利用 MPEG-4 IPMPX 系統介面在 MPEG-21 測試平台上實現 MPEG-21 IPMP 與 REL

An Implementation of MPEG-21 IPMP and REL with MPEG-4 IPMPX Framework On MPEG-21 Testbed

研 究 生：呂家賢

指導教授：杭學鳴 博士

中 華 民 國 九 十 五 年 六 月

利用 MPEG-4 IPMPX 系統介面在 MPEG-21 測試平台上實現 MPEG-21 IPMP 與 REL

An Implementation of MPEG-21 IPMP and REL with MPEG-4 IPMPX Framework on MPEG-21 Testbed

研究生：呂家賢　　　Student: Chia-Hsien Lu

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
College of Electrical and Computer Engineering
National Chiao Tung University
In Partial Fulfillment of the Requirements
For the Degree of
Master of Science
In
Electronics Engineering

June 2006
Hsinchu, Taiwan, Republic of China

中華民國九十五年六月

利用 MPEG-4 IPMPX 系統介面在 MPEG-21 測試平台上實現 MPEG-21 IPMP 與 REL

研究生: 呂家賢　　　　　　　　　　　　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

隨著網路科技與數位媒體技術的進步，人人可以容易地製造和散佈數位化的多媒體內容。於是，如何有效的管理與保護數位內容變成一個重要的議題。本論文將介紹在 MPEG 標準中的智財權保護與管理 MPEG-21 IPMP 以及權利描述語言 MPEG-21 REL。此外，在 MPEG-4 標準內所制訂的多媒體智財權保護與管理系統 IPMPX 提供本論文一個基本的系統架構。

為了模擬一個即時傳輸系統，我們使用 MPEG-21 多媒體傳輸測試平台作為系統傳輸之用。MPEG-21 IPMP 提供保護 DIDL 物件的方法。MPEG-21 REL 能夠描述各種權利，並且提供一個可產生認證結果的架構來管理權利。為何實現這些標準規格，我們設計一組擁有 MPEG-21 IPMP 與 REL 功能之 MPEG-4 IPMP 工具，並整合在至具有 MPEG-4 IPMPX 系統介面之 MPEG-21 測試平台。一個 MPEG-4 IPMP 工具能夠透過 IPMP 訊息交換機制，要求認證結果以確認是否允許繼續處理資料。

為了安全地傳遞數位內容，我們使用伺服器端的加密工具來加密數位內容。因此我們提出一個擁有金鑰管理能力的內容保護機制，並用來確保傳輸解密金鑰的安全性。在這個機制中，我們另外設計一個能夠管理金鑰並傳遞加密過的解密金鑰至用戶端的伺服器。最終，我們詳述三個應用範例，以說明我們的系統能夠成功地保護數位資源和有效地管理權利。

An Implementation of MPEG-21 IPMP and REL with MPEG-4 IPMPX Framework on MPEG-21 Testbed

Student: Chia-Hsien Lu Advisor: Dr. Hsueh-Ming Hang

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

Abstract

As the network and digital media technologies advance, it is easy nowadays for everyone to create and distribute digital multimedia contents. How to effectively manage and protect the digital content becomes an important issue. In this thesis, the technologies of MPEG-21 IPMP (Intellectual Property Management and Protection) and REL (Rights Expression Language) are adopted to construct a DRM system. The MPEG-4 IPMPX (Intellectual Property Management and Protection Extension) system is also used as a basic framework for our implementation.

To simulate a real-time streaming system, we use the MPEG-21 Testbed as our multimedia delivery platform, which is part 13 of the MPEG-21 standard for testing the multimedia resource delivery. The MPEG-21 IPMP provides ways to protect a DIDL element. The MPEG-21 REL is able to describe various kinds of rights and it provides an authorization model to generate an authorization proof to manage the rights. To implement these standard specifications, we design a set of IPMP Tools, which owns the functionalities of the MPEG-21 IPMP and REL, and we integrate this set of tools into the MPEG-4 IPMPX framework on the MPEG-21 Test Bed. Through the IPMP Message exchange mechanism, an IPMP Tool can request an authorization proof to check if the data processing is permitted.

In order to deliver digital contents securely, we encrypt digital contents by using a server-side encryption Tool. To ensure the security of delivering decryption keys, we propose a content protection mechanism with a key management mechanism. In this mechanism, we design a key server that manages decryption keys and sends encrypted decryption keys to the client. At the end, we develop three application examples to demonstrate that our system can successfully safeguard digital resources and effectively manage the rights.

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，老師對研究的熱忱與謙虛的學習態度是我學習上的榜樣。在做研究的過程中，遇到不少的問題與困難，老師總是適時的給予幫助與指點方向，讓我能順利的完成。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，讓我感到溫暖受用並度過了心情的低潮，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一個極佳研究環境，讓我在研究中有充足的資源可以運用。也感謝實驗室全體成員，營造了一個充滿活力與和諧氣氛的環境，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。另外，感謝張峰誠學長，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與系統設計的知識。也感謝蔡家揚學長和蔡崇諺、陳治傑、林鴻志、陳旻宏等實驗室的伙伴，適時提供技術上的支援與生活上的幫忙，並陪伴我走過兩年的研究生歲月，感謝這一切，使得論文研究能夠順利的進行。

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

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

List of Figures

Chapter 1

Introduction

As network technology advances, it gets increasingly popular to transport and search digital information. In addition, various efficient compression and encoding algorithms allows efficient digitalization of multimedia sources. People can easily create their own video/audio clips and distribute these clips among their relatives and friends. Issues concerning about the protection of the information delivery against eavesdropping and the unauthorized distribution draw the attention of the public. Therefore, it becomes an important topic to provide a secure platform which can safeguard both the intellectual properties of authors and the rights of consumers.

Digital Rights Management (DRM) is a systematic concept that protects and manages digital assets. An essential element to achieve Right Management is "rights expression", which represents the rights that authorizes to the users to consume the resource under specific conditions or constraints. The right entity can be described by a digital rights expression, which offers exact definitions of various rights declarations. There exist two rights expression languages, Open Digital Rights Language (ODRL) [1] and Rights Expression Language (REL) [2].

Since Rights Management becomes increasingly important, there are several groups that to define DRM systems with the Rights Management capability such as OMA (Open Mobile Alliance) [3] and TIRAMISU (The Innovative Rights and Access Management Inter-platform Solution) [4]. In OMA DRM v1.0, it adopts the ODRL as it rights expression langrage and supports simple rights control, such as Forward Lock and Combined Delivery. In TIRAMISU, it adopts the MPