * @param pESD pointer to the Elementary Stream Descriptor to be processed by the
 MR

```

```

    * @return true if successful
    */
    virtual enum ACCESS_PERMISSION ProcessESDescriptor(ES_Descriptor* pESD) = 0;

    /**
     * Called by the Terminal to inform the MR that an ESDescriptor is not valid anymore.
     * @param pESD pointer to the Elementary Stream Descriptor to be removed
     * @return true if successful
     */
    virtual bool RemoveESDescriptor( ES_Descriptor* pESD ) = 0;

    /**
     * Called by the Terminal to create a sink for an IPMP stream.
     * @param pOD a pointer to the Object Descriptor where the ipmp stream is declared
     * @param pESD a pointer to the relevant Elementary Stream Descriptor
     * @param ipmpStream the MediaStream object handling the IPMP stream
     * @return true if successful
     */
    virtual bool SetUpIPMPStream(ObjectDescriptor *pOD, ES_Descriptor* pESD,
    MediaStream* ipmpStream) = 0;

    /**
     * Called by the Terminal to destroy a sink for an IPMP stream, that was previously
     * created by the Terminal
     * using the SetUpIPMPStream method.
     * @param pESD a pointer to the relevant Elementary Stream Descriptor
     * @return true if successful
     */
    virtual bool RemoveIPMPStream(ES_Descriptor *pESD) = 0;

```

4.1.3 TMInterface

The TMInterface is an independant class that does not inherit any classes, and is to implement the conceptual entity of IPMP Tool Manager. It has the APIs as follows.

```

    /**
     * setting message router's pointer, so that the TM can use it to access MR's member
     * functions
     * @param tmPointer the pointer to the Message Router
     * @return true if successful
     */
    virtual bool SetMRPointer( MR_Interface * tmPointer) = 0;

    /**
     * Called by the MR to indicate a tool is no longer needed.
     * @param toolPtr the pointer to the tool to be disconnected
     * @return true if successful

```

```

*/
virtual bool DisconnectTool( void* toolPtr) = 0;

/**
 * Called by the MR to request a tool be instantiated.
 * @param OD_id the id of the Object Descriptor in which scope the tool will run
 * @param pESD a pointer to the ESD of the Elementary Stream protected by the tool
 * @param toolDescriptor the descriptor carrying initialization information for the tool
 * @return the pointer to the tool
 */
virtual void* ConnectTool(ES_Descriptor *pESD, IPMP_Descriptor* toolDescriptor) =
0;

/**
 * Called by the Terminal to pass a Tool elementary stream
 * @param tool_ES the interface to the media stream implemented in IM1
 * @param tool_ESD the relevant Elementary Stream Descriptor
 * @return true if successful
 */
virtual bool ReceiveToolES(MediaStream *tool_ES, ES_Descriptor *tool_ESD) = 0;

/**
 * Called by the Terminal when a Tool elementary stream is removed
 * @param pESD a pointer to the relevant Elementary Stream Descriptor
 * @return true if successful
 */
virtual bool RemoveToolES(ES_Descriptor *pESD) = 0;

/**
 * Called by the Terminal to pass the tool list.
 * @param toolList the list of tools as defined in MPEG-4 IPMP Extension
 * @return true if successful
 */
virtual bool ReceiveIPMP_ToolListDescriptor( IPMP_ToolListDescriptor* toolList ) =
0;

```

4.1.4 IPMPSystem

The class, IPMPSystem, is mainly to create and destroy the IPMPServices object that inherits the MRInterface and TMInterface. It has the member function as follows.

```

/**
 * Instantiate an IPMPServices object
 * @param szServiceURL the URL identifying this IPMP Service
 * @param toolList the list of tools contained in the initial Tool list
 * @return an IPMPServices object containing the Tool Manager and the Message
Router

```



```

*/
virtual IPMPServices *CreateIPMPServices (const char *szServiceURL,
IPMP_ToolListDescriptor* toolList) = 0;

/**
 * Delete an IPMPServices object
 * @param pIPMPServices the pointer to the IPMPServices to be removed
 * @return true if successful
 */
virtual bool RemoveIPMPServices(IPMPServices *pIPMPServices) = 0;

```

At the bottom of the whole hierarchy, there still two classes, IPMPXFull, and IPMPServicesFull. The IPMPXFull inheriting from IPMPSystem is to create and destroy the IPMPServicesFull object to perform the IPMP functionalities. And, the two classes can be viewed as the classes to implement all the virtual functions of their parents, that is, implement all the interfaces of the MR, TM, and IPMP Tool. So, we do not list the APIs of the bottom two classes, IPMPXFull, and IPMPServicesFull.



4.2 PSL MPEG-2 IPMP-X Reference Software

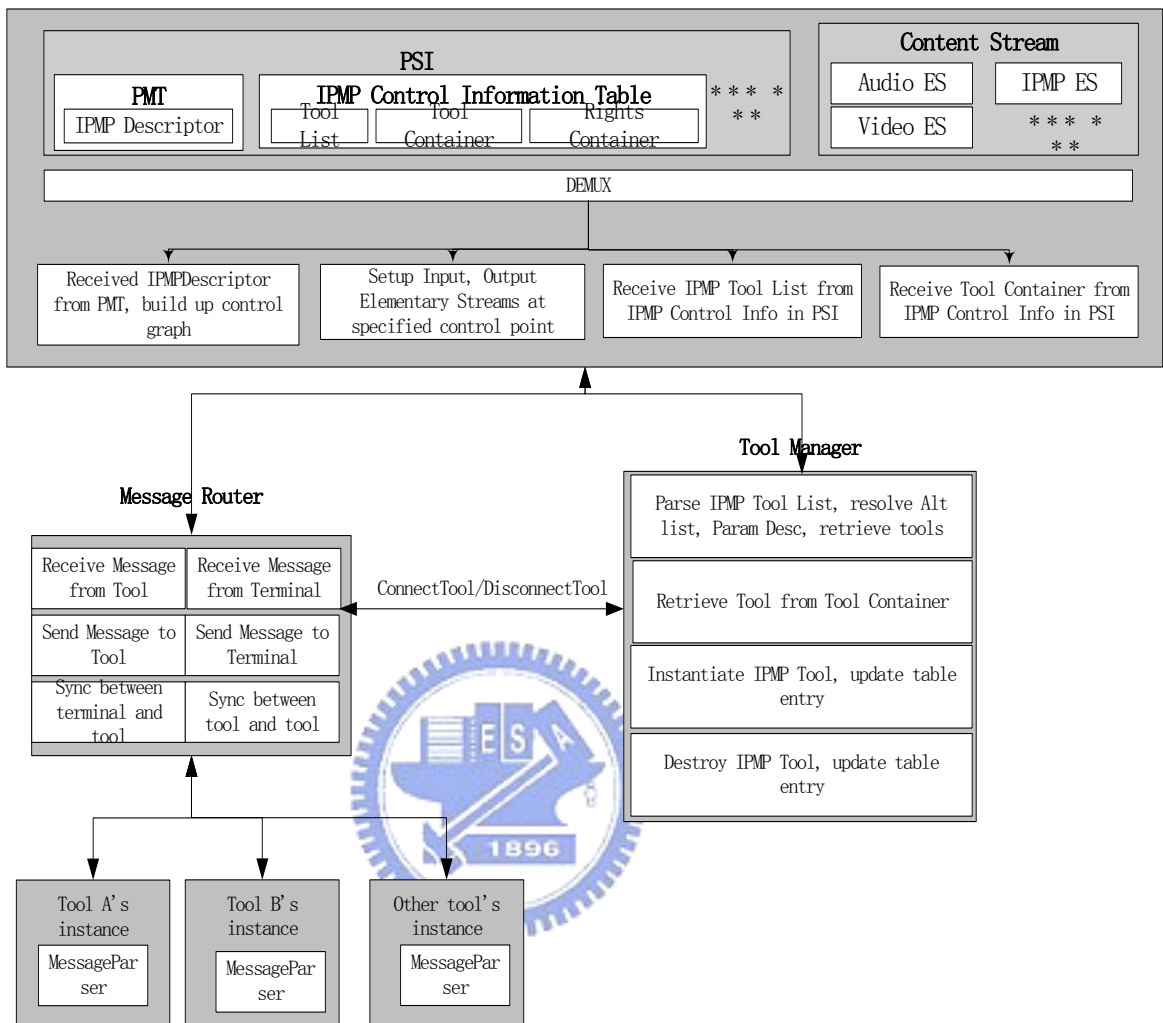


Figure 9 MPEG-2 IPMPX system software architecture [4]

Because the MPEG-2 system part is quite different from the MPEG-4 system part, we only focus on the Message Interface that implemented by PSL. Even the difference exists between the system part of MPEG-2 and MPEG-4, the IPMP messages are the same. The Message Interface of PSL includes the implementation of the IPMP messages that specified in the specification.

Basically, the MPEG-2 IPMP Extension is developed from the MPEG-4 IPMP Extension,

so, they are almost the same except the part associated with the MPEG-2 system and MPEG-4 system.

Figure 9 shows the MPEG-2 IPMPX system software architecture, and the architecture is familiar to the one in MPEG-4.

Here, we do not concern about the APIs of every module within the MPEG-2 IPMPX system. However, what we focus on is the IPMP Message structure of the IPMPX system developed by PSL. In general, we can view the top class of the whole structure as the class named BitField. All the data structure within the implementation should be turned into binary data, so all the classes inherit from the BitField class for the translation between data structure and the binary data. In the BitField class, there are four member functions that are declared as pure virtual functions. All the classes that inherit from it must implement these functions. They are listed as follows including the functionalities of them.

```
/**
 * reads the data from the buffer and stores the fields internally.
 * Must be implemented in derived classes.
 * @param buf the buffer to read
 * @return the number of bits that have been read from the buffer
 */
virtual int fromBuffer(const byte* buf) = 0;

/**
 * writes the internal fields into the buffer
 * @param buf the buffer to write
 * @return the number of bits that have been written into the buffer
 */
virtual int toBuffer(byte* buf) = 0;

/**
 * checks if the syntax is correct.
 * @return true if the syntax is correct.
 */
virtual bool checkSyntax() const = 0;

/**
 * this method must be implemented in derived classes so that it returns the
 * size in bytes of the binary message generated by the toBuffer method.
 * @return the size (in bytes) of this class.
 */
virtual unsigned int getSize() const = 0;
```

Then, the classes of the second layer are divided into many branches. Here, we only

focus on the classes associated with the IPMP Messages, the ExpandableBaseClass. In the ExpandableBaseClass, the virtual functions of BitField are implemented and it provides another pure virtual function in order to compute the size of this object when serialized into a bitstream. The declaration of the function is shown below including the description.

```

/**
 * Compute the size (in bytes) of this object when serialized into a bitstream, according
 * to the current content of the member variables.
 * This method must be overloaded by any derived class to indicate the correct size.
 * This size does NOT include the object_id and the length of the ExpandableBaseClass.
 * @return the size in bytes.
 */
virtual unsigned int sizeOfInstance() const = 0;

```

There is a special object named Length contained in the ExpandableBaseClass. The Length class is also inherits from the BitField class. This object is to record the total size of the object that inherits from the ExpandableBaseClass. The Length object has its size vary from one to four bytes. In the current byte, if there are any concatenating bytes presenting the length of the object, the first bit of the current byte will be true. So, the maximum size of the object length is 2^{28} bytes.

There are several branches expanded from the ExpandableBaseClass. They are BaseDescriptor, ByteArray, IPMP_Data_BaseClass, IPMP_DeviceMessageBase, and IPMP_ToolMessageBase.

The IPMP_DeviceMessageBase and IPMP_ToolMessageBase are the two kinds of basic IPMP Messages that contain the IPMP information as described in section 2.3.2. The IPMP_ToolMessageBase will contain the object IPMP_StreamDataUpdate inheriting directly from the BitField. And there are several classes that inherit from the IPMP_ToolMessageBase. They are IPMP_DescriptorFromBitstream, IPMP_MessageFromBitstream, and IPMP_MessageFromTool. And they are all designed to contain the IPMP information as their loads.

Several classes inherit from the class IPMP_Data_BaseClass. They are the classes that represent different IPMP information such as IPMP_GetTools, IPMP_GetToolResponse, and etc. They have the name the same with tags that extended from the IPMP_Data_BaseClass that is described in section 2.3.1.

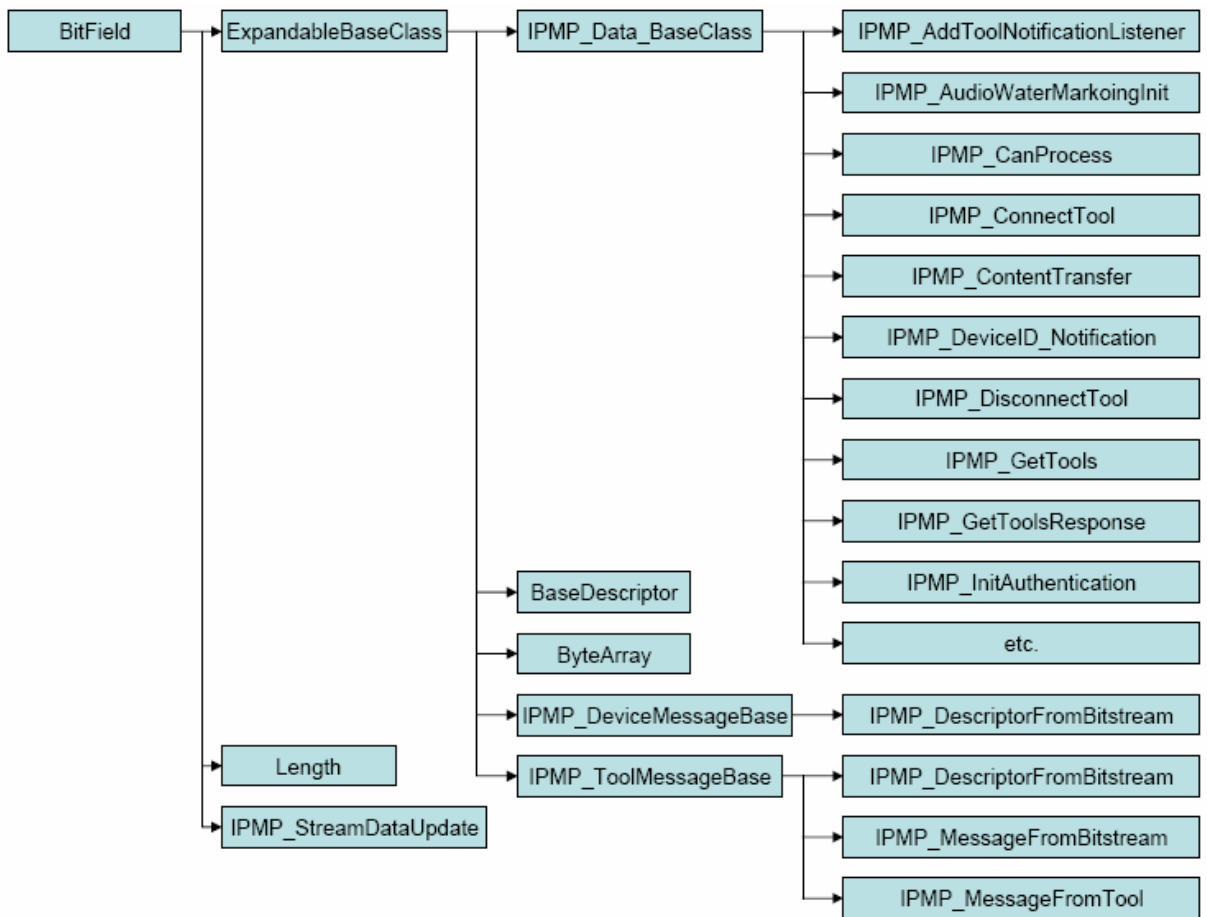


Figure 10 Class hierarchy of message interface of PSL IPMPX reference software

Figure 10 shows the overall inherent relationship between classes within PSL MPEG-2 IPMP Extension reference software.

MPEG-4 IPMP Extension System Implementation on MPEG-21

Test Bed

This chapter describes our implementing of the MPEG IPMP Extension system on the MPEG-21 test bed. Because of the existing architecture of the MPEG-21 Testbed, we have to design an architecture that matches the MPEG-21 test bed. This software architecture is modified from the MPEG-4 IPMP Extension system in IM1 and the MPEG-2 IPMP Extension system developed by PSL. However, the entire system has been restructured and the software is rewritten. The redesigned architecture contains mainly six modules. They are Message Router, Tool Manager, Terminal, IPMPFilter, Context, and IPMP Tool. The relationship among these modules is shown in Figure 11.

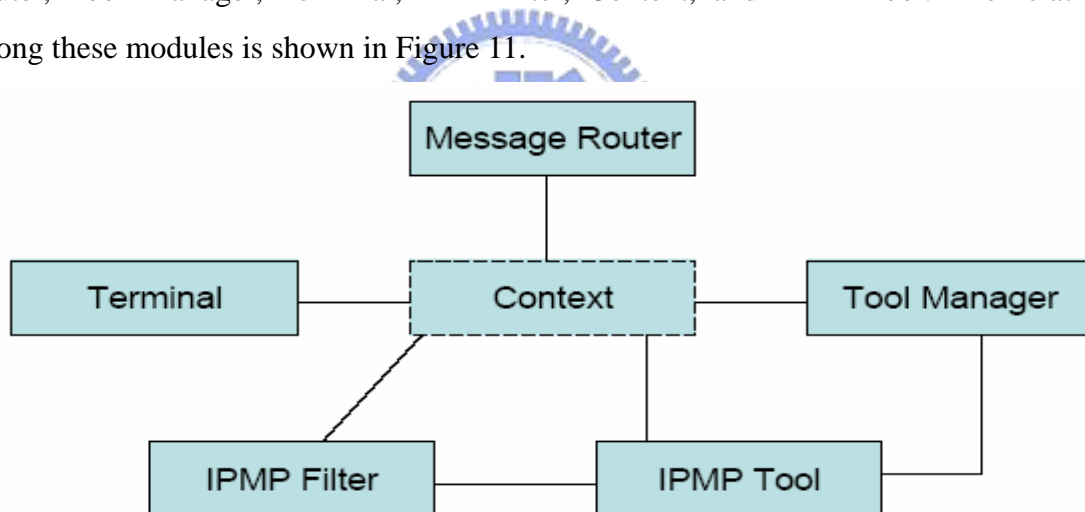


Figure 11 Relationships among modules in the IPMPX system

Here we focus on the modules, Context and IPMP Filter, because the functionalities of the others have been described in the previous sections. First, we describe the concept of the Context object. Then, we give the details about the design of the IPMP Filters. Next, the design of Message Router, Tool Manager, Terminal, and the IPMP Tool are discussed. Because MPEG-21 Testbed is separated into two sides, the Client side and the Server side, when we discuss the IPMP subsystem, we will point out the differences between the designs in both sides. Although, basically, the IPMP systems at the Client and the Server sides are

almost the same, there still some differences between them. We will also discuss the design the APIs of modules related to the IPMP subsystem.

5.1 Architecture Design

5.1.1 Context

Because there are many pieces of shared information among the modules in the IPMP system, we move all the shared information to an entity called Context rather than distributing them into individual modules. These modules including Tool Manager, Message Router, Terminal, IPMP Filters, and IPMP Tools, are connected to the Context in order to retrieve the shared information. The relationships between these modules and the Context are shown in Figure 11.

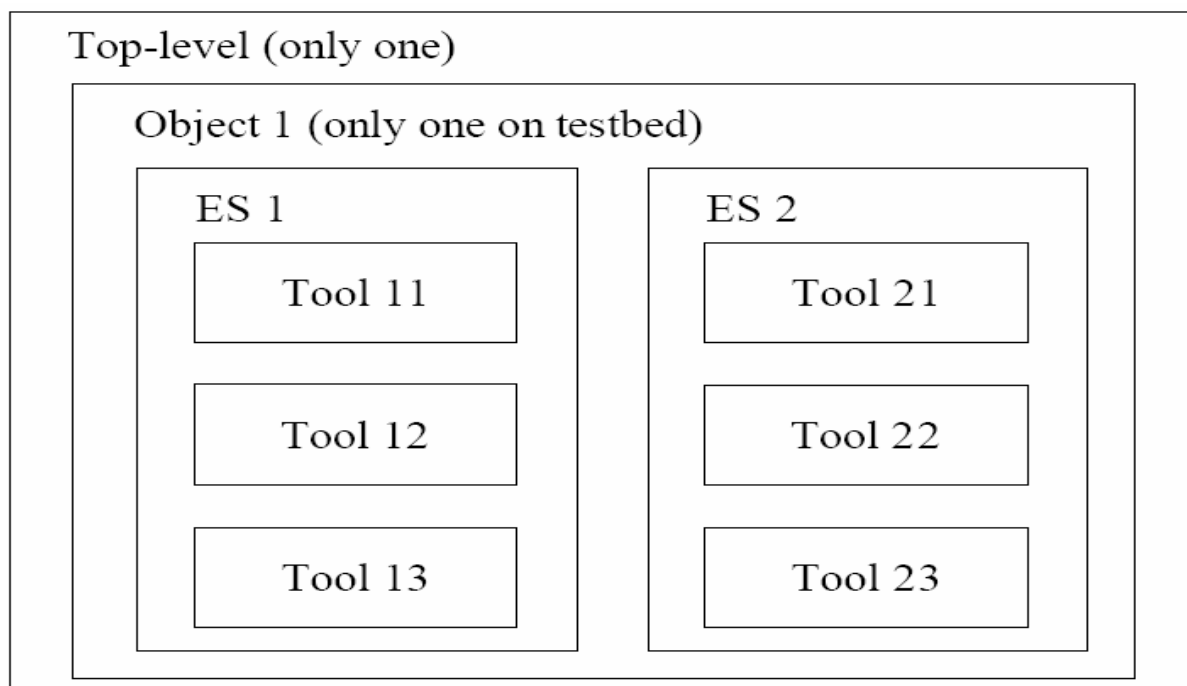


Figure 12 Context hierarchy

The Context is designed as a tree structure for easily navigating the associated shared data in it. In our design, the Context can be split into four levels, Top Context, Object Context, ES Context, and Tool Context. And the relationships between the four levels of Context are shown in Figure 12.

In the Testbed software, there is only one Top Context at each side to record the

references to Tool Manager, Message Router and Terminal. And it also bookmarks the map of the IPMP Tool Descriptor. The Context of the object level is empty in our design. In the Context at ES level, we store the information about the ES Descriptor and the references to all the IPMP Filters within the IPMP system. The bottom level of the Context in the tree structure is the Context of Tools. It records the information of the Tool Descriptor and the references to the IPMP Tools.

In order to construct the tree structure of the IPMP Context, the Contexts at any levels has the following five members, context ID, context type, parent context, children context, and top context. The context ID serves as the identifier of this context. The context type records what type the context is. The parent context and children context are the links to the tree structure. For instance, for an IPMP ES Context, it has the children contexts as Tool Contexts, and the parent context as the Object Context. We show the tree structure of the Context and its links in Figure 13. The arrow with the symbol, *, denotes that there may be multiple objects.

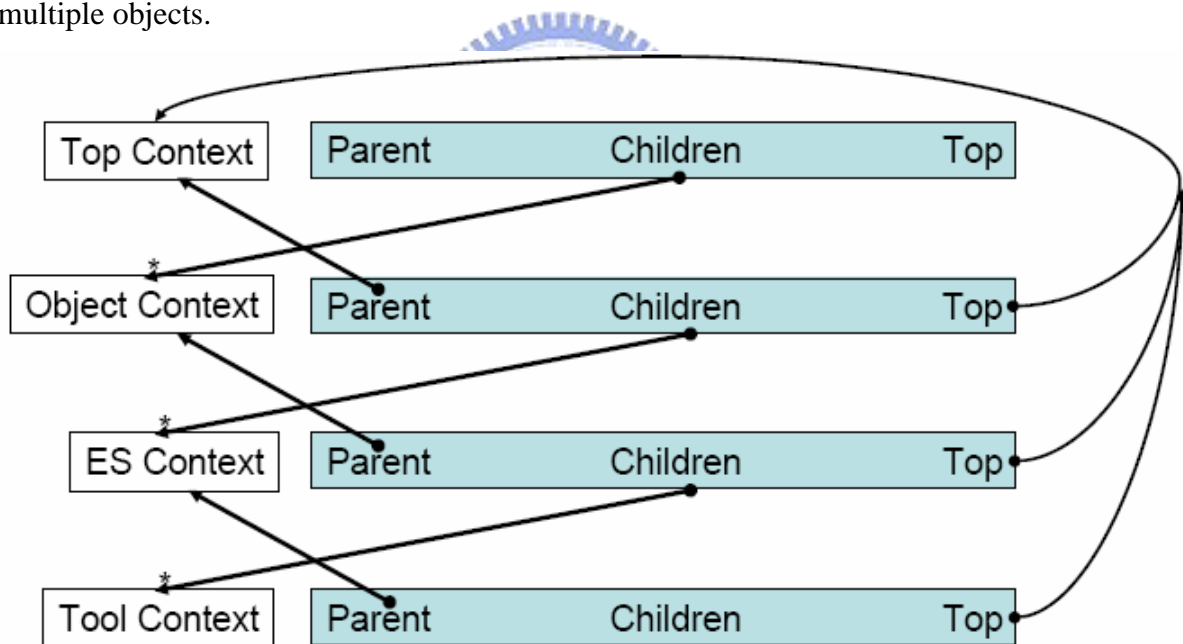


Figure 13 Links between four level contexts

As shown in Figure 13, the Context has a tree structure, and we can easily find the shared information by navigating through the context tree. The links of the context tree are discussed earlier, now we discuss the links of the contexts and the shared entity. There are different types of members in the contexts depending on the context type. Figure 14 shows the

relationships of modules of IPMP system and the four level Contexts. From the Figure 14, we observe that it is easy to find the component of the IPMP system by searching the context tree.

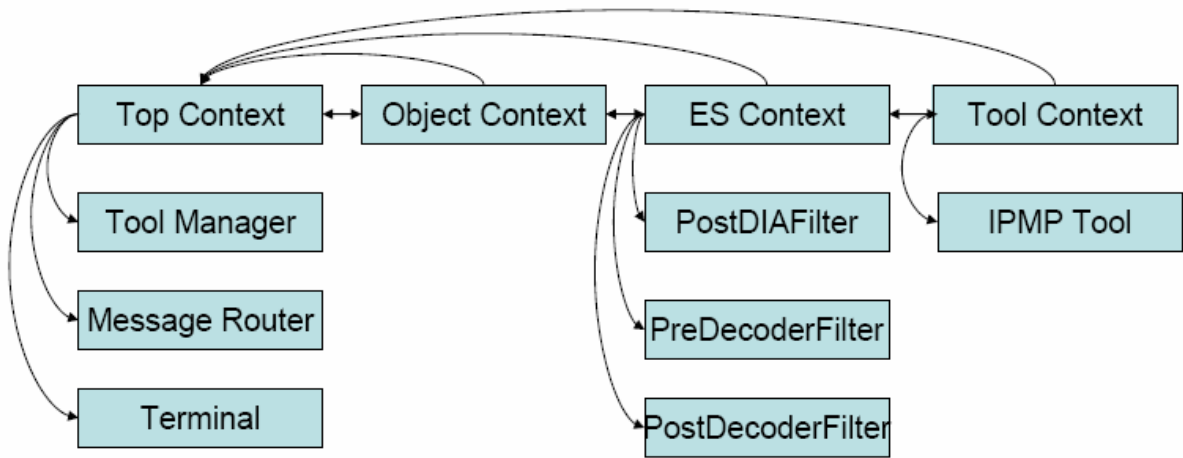


Figure 14 Search the context tree

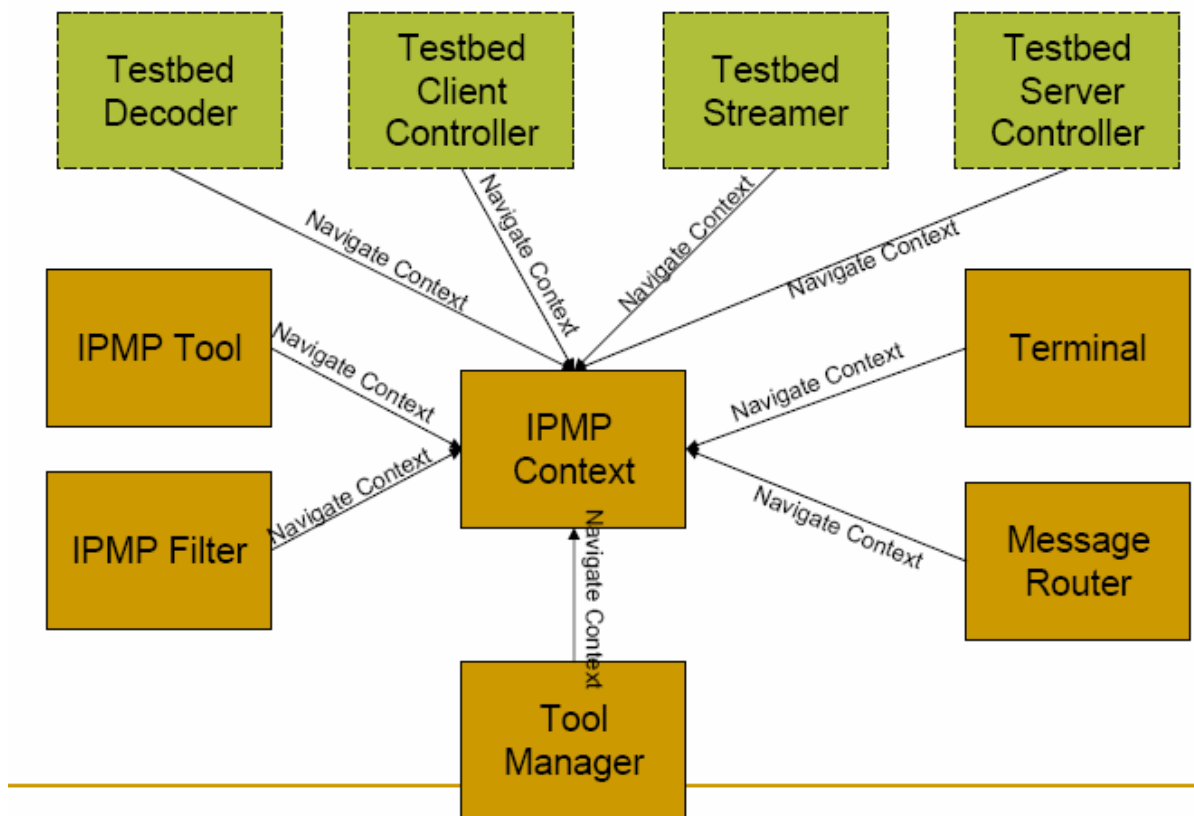


Figure 15 Relationship between IPMP Context and other modules

From the link diagram shown in Figure 14, for instance, if there is an IPMP Tool A that wants to send a message to IPMP Tool B, A first finds the context tree for the Message Router

in the Top Context, then it passes the message to the Message Router. After receiving and determining the recipient of this message, the Message Router identifies the IPMP Tool B in the Context tree, from Top to Object, from Object to ES and from ES to Tool. Then it routes the message to the IPMP Tool B. The method to find the other components is the same.

The relationship between the IPMP Context and other modules associated with IPMP Subsystem is shown in Figure 15. All modules associated with the IPMP Subsystem have relationship with IPMP Context.

5.1.2 IPMP Control Points (IPMP Filters)

Considering the original architecture of MPEG-21 Testbed, it is important to choose the appropriate IPMP Control Points and insert the IPMP Filters at those locations. By surveying the MPEG-21 Testbed, we finally allocate three IPMP Control Points on the MPEG-21 Testbed, one at the server side and the other two at the client side.

The IPMP Control Points at client side are allocated between the Stream Buffer and the Decoder, and between the Decoder and Output Buffer. In Figure 2, there is an IPMP Control Point defined in between the Decoder Buffer and the Decoder. It is mapped to the one between the Stream Buffer and the Decoder (PreDecoderFilter) in our design. And there is also one IPMP Control Point defined between the Decoder and the Composition Buffer in Figure 2. However, there is no Compositor designed in MPEG-21 Testbed, so we mapped this IPMP Control Point to the one between the Decoder and the Output Buffer (PostDecoderFilter).

Since the specification of MPEG-4 IPMPX mainly focuses on the client side only, our design on the server side is inferred from the client side design. The PreDecoder Filter at client side is mapped to the IPMP Control Point between the DIA module and Streamer at server side and is called PostDIAFilter. In current implementation, we insert an IPMP Tool to perform the decryption in PreDecoderFilter at client side, and an IPMP Tool for real-time encryption in the PostDIAFilter at server side.

The IPMP Control Point between the Decoder and the Output Buffer at client side has no corresponding IPMP Control Point at server side. It is because the encoding of the media content is done offline and the encoded media contents are stored in the module called Media Database before streaming. In theory, the PostDecoderFilter is mapped to an IPMP Control

Point between the encoder and the module providing the un-coded media content. The main purpose of PostDecoderFilter is to extract or determine the watermarking of the un-coded media content. In current design, we could perform watermarking insertion off-line, and extract it by the PostDecoderFilter.

Figure 16 shows the relationships between the IPMP Control Points and IPMP system and the modules related with the IPMP system at client side. Because the Decoder in the MPEG-21 Testbed is designed base on the MPEG-4 reference decoder, it is not a passive decoder. In other words, the Decoder is an active device that retrieves the stream data from the Stream Buffer and puts the decoded data to the Output Buffer. The original concept of IPMP Control Points is that they are somewhat like filters to filter the passing bitstream data. However, in the MPEG-21 Testbed, the filter concept does not fit well here, and we have to design the IPMP Control Points using the concept of processor.

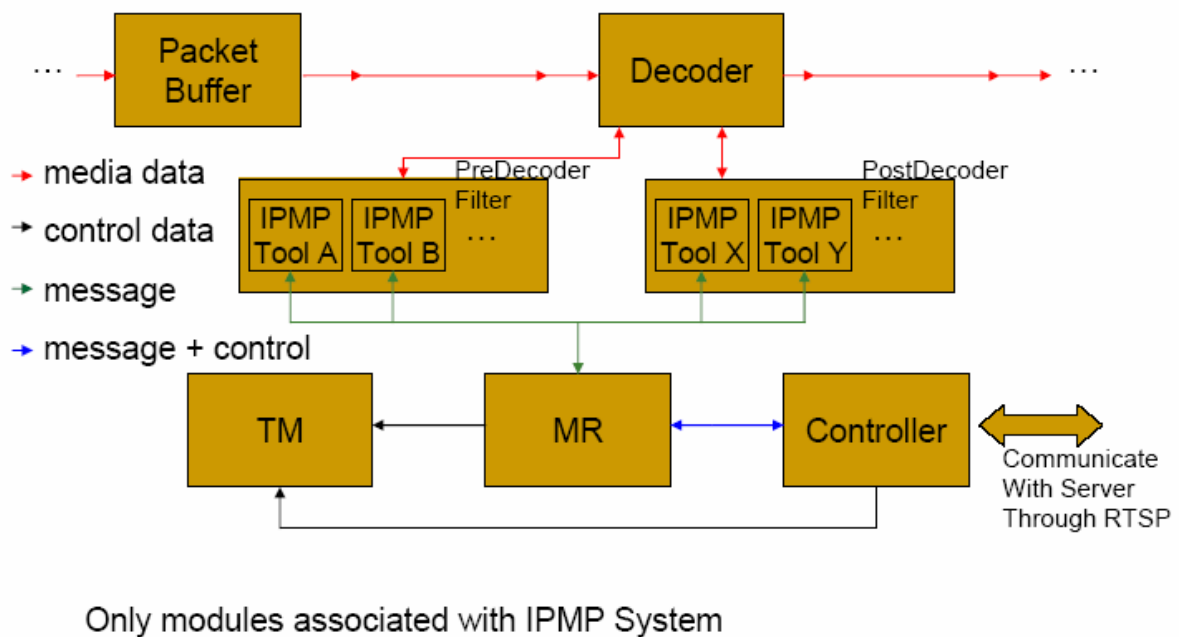


Figure 16 Block diagram of IPMPX system at client side

Since the Decoder is an active component, it decides the actions of getting the data and putting the decoded data. Hence, we design these two IPMP Filters as processors that are attached to the Decoder, and process the bitstream data as similar to the function of the Decoder. The Decoder retrieves the bitstream data from the Stream Buffer then passes data to the PreDecoderFilter immediately before decoding it. After being processed by the

PreDecoderFilter, the data are then returned to the Decoder and are decoded by the Decoder. Before putting the data to the Output Buffer, the Decoder shall pass the decoded data to the PostDecoderFilter for processing, and then puts the returned data to the Output Buffer.

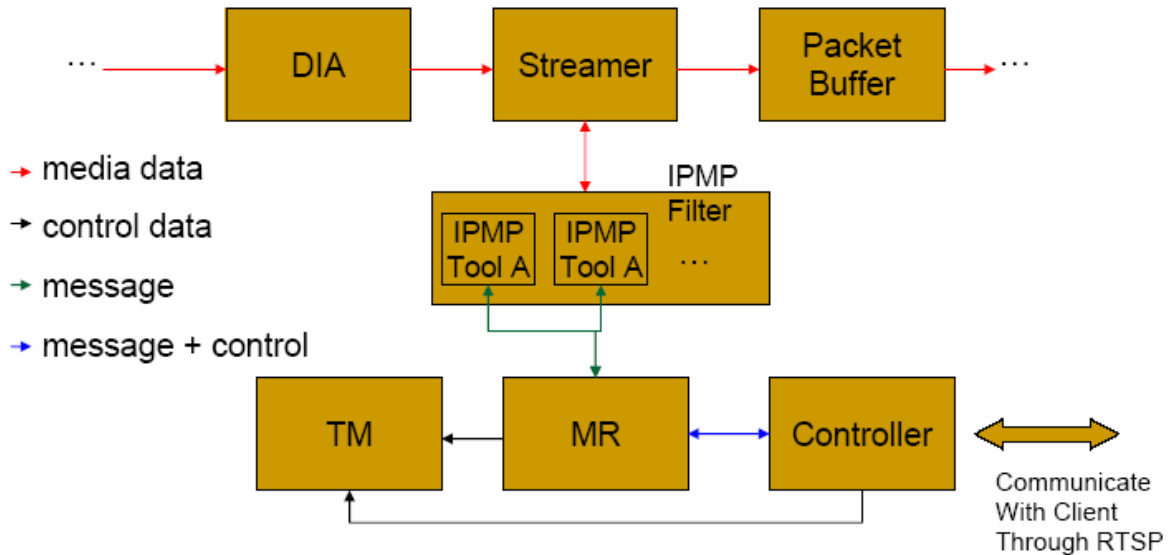


Figure 17 Block diagram of IPMPX system at server side

Because the size of the data passed to the PreDecoderFilter and of the data processed by PreDecoderFilter may be not the same, the Decoder may crash when gets data of the different size. In order to avoid this problem, we modify slightly the Decoder by inserting a buffer named “decoder buffer” into Decoder. Note here that the decoder buffer is not the Decoder Buffer appear in the specification, but a simple buffer to buffer the bitstream data for the Decoder before decoding. Therefore, this modification does not affect the software in terms of satisfying the MPEG-21 Testbed specifications. The decoder buffer holds the data for the Decoder so that the size of the data requested by Decoder is unchanged to prevent the Decoder from crashes. In the original Decoder design, the operation of data access from the Stream Buffer is spread over the entire decoding procedure, but this arrangement makes the insertion of IPMP devices complicated. The inserted buffer collects the spread operations of data access to ease the management of IPMP. The modification is show in Figure 18.

At server side, the design of the IPMP Filter, PostDIAFilter, is similar to that at client side. And the relationships between the modules associated with the IPMP system and the IPMP Control Points are shown in Figure 17. Here, the Streamer is an active module. So the

PostDIAFilter processes the bitstream as Streamer's will. The procedure of processing the bitstream data is similar to that at client side.

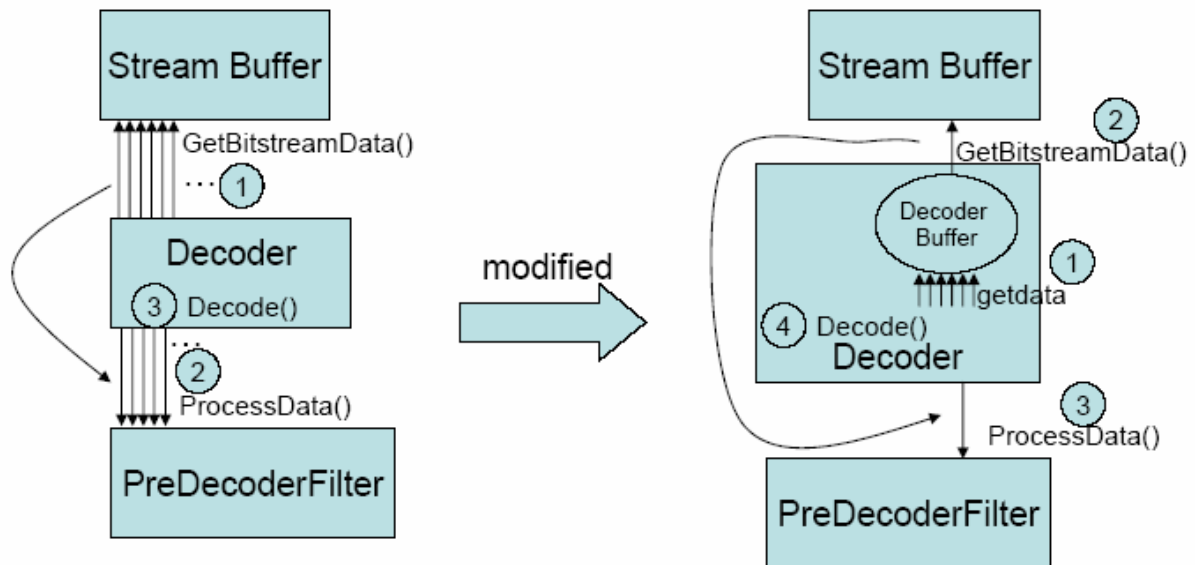


Figure 18 Modification of Decoder

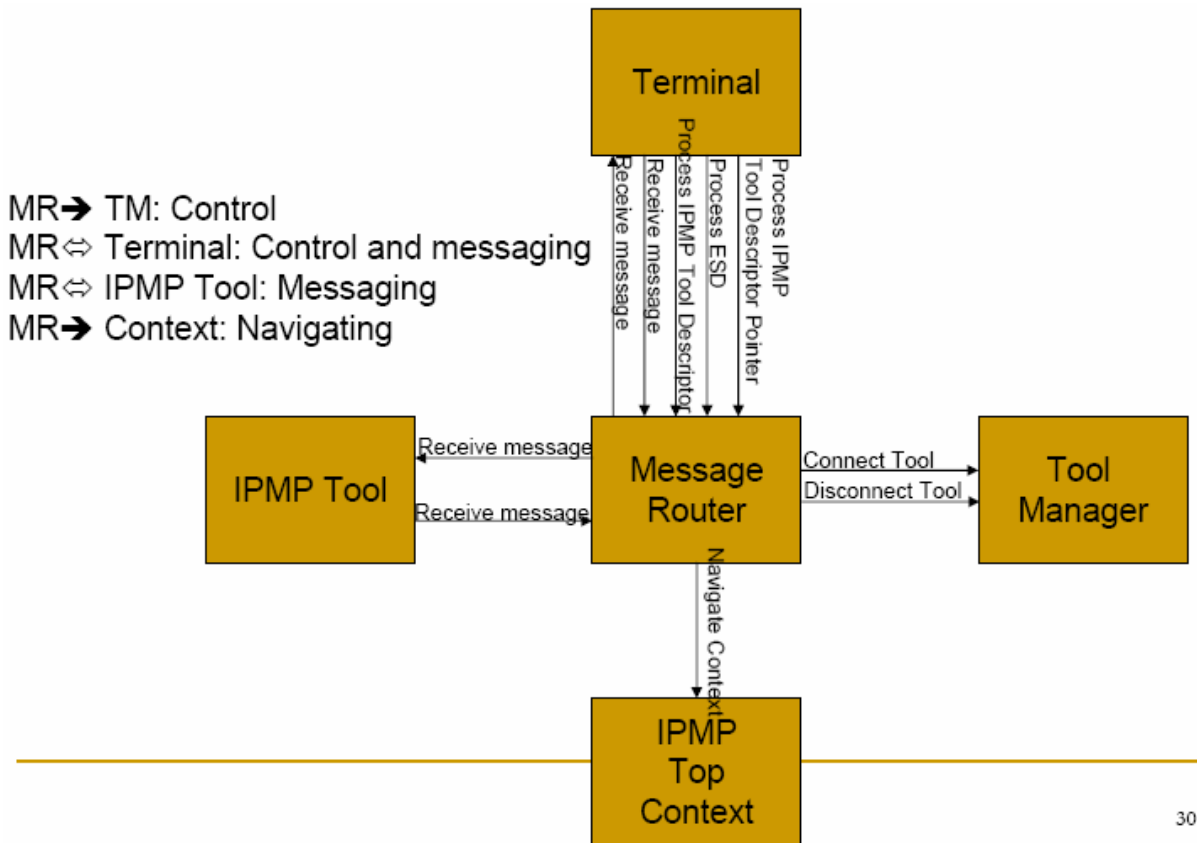
Nowadays, the design trend of the decoder is moving toward passive decoder in recent streaming system. And the concept of filter is fit in such kind of system. However, in MPEG-21 Testbed, the decoder is designed as an active decoder, so the concept of the filter is not proper within. Therefore, in our design, we have to adapt the design scheme of processors.

5.2 Design and Implementation of Software

In the architecture of MPEG-21 Testbed shown in Figure 5, only three IPMP modules are visible in the architecture. Two of them, PostDecoder Filter and PreDecoder Filter, are at the Client side and the other one, PostDIA Filter, at the Server side. However, there are other important modules that integral parts of IPMP system. They are Message Router, Tool Manager, IPMP Tools, and the Terminal. Here, we discuss the API design and the implementation issues of the IPMP system in MPEG-21 Testbed module by module.

5.2.1 Message Router

The relationship between the Message Router and other modules is shown in Figure 19.



30

Figure 19 Relationship between Message Router and other modules

The Message Router of IPMP system in the MPEG-21 Testbed has four main tasks. The Message Router here has to process the IPMP Tool Descriptor whenever the Terminal receives it. The IPMP Tool Descriptor may come from the IOD or from the bitstream. At the Server side, the IOD is stored in a file along with the media file. Before accessing the media file, the Server has to access the IOD file of the requested media file for setting up the streaming system and the IPMP system at the Server side. When the Terminal accesses the IOD, it parses the IOD and then passes IPMP Tool Descriptor, ES Descriptor, and IPMP Tool Descriptor Pointer to the Message Router for processing. The Message Router processes IPMP Tool Descriptor in order to get the IPMP information, some for initializing IPMP Tools, and routes it to the corresponding IPMP Tools. Moreover, after parsing the IPMP Tool Descriptor, the Message Router informs the Tool Manager to connect the IPMP Tools to specified location according to IPMP Tool Descriptor.

The Message Router also has to process the ESD coming from the Terminal in order to parse the IPMP Tool Descriptor Pointer within it. The IPMP Tool Descriptor Pointer included

in the ESD indicates the existence of an IPMP Tool with its description as the IPMP Tool Descriptor pointed by the IPMP Tool Descriptor Pointer in the ES. The location within ES is indicated by the control point code and sequence code within the IPMP Tool Descriptor.

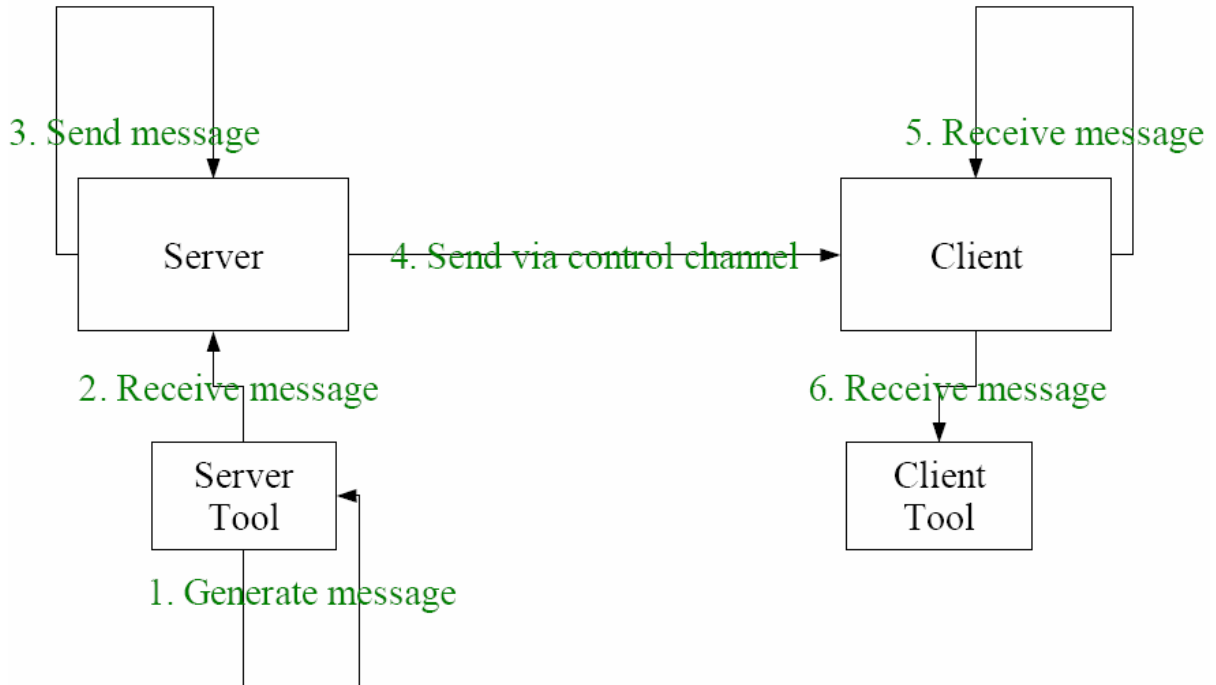


Figure 20 The messaging procedure



During the content consumption, there may be many run-time messages carrying the IPMP information generated by the IPMP Tools or coming from the bitstream. The Message Router has to receive these messages and routes them to corresponding IPMP Tools, Terminal, or other IPMP Devices. A possible scheme for the generating, routing, and receiving of the message is shown in Figure 20. In Figure 20, the Server means the server system of Testbed including the IPMP system at server side, and the Client means the client system including the IPMP system at the client side. In the client-server messaging scheme, first, one of the IPMP Tools at server side generates an IPMP message that has to be sent to one of the IPMP Tools at the client side. The message is then passed to the Message Router. The Message router receives the message from the IPMP Tool and determines that if the recipient of the message is at server side. If the recipient of this message is locally available, the Message Router routes the message to indicated IPMP Tool locally. However, if the recipient is at the client side, the Message Router has to route the message to the Terminal first, and then the message

is send to the client side through the control channel. The send message is first received by the controller at the client side. Here we can view the controller as one part of the Terminal. Then the controller passes the message to the Message Router for routing the message to the corresponding IPMP Tool. Finally, the Message Router calls the API named `ReceiveMessage()` to pass the message to the IPMP Tool. However, the process of the message within the IPMP Tool is an implementation issue depending on the design of the IPMP Tool.

The APIs associated with the Message Router are listed bellow.

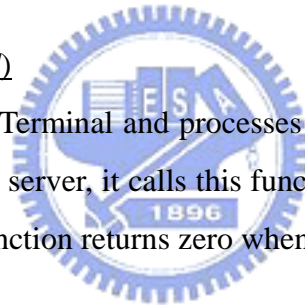
Method:

`int ProcessIPMPToolDescriptor(IPMPToolDescriptorD* IPMPtooldescriptor)`

This API is called by the Terminal to process the IPMP Tool Descriptor. When the Terminal receives the IOD from the server, it calls this function to pass *IPMPtooldescriptor* contained in IOD to MessageRouter. The function returns zero when succeed.

`int ProcessESD(ESD* esd)`

This API is called by the Terminal and processes the ES Descriptor. When the Terminal receives the IOD from the server, it calls this function to pass all ESDs contained in IOD to MessageRouter. The function returns zero when succeed.



`int ProcessIPMPDescriptorPoniterD (uint32 contextID, IPMPDescriptorPointerD* ptr)`

This API is called when processing an ES Descriptor. MessageRouter instructs ToolManager to instantiate an IPMPTool at the given control point of the given ES. The *contextID* specifies the context where the pointer resides. The *ptr* contains the information of which IPMPToolDescriptor is to be used. The function returns zero when succeed.

`int ReceiveMessage(IPMPToolMessageBase* msg)`

This API is called by an IPMPTool or the Terminal. On receiving a message (the parameter *msg*), MessageRouter determines and sends it to the corresponding recipient. The function returns zero when succeed.

`IPMPContext* GetContext()`

This API provides means to access the corresponding context. All IPMP objects can refer to each other by navigating through the context tree.

5.2.2 Tool Manager

The relationship between the Tool Manager and other modules is shown in Figure 21.

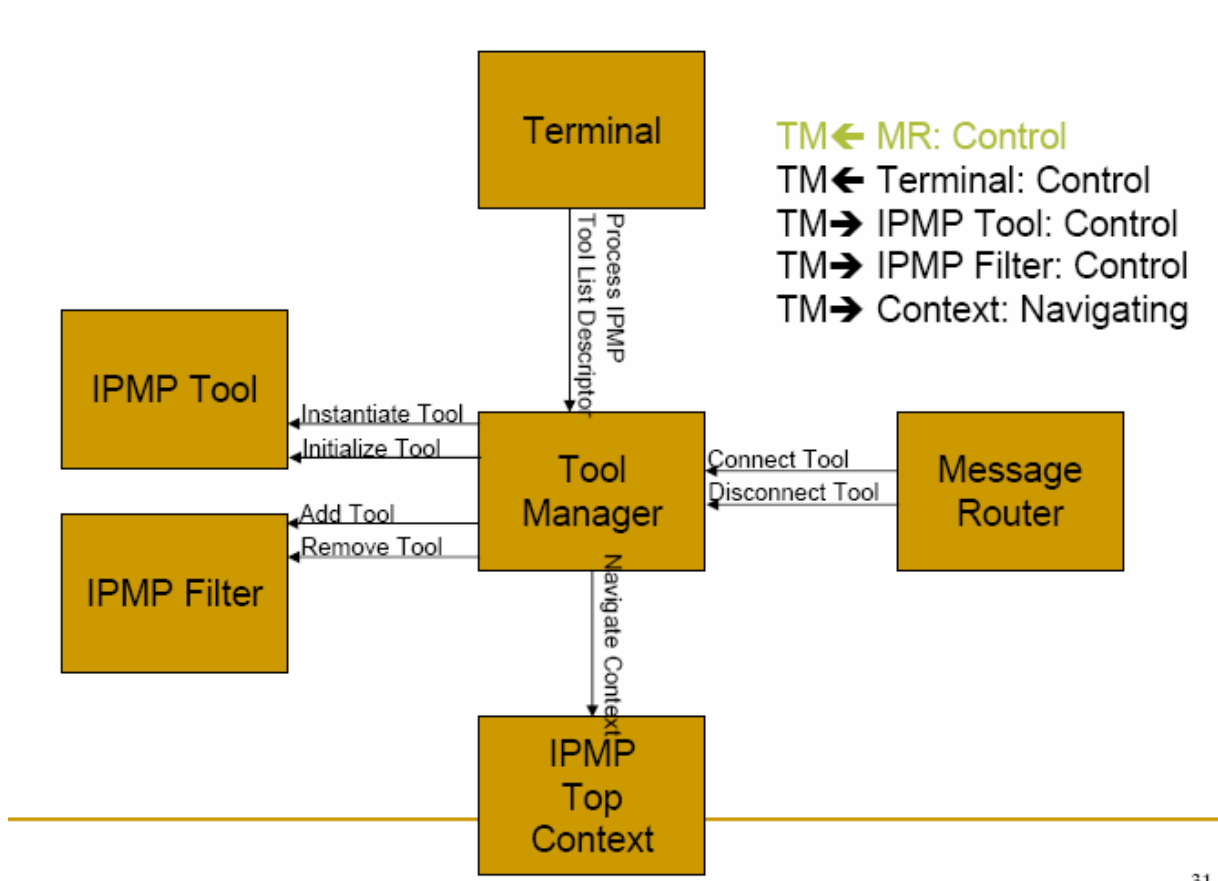


Figure 21 Relationship between Tool Manager and other modules

In the MPEG-21 Testbed, the Tool Manager mainly takes charge of three jobs. It receives the IPMP Tool List Descriptor passed from the Terminal. The IPMP Tool List Descriptor is contained in the IOD and is passed to the Tool Manager when the Terminal accesses the IOD. Receiving the IPMP Tool List Descriptor, the Tool Manager has to resolve the IPMP Tools listed within. As described in chapter 2, the IPMP Tool List Descriptor supports three modes for describing the IPMP Tools, unique, alternates, and parametric. In the current version, we use the unique mode to indicate the required IPMP Tools. However, we also implement the description of alternates and parametric mode. Hence, users interested in these two modes

could use these two modes directly.

Besides processing the IPMP Tool List Descriptor, the Tool Manager is also responsible for adding (or removing) the tool to (or from) the specified IPMP Filter. When the Tool Manager is asked to add one IPMP Tool in an IPMP Filter, it should be fed with the IPMP Tool Descriptor belonging to this IPMP Tool. However, the one who holds the information of IPMP Tool Descriptor is the Message Router. So, it is reasonable that the Message Router asks the Tool Manager to connect the IPMP Tool to the specific IPMP Filter. It is the same from removing IPMP Tool from an IPMP Filter.

The APIs associated with the Tool Manager are listed below.

Method:

int ReceiveToolListDescriptor(ToolListDescriptorD* *tool_list*)

This API is called by the Terminal and processes the IPMP tool list. When Terminal receives the Tool List Descriptor in the IOD, the function is called to pass *tool_list* to ToolManager. Then, the ToolManager resolves the tool IDs for later instantiation. Note that the server-side and client-side resolving process may be quite different. The function returns zero when succeed.

IPMPTool* ConnectTool(uint32 *context_id*, IPMPToolDescriptorD* *descr*)

Instantiate and connect a tool. When this method is invoked, the tool is instantiated by the parameters specified in *descr*. Then, the ToolManager connects it to the specified location (*context_id* specifies the ES to apply, and the control point is specified in *descr*).

int DisconnectTool(IPMPTool* *tool_ptr*)

Disconnect the given tool pointed by *tool_ptr* from the IPMPFilter. The function returns zero when succeed.

IPMPContext* GetContext()

This API provides means to access the corresponding context. All IPMP objects can refer to each other by navigating through the context tree.

5.2.3 Terminal

The relationship between the Terminal and other modules is shown in Figure 22.

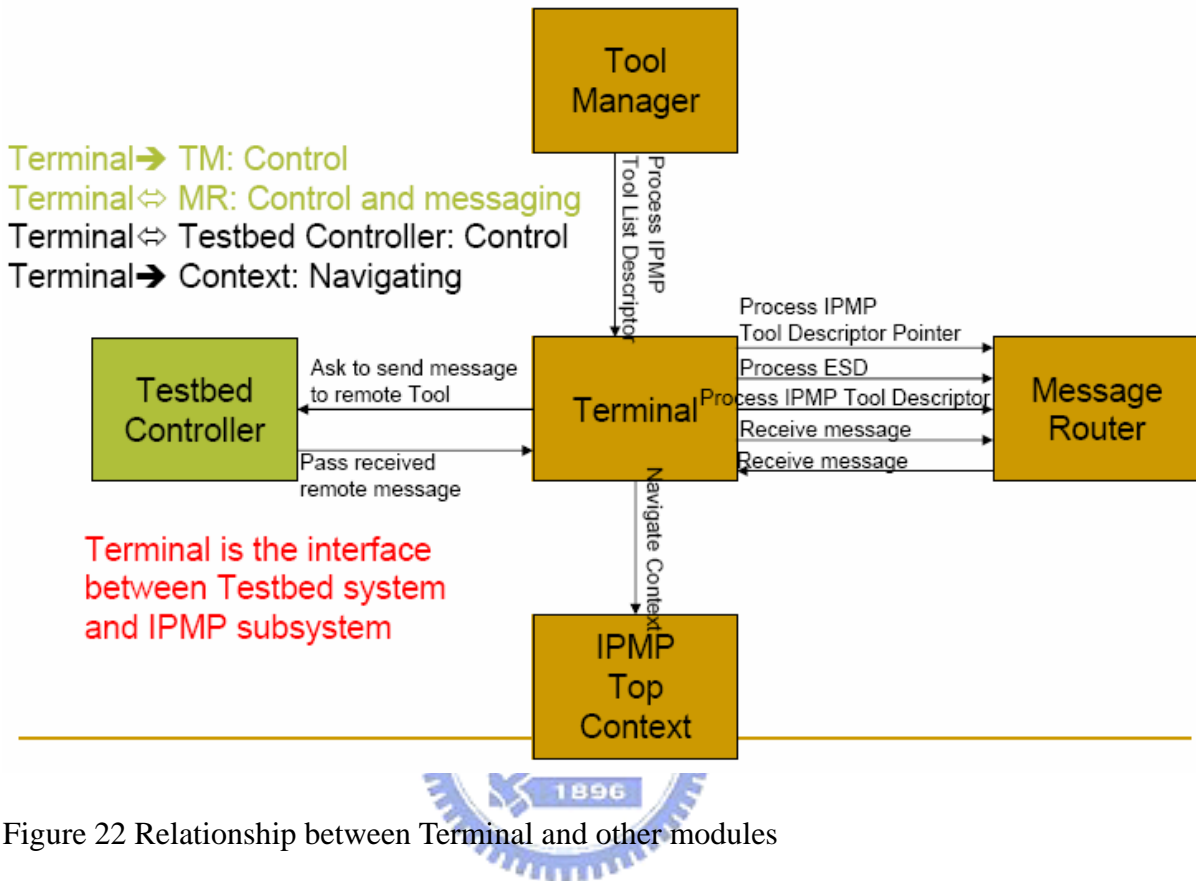


Figure 22 Relationship between Terminal and other modules

The Terminal class here can be viewed as the abstract of the one whole side Testbed system except for the IPMP subsystem. It serves as the interacting interface between the Testbed system and the IPMP subsystem. The interactions between the Testbed system and the IPMP subsystem include local messaging and remote messaging. When there are any messages with the recipient as the Terminal, the Message Router routes the messages to the Terminal by calling the API, ReceiveMessage(), of Terminal. And the processes of receiving messages by Terminal depend on the types of the messages.

As for the remote messaging, we design another API named SendMessageToBitstream() to support this type of messages. In the Testbed system, we send remote IPMP messages through the control channel. The message types, local or remote, are determined by the Message Router when the Message Router receives the message from IPMP Tools. Then, if the message is local message, the Message Router routes the message to the assigned IPMP

Tool locally; otherwise, the Message Router calls the API, `SendMessageToBitstream()`, to pass the message to Terminal for remote messaging.

The APIs associated the Terminal are listed bellow.

Method:

`int ReceiveMessage(IPMPToolMessageBase* msg)`

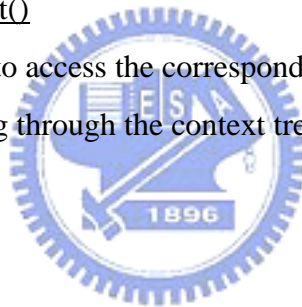
This method is called by the MessageRouter to pass terminal specific IPMP messages. The *msg* object is the message to be processed. The method returns zero when succeed.

`int SendMessageToBitstream(IPMPToolMessageBase* msg)`

This method is called by the MessageRouter to pass remote-terminal specific IPMP messages. The *msg* object is the message to be sent to the other side. The method returns zero when succeed.

`IPMPContext* GetContext()`

This API provides means to access the corresponding context. All IPMP objects can refer to each other by navigating through the context tree.



5.2.4 IPMP Filter

The relationship between IPMP Filter and other modules is shown in Figure 23.

The IPMP Filters serves as the IPMP Control Points in the IPMP system within the MPEG-21 Testbed. It supports that the user could add or remove the IPMP Tool in the IPMP Filter. There are three IPMP Filters in MPEG-21 Testbed system, one at the server side, and the other two at the client side. At the server side, in order to perform the encryption on the media content bitstream, we allocate one IPMP Filter at the location between two modules, DIA and Streamer. It is called the PostDIAFilter in the Testbed system. However, the PostDIAFilter here allows user to add IPMP Tools, whose functions are other than encryption. What kind of the IPMP Tool should be used is an application issue.

Since the encryption is performed at the server side, there must be one corresponding IPMP Filter to perform the functionality of decryption at client side. So, we locate an IPMP Filter between the Decoder and the Stream Buffer. It is called PreDecoderFilter. Because the output data of the module, DIA, is in the bitstream format, and we do the encryption at the

bitstream level, therefore, the PreDecoderFilter is located here to decrypt the data at the same level.

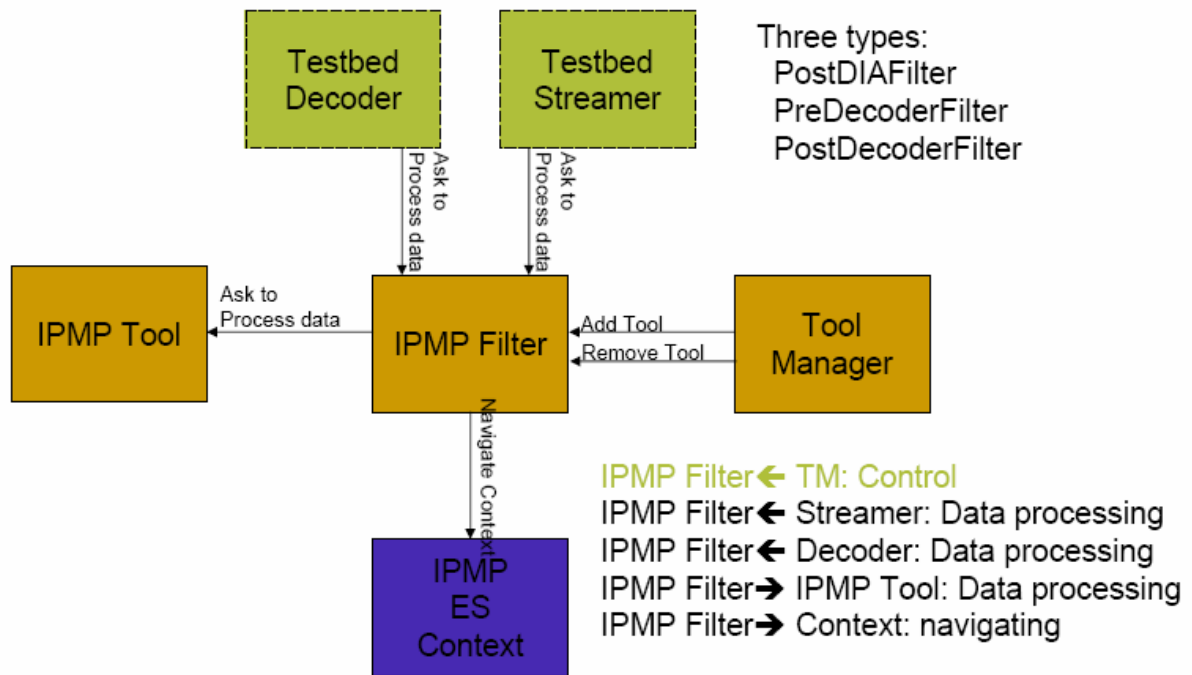


Figure 23 Relationship between IPMP Filter and other modules

There is another IPMP Filter, PostDecoderFilter, located between the Decoder and the Output Buffer. With this IPMP Filter, the functionality of extract watermarking from a decoded bitstream data can be implemented. Note that there is no corresponding IPMP Filter of the PostDecoderFilter at the server side because we assume that the insertion of the watermarking is done offline. There could be other possible use of the IPMP Filters in different combination, and it is an application issue. Here we will show one possible application of them.

Because the data before being processed by the Filter and the data after being processed may have different size, we design an API, ProcessData(), which has five parameters. The caller of the API specifies the size of the input data for processing and the data buffer. After processing the data the caller could get the processed data from the *out_data_ptr* with its size specified in *out_size_ptr*. For example, there is one DES decryption tool in the PreDecoderFilter, and the DES tool decrypts the data block by block. However, the size of the bitstream data may not be multiples of blocks, so there may exist blocks with zero padding.

When the DES tool decrypts a block with zero padding, it returns the data with the size differ from the original block size, which is the input data size, ProcessData(). In order to solve this problem, we need an API, ProcessData(), as described bellow.

The APIs associated with the IPMP Filter are listed bellow.

Method:

int AddTool(IPMPTool* tool, uint8 sequence_code)

Add the given IPMPTool to the filter. The *tool* object is an initialized IPMPTool, and the *sequence_code* specifies the priority of the tool. A higher value of *sequence_code* denotes higher priority when processing data. This method returns zero when succeed.

int RemoveTool(IPMPTool* tool)

Remove the specified IPMPTool from the filter. The parameter *tool* specifies the IPMPTool to be removed. This method returns zero when succeed.

int ProcessData(uint32 timestamp, uint8* in_data, uint32 in_size, uint8** out_data_ptr, uint32* out_size_ptr)

This method iterates through each contained IPMPTool and chained all data processing operations to produce the final result. The *in_data* is the input data buffer of length *in_size*. The created output buffer is returned in *out_data_ptr* with length *out_size_ptr*. This method returns zero when succeed.

In the current implementation of IPMP system embedded in the MPEG-21 Testbed system, since the IPMP Filters are connected to the Streamer at the server side and to the Decoder at the client side, the caller of the API, ProcessData(), should be the Streamer and the Decoder. At the client side, the Decoder acquires the bitstream data from the Stream Buffer by calling the API, GetBitstreamData(). After receiving the bitstream data from the Stream Buffer, Decoder sends data to PreDecoderFilter before decoding the data. The data decoding process should take place after the data are returned by the PreDecoderFilter. Before the decoded data being sent to the Output Buffer, the Decoder sends them to the PostDecoderFilter first. After getting the returned processed data, Decoder sends them to the Output Buffer. The scheme of processing the bitstream data at client side is shown in Figure 24.

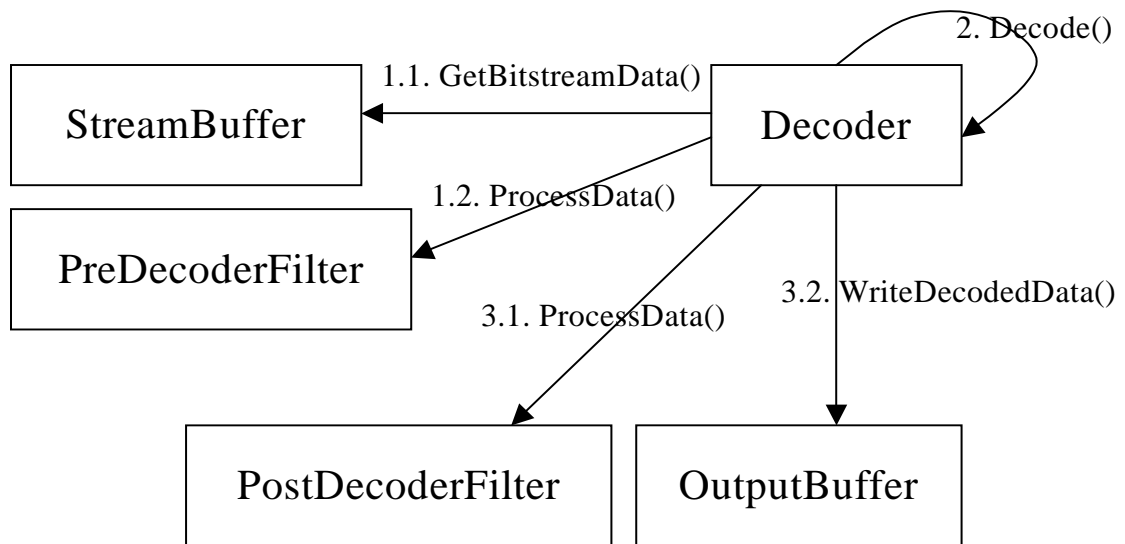


Figure 24 The procedure of processing data by IPMP Filter at client side

At server side, the Streamer gets the bitstream data from DIA by calling the API of DIA named `GetNextResourceUnit()`, and then sends the data to `PostDIAFilter` for processing right away. After getting the data returned by `PostDIAFilter`, the Streamer begins to process the bitstream data. The detail scheme is shown in Figure 25.

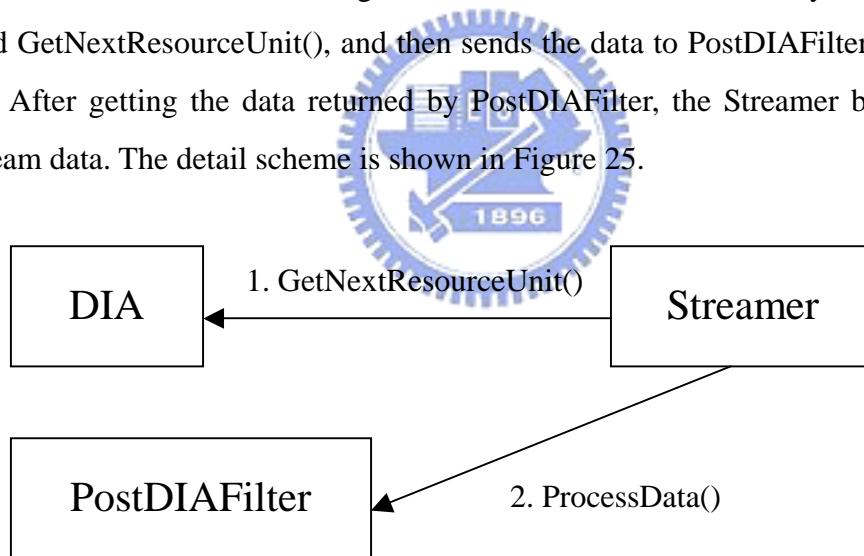


Figure 25 The procedure of processing data by IPMP Filter at server side

There may be multiple IPMP Tools within one IPMP Filter, so we design an `IPMPToolChain` to record the IPMP Tools within the IPMP Filter. As described before, the order of processing data is specified by the sequence code, and the IPMP Tool with higher sequence code processes the data first. The procedure of processing the bitstream data could be written as the pseudo codes bellow.

```

for (sequence_code=255; sequence_code >=0; sequence_code --) {
    IPMPTool* tool = GetIPMPToolChain()[sequence_code];
    if (tool != 0) {
        if (first_flag != 1) {
            clear_the_buffer();
            set_in_out_buffer();
        } else {
            first_flag = 0;
        }
        tool->ProcessData(timestamp, cur_in_data, cur_in_length, &cur_out_data,
            &cur_out_length);
    }
}

```

5.2.5 IPMP Tool

The relationship between IPMP Tool and other modules is shown in Figure 26.

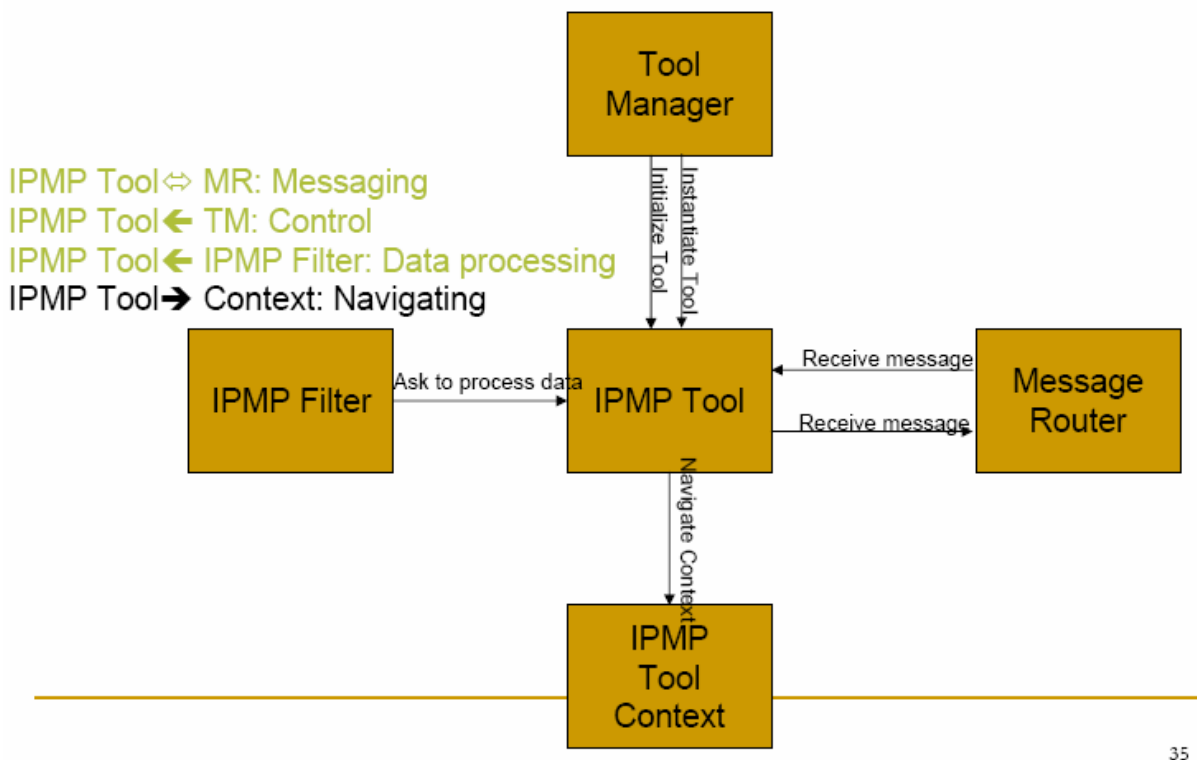


Figure 26 Relationship between IPMP Tool and other modules

In order to make the design and implementation of the IPMP Tools more flexible, here we design the IPMP Tool with the highest principle of simplicity. This class is designed as the basis interface of an IPMP Tool. An IPMP Tool, for instance, the DES decryption tool, may process the bitstream data in an ES. Thus, the ProcessData() API is a part of the basic

interface of an IPMP Tool. And, the concept of designing this function is the same with the one of the IPMP Filter.

An IPMP Tool should also have the capability to receive the message routed by the Message Router. However, the procedure of processing the received message is the implementation issue and hence we ignore it here.

The APIs associated the IPMP Tools are listed bellow.

Method:

int ReceiveMessage(IPMPToolMessageBase* msg)

This API is called by the MessageRouter to pass a message to the IPMPTool object. The parameter *msg* is the IPMP message to be handled by this tool. The function returns zero when succeed.

int ProcessData(uint32 timestamp, uint8* in_data, uint32 in_size, uint8** out_data, uint32* out_size)

This API is called by the IPMPFilter to process the given data. The parameter *in_data*, is the input buffer of length *in_size*. The indirect pointer ***out_data* returns a pointer to the processed data, and the size returned as *out_size*. The output buffer should be created during the invocation of the method, and should be released somewhere else. The IPMPFilter should take care of the release operations, except the last output buffer of the IPMP tool chain. The function returns zero when succeed.

void Initialize(IPMPToolDescriptorD* init)

Initialize the IPMPTool by the given *init* object. This method is called by the ToolManager after this tool object is instantiated.

IPMPContext* GetContext()

This API provides means to access the corresponding context. All IPMP objects can refer to each other by navigating through the context tree.

In the current implementation, we implement a pair of DES tool, one for encryption, and the other for decryption. The basic feature of DES algorithm is that it is the block cipher algorithm. In our implementation, we use the library called crypto++, a C++ class library of

cryptographic primitives. It provides many implementations of cipher algorithm such as AES (Rijndael), DES, and many other different algorithms in cipher. And the library provides both stream cipher and block cipher. However, in our implementation, because we focus on the IPMPX system design, not the design of encryption algorithm. We choose the one for easiest implementation, the DES block cipher.

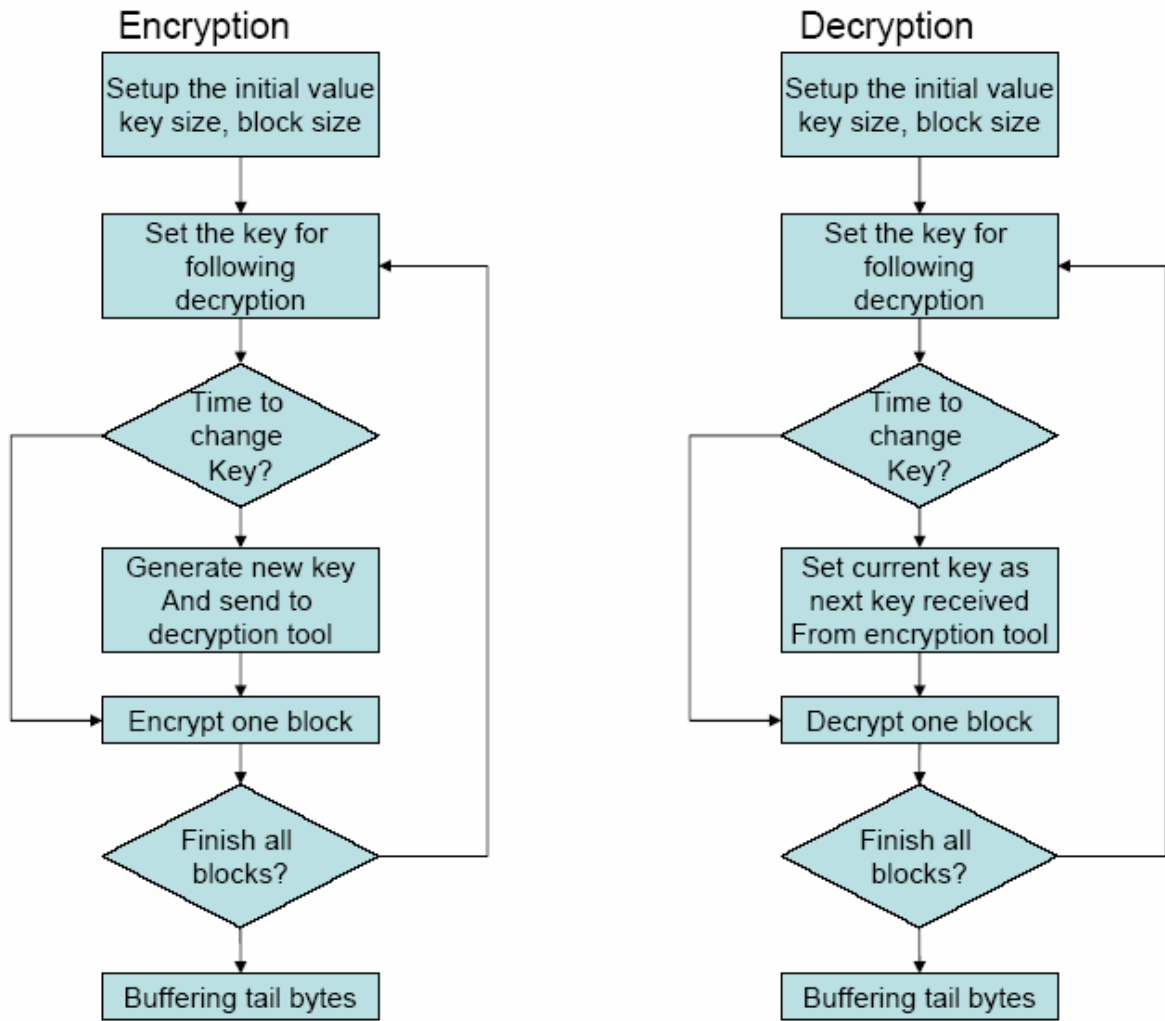


Figure 27 Flow chart of IPMP DES Tool

The DES cipher algorithm has been implemented in the library, `crypto++`. Its default block size is 8 bytes, and the default key is also 8 bytes, too. Because these IPMP Tools are associated with DES block cipher algorithm, when the data are processed, the data have the size of multiples of block. Therefore, we design a simple buffer in the IPMP Tools to buffer the tailing data whenever the accumulated data have the size as multiples of a block. In order

to perform the real-time encryption at server side, we design the DES encryption tool with the ability to generate the new key and send the new key to the decryption tool.

The flowchart of the current implementation of the IPMP Tools with the DES block cipher algorithm is shown in Figure 27.



Chapter 6

Demo

In this chapter, we design an application example of MPEG-4 IPMPX system and implement it. Here we only describe the client side of the MPEG-21 Testbed as our testing environment and omit the server side and of network. Oenerally, to test the functionalities of IPMPX system, it is sufficient to look at the client side only.

6.1 Structure

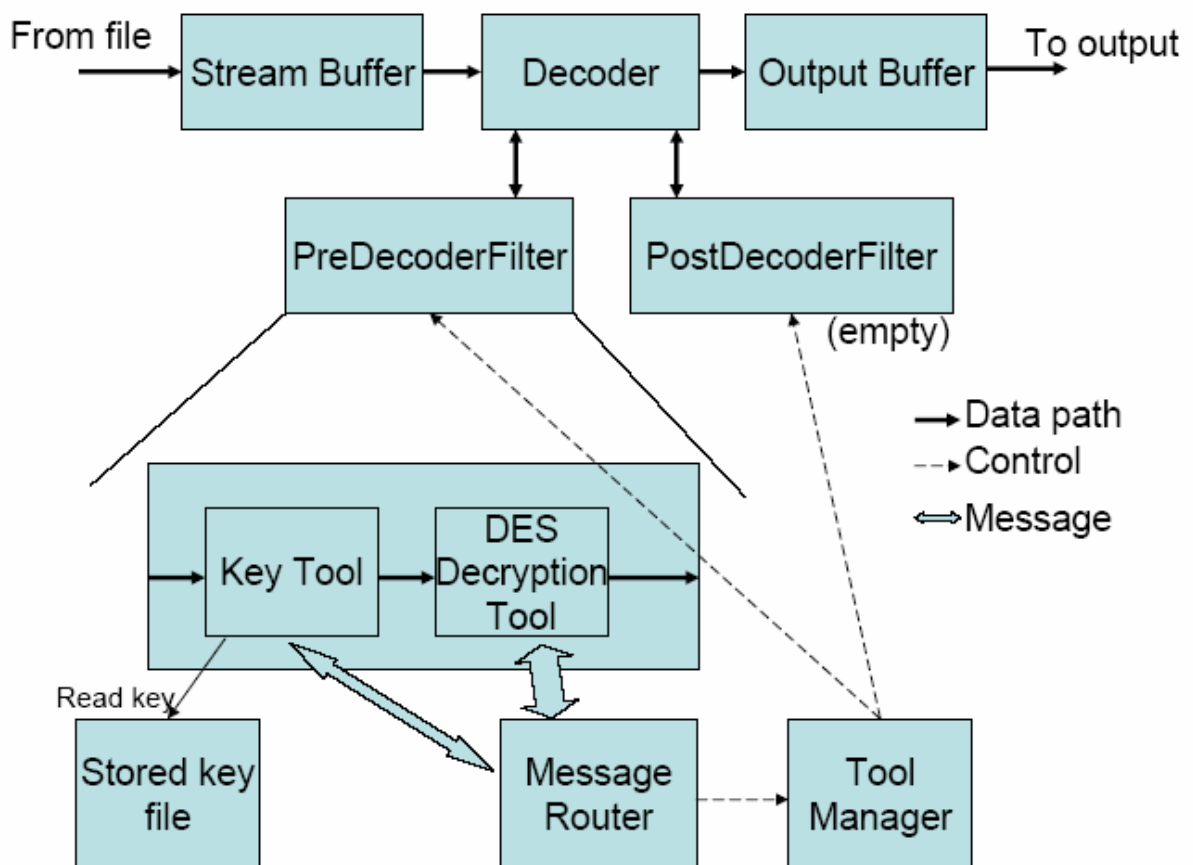


Figure 28 System diagram of our application

In this demonstration, we build up the system as shown in Figure 28. This demonstration

is designed to test the functionality of IPMPX system for Conditional Access (CA).

The Decoder here is an active device. It acquires the bitstream data from the Stream Buffer and writes the decoded data to the Output Buffer. Before decoding data, the Decoder asks the PreDecoderFilter to process these data. After writing data to the Output Buffer, It first as the PostDecoderFilter to process these decoded data. In this application, there are two threads, the decoder thread and the thread for displaying the frame.

There are two IPMP Tools in the PreDecoderFilter, Key Tool and DES Decryption Tool. The Key Tool is to read the key and send the key to DES Decryption Tool through IPMP messages. The DES Decryption Tool is to decrypt the data. It is empty in the PostDecoderFilter hence data processed by the PostDecoderFilter are unchanged.

6.2 Setup

We first offline encrypt a bitstream and store the bitstream in a media file. During the encryption of the bitstream, the encryption procedure changes the encryption key periodically, that is, one new key for a fix number of blocks. The key is generated by a random function. The key list is stored in a key file for decryption. These jobs are done prior to testing the IPMPX system as shown in Figure 28.

Except for the key file and the media file, there is still one important piece of data that has to be prepared, the Initial Object Descriptor file. There are three main components in the Initial Object Descriptor: IPMP Tool List, IPMP Tool Descriptor, and the IPMP Tool Descriptor in the ESD. We must set up these three pieces of information. The attributes and parameters of these three components are shown bellow.

IPMP_ToolListDescriptor

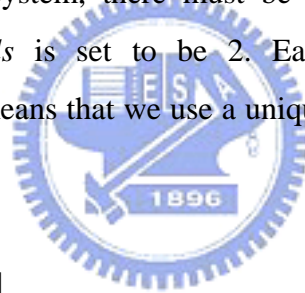
```
{
    numTools = 2;
    IPMPToolID[0]
    {
        IPMP_ToolID = [140, 113, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1];
        isAltGroup = 0;
        isParametric = 0;
    }
}
```

```

numURLs = 0;
ToolURL = NULL;
}
IPMPToolID[1]
{
    IPMP_ToolID = [140, 113, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2];
    isAltGroup = 0;
    isParametric = 0;
    numURLs = 0;
    ToolURL = NULL;
}
}

```

There is one IPMP Tool List in the Initial Object Descriptor. Because there are two IPMP Tools in this demonstrating system, there must be two IPMPToolIDs on the IPMP Tool Descriptor List and *numTools* is set to be 2. Each IPMPToolID has its own unique *IPMP_ToolID* within, which means that we use a unique mode to describe the required IPMP Tools for content consumption.



IPMP_ToolDescriptor[0]

```

{
    IPMP_ToolDescriptorID = 1;
    IPMP_ToolID = [140, 113, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1];
    URLString = NULL;
    isInitialize = 1;
    IPMP_Initialize
    {
        controlPointCode = 0x01;
        sequenceCode = 220;
        numOfData = 1;
        IPMPX_data[0] : IPMP_OpaqueData
        {
            opaquedata = {"duration to change the key", "Initial Key", "corr_tool"}
        }
    }
}

```

```

    };
}
numOfData = 0;
IPMPX_data = NULL;
}

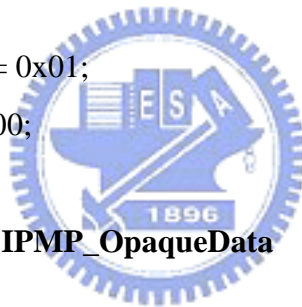
```

IPMP_ToolDescriptor[1]

```

{
    IPMP_ToolDescriptorID = 2;
    IPMP_ToolID = [140, 113, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2];
    URLString = NULL;
    isInitialize = 1;
IPMP_Initialize
    {
        controlPointCode = 0x01;
        sequenceCode = 200;
        numOfData = 1;
IPMPX_data[0] : IPMP_OpaqueData
        {
            opaquedata = {"duration to change the key", "Initial Key", "corr_tool"}
        };
    }
    numOfData = 0;
IPMPX_data = NULL;
}

```



Because there are two IPMP Tools instantiated in this demonstrating system, we need two IPMP Tool Descriptors for them. Although there may be multiple IPMP Tools sharing the same IPMP Tool Descriptor, the IPMP Tools appear in the demonstrating system are quite different. The *IPMP_ToolDescriptorID* could be set at the users' will, so we assign the value to them as we wish. In both IPMP Tool Descriptors, the *isInitialize* is set to 1 to indicate that the IPMP Tools are newly instantiated. The *controlPointCode* here are set to 0x01, which means that these two IPMP Tools are both connected to the PreDecoderFilter. And the

sequenceCode indicates the order for processing the data. Finally, the IPMPX_data carried in IPMP_Initialize is set as IPMP_OpaqueData to carry initialization information for IPMP Tools such as the duration for changing keys.

```

ES_Descriptor[0]
{
    ES_ID = 0;
    IPMP_ToolDescriptorPointer[0]
    {
        IPMP_ToolDescriptorID = 1;
    }
    IPMP_ToolDescriptorPointer[1]
    {
        IPMP_ToolDescriptorID = 2;
    }
}

```

IPMP Tool Descriptor Pointers within the ES Descriptor indicate that the stream described by this ESD is protected by the IPMP Tools indicated in the IPMP Tool Descriptors pointed by them. The relationship among these pointers is shown in Figure 29.

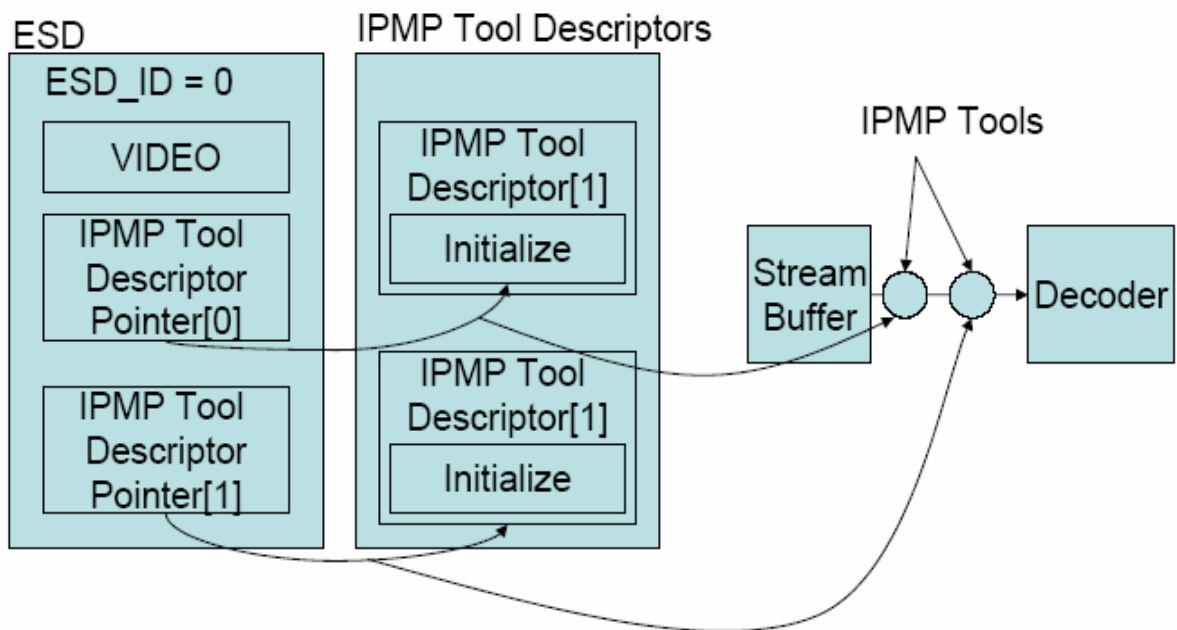


Figure 29 Relationship of various pointers in the demonstration system.

6.3 Execution

First, the Terminal accesses the Initial Object Descriptor in the IOD file, and resolves it. Then it passes the IPMP Tool List Descriptor to the Tool Manager and passes the IPMP Tool Descriptor and ES Descriptor to the Message Router. Tool Manager parses the IPMP Tool List Descriptor to resolve the Tool IDs of the required IPMP Tools for content consumption and then build up the IPMP Tool map. Message Router parses the IPMP Tool Descriptor to get the information for initialing of IPMP Tools. Besides, Message Router parses the IPMP Tool Descriptor Pointer in ESD to connect the required IPMP Tools to the locations specified in the IPMP Tool Descriptor. During the procedure of setting up the IPMP system, the context tree is built for navigation in the following steps.

The behavior of these modules, Message Router, Terminal, and IPMP Tools, in the IPMP system after setting up the IPMP system is shown in Figure 30.

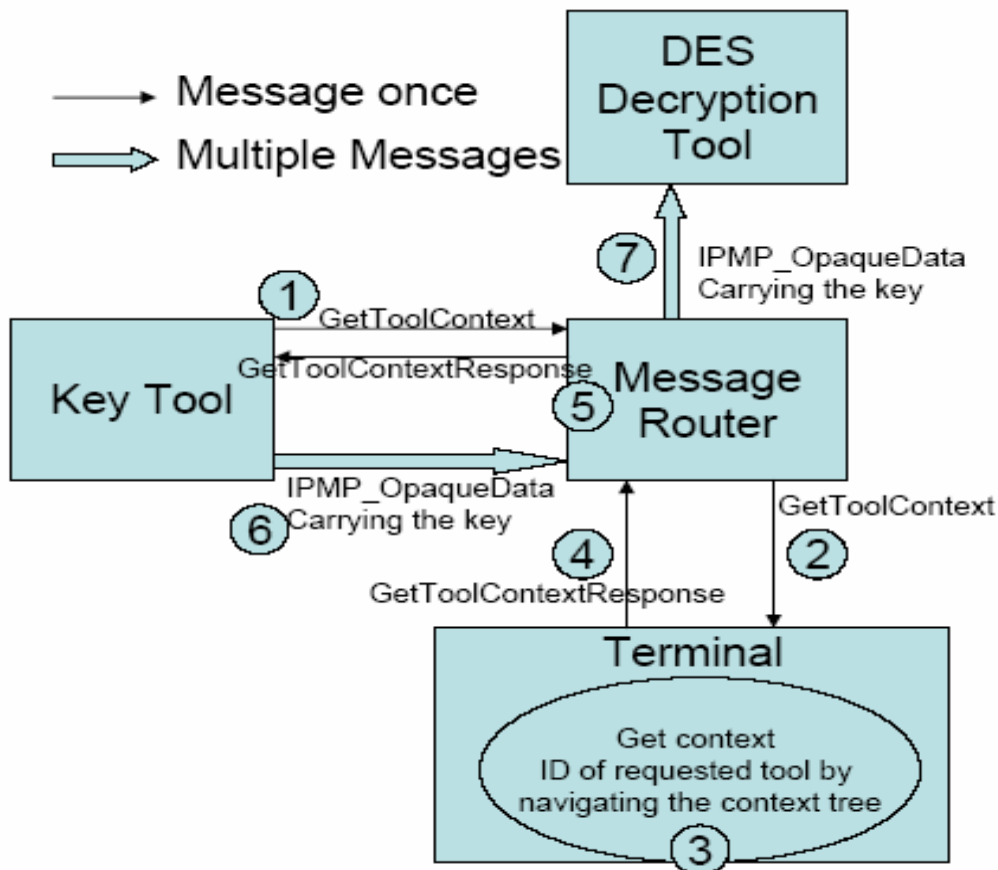


Figure 30 Messaging routine in the demonstration system

In order to enable passing messages between the key tool and the DES description tool,

the key tool generates an IPMP message with type GetToolContext and sends it to the Terminal for querying the IPMP Tool with the IPMP Tool Descriptor specified in the message. Then, the Terminal processes the message and retrieves the context tree for the context ID of requested IPMP Tool. If found, the Terminal will generate the IPMP message with its type as GetToolContextResponse and send it back to the IPMP Tool, which requests the context ID. After getting the context ID of the DES decryption tool, the key tool informs the DES decryption tool to change the key through the IPMP messages. The key tool inserts the key data into the IPMP opaque data as a message and sends it to the DES decryption tool. If the key is incorrect, the user can only view the video at very low quality or nothing at all. However, if the negotiation of the IPMP Tools is not successful, the key tool will hold the bitstream such that the user sees nothing in the display window.

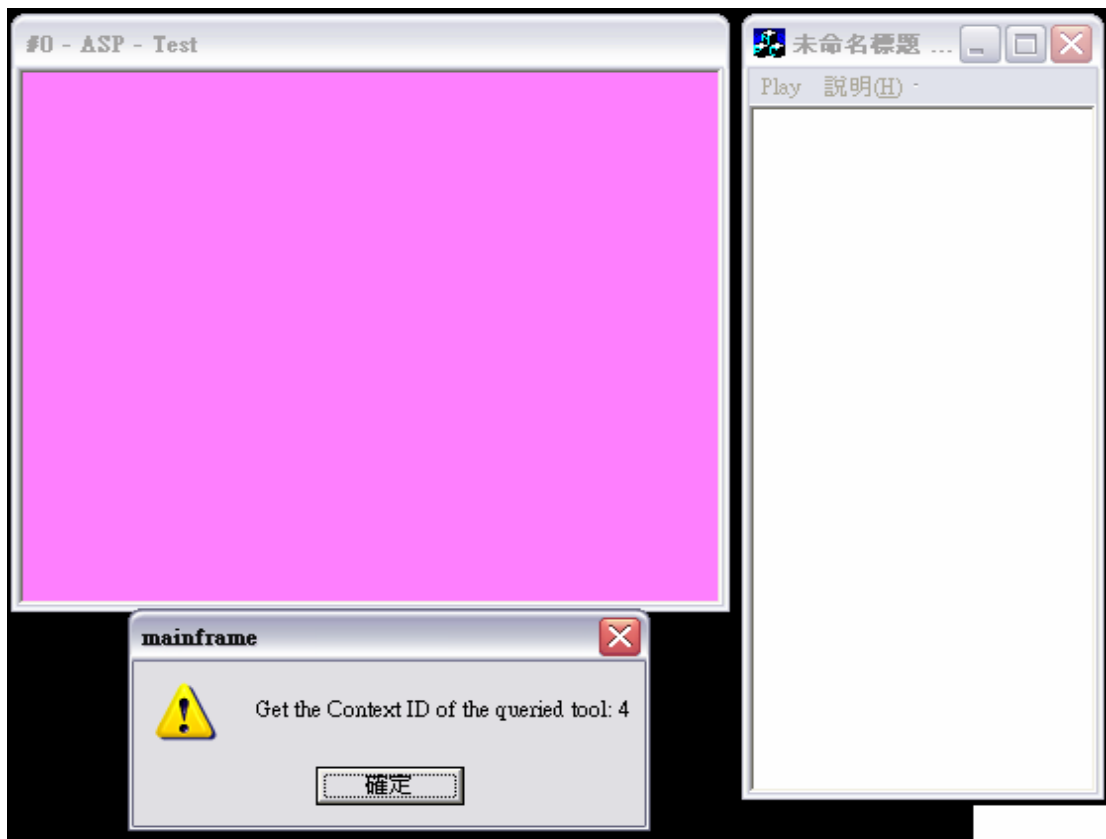


Figure 31 Screenshot_1 of demonstration system

As shown in Figure 31, the window with title “- ASP - Test” is the one for displaying. And the message box shows the current status of processing. The main window of the entire system is the one at right side. Entire process begins when user clicks the “Play” button.

Figure 31 is the screenshot before displaying the media data. Hence, there is nothing in the displaying window.

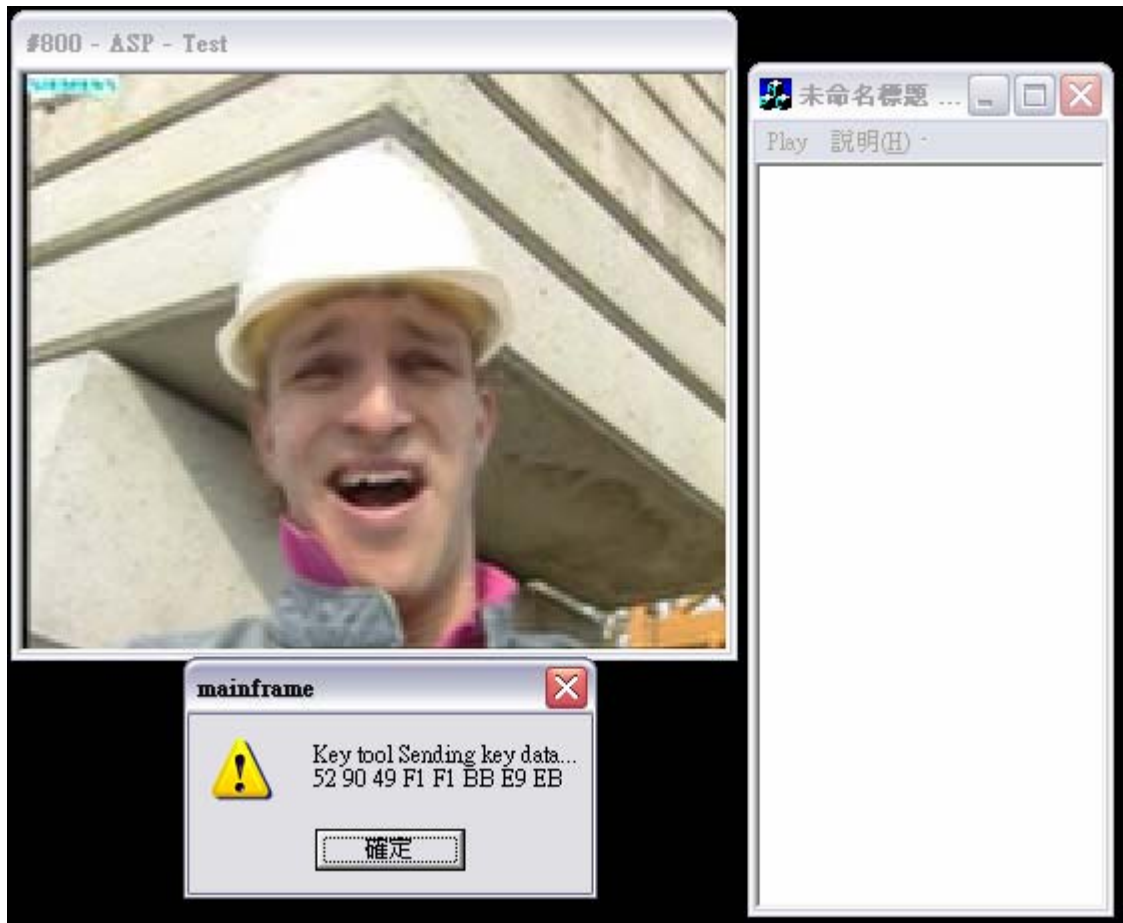


Figure 32 Screenshot_2 of demonstration system

The screenshot during displaying is shown in Figure 32. And the processing steps are also shown in the message box. The message in the current message box indicates that the Key Tool sends the key data to the DES Decryption Tool. And the content of the key is “52 90 49 F1 F1 BB E9 EB”.

When the DES Decryption Tool decrypts data with incorrect key, the decrypted data are viewed as lost packets by the Decoder. Hence the Decoder does not decode these data and holds the process until the key is correct. When the key data is correct again, the Decoder decodes these data again. Because of the lost data, the frames shown in the window are fractured. We show this condition in Figure 33



Figure 33 Screenshot_3 of demonstration system

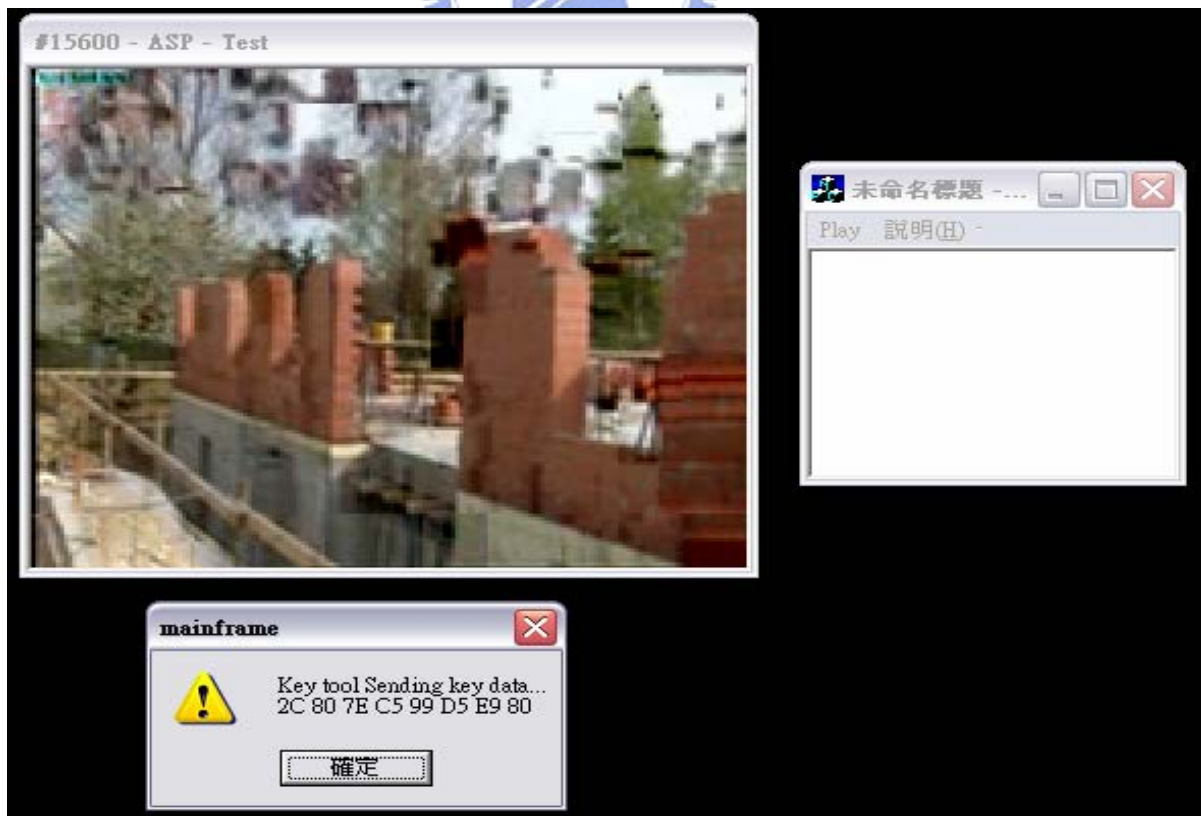


Figure 34 Screenshot_4 of demonstration system

If the key is continuously correct for a certain time, the frames will be reconstructed gradually as show in Figure 34. However there is only one “I frame” within entire sequence, hence the reconstructions of the frames are slow. That is, we can not get the perfect frame immediately when we get the correct keys.



Chapter 7

Conclusion

Because the issues of Digital Rights Management become more and more important, there are many companies that begin to develop their own Digital Rights Management systems, such as the Microsoft Media Digital Rights Management systems. However, the DRM systems come from different companies can not provide interoperable services among them. Therefore, it is a trend to standardize the Digital Rights Management interfaces and systems. The IPMPX system is an example of standardized Digital Rights Management systems with powerful functionalities.

The advantages of using the IPMPX system for industry and the end users are as follows.

Many DRM solutions use similar or same tools for a number of functionalities. For example, many different DRM systems use the same encryption algorithm. However, because there are no specifications on the interface of these tools, they can not be reused in another DRM system. Using the standardized IPMPX system could reduce redundant implementation and thus can facilitate the design and distribution of tools.

The MPEG IPMPX system provides a common ground for mutual authentication of IPMP Tools and Terminals such that the IPMP Tools and the Terminals can communicate with each other through the secure authenticated channel. The mutual authentication can also verify the trusty relationships between IPMP Tools and the Terminals.

In the MPEG IPMPX system, the IPMP messages are clearly defined and so are the interfaces. Thus, the points of non-interoperability are clearly specified allowing industry to further minimize incomparable issues.

From the application view point, the renewability of a DRM system must be provided. The IPMPX system could verify the validity of certificates and credentials by using the mutual authentication. Additionally, there are many updating messages routed dynamically to update IPMP Tools or to insert new IPMP Tools.

In an IPMPX system, one could choose whichever IPMP Tool to perform the functionalities such as decryption, watermarking, user authentication or data integrity checking. These functions provide through and yet flexible protection on the media content.

IPMPX is a system based on the message exchange concept. Hence there are a number of messages dynamically generated and routed for the communication between the IPMP Tools and the Terminals. The IPMP messages carry the IPMP information for updating the IPMP Tools or resulting in a new instance of the IPMP Tool to perform a new functionality.

IPMPX system is a practical DRM system and is a powerful one. However, the reference software of the IPMPX system included in IM1 is strongly confined by the MPEG-4 system software. It is very difficult to separate the IPMPX system from IM1 and port it to the MPEG-21 Testbed. In order to test the functionalities of IPMPX system in a streaming environment, we design and implement the IPMPX systems on the MPEG-21 Testbed.

In this thesis, we design a new and standalone IPMPX system, separated from the other parts of the MPEG-4 system. Then we implement this system on the MPEG-21 Testbed. Furthermore, we give several application examples in using the IPMPX system with MPEG-21 Testbed to show its powerful functionalities. The users of MPEG-21 Testbed could use the IPMPX system at their choice. And since we provide the IPMPX system implementation on the MPEG-21 Testbed, the manufactures of tools could insert the tools of their own design into the IPMPX system easily and test and verify the functionalities of the new tools.

Until now, the IPMPX system is built mostly on the client side of the MPEG-21 Testbed and is tested for its functionalities. However, there are several items remained to be done in order to make the whole system more complete.

In the current implementation, the bitstream comes from a pre-saved file, and there is no server side online DRM operation. Therefore we should integrate the IPMPX system into the server side and test it in real streaming environment for IPMPX system.

Because the IPMPX system is very complicated, there still several functionalities that have not yet been implemented, such as the IPMP Tool within ES (Elementary Stream). The IPMPX system allows that the IPMP Tool required for content consumption can be carried by the IPMP ES. The content provider can thus insert an IPMP Tool into the ES and transfer it through the IPMP ES to the end user. This option is not implemented in current version of software. We will continue to make this system as complete as possible in the future.

Furthermore, in the current implementation, we include an IPMP decryption tool without considering the effect of packet loss. However, in a real application streaming system, the packet loss often occurs. So, we have a robust decryption tool that can tolerate the packet loss

in a streaming environment. Further, we can test its robustness by the IPMPX system embedded in MPEG-21 Testbed.

MPEG-4 IPMPX and MPEG-2 IPMPX standards are completed. However, there is a new version of IPMP to be adopted by MPEG-21. Until now, the IPMP in MPEG-21 is at the status of requirement definition. As the MPEG group moves gradually their focus toward MPEG-21, we naturally will continue the research work on MPEG-21 IPMP in the future.



References

- [1] C.A. Schultz, “Study of FPDAM ISO/IEC 14496-1:2001 / AMD3,” ISO/IEC JTC 1/SC 29/WG11 N4849, Klagenfurt, July 2002.
- [2] J. Ming and S.M. Shen, “Study Text of ISO/IEC 13818-11/FCD,” ISO/IEC JTC 1/SC 29/WG11 N5469, Awaji, Dec 2002.
- [3] J. Ming and C.A. Schultz, “MPEG-2 and MPEG-4 IPMP Extension Reference Software Architecture based on IM1,” ISO/IEC JTC1/SC29/WG11 N4850, Fairfax, May 2002.
- [4] J. Liu, et al., “WD1.0 of ISO/IEC 13818-5:1997/AMD2:2003 MPEG-2 IPMP Reference Software,” ISO/IEC JTC1/SC29/WG11 M9840, Trondheim, July 2003.
- [5] C.J. Tsai, M. van der Shaar and Y.K. Lim, “Working Draft 3.0 of ISO/IEC TR2100-12 Multimedia Test Bed for Resource Delivery,” ISO/IEC JTC1/SC29/WG11 MPEG2003/M10299, Hawaii, December 2003.
- [6] C.N. Wang, et al., “FGS-Based Video Streaming Test Bed for MPEG-21 Universal Multimedia Access with Digital Item Adaptation,” ISO/IEC JTC1/SC29/WG11 MPEG2003/M8887, October 2002.
- [7] J. Bormans and K. Hill, “MPEG-21 Overview v.4,” ISO/IEC JTC1/SC29/WG11 N4801, Fairfax, May 2002.
- [8] J. King, et al. “MOSES Progress report on MPEG-4 IPMPX,” ISO/IEC JTC 1/SC 29/WG11 M9161, Awaji, Dec 2002.
- [9] M. Carson and D. Santay, “NIST Net – A Linux-based Network Emulation Tool,” ACM SIGCOMM Computer Communications Review, Volume33, Number3, July 2003.

自 傳

范振韋：民國 1980 年生於南投縣。2002 畢業於台灣新竹的國立交通大學電機與控制工程學系，之後進入該校電子工程所攻讀碩士學位。以 IPMP 為論文研究主題。

Chen-Wei Fan was born in NanTou in 1980. He received the BS degree in Electronic Control Engineering, National Chiao Tung University (NCTU), HsinChu, Taiwan in 2002. His current research interests are Intellectual Property Management and Protection.

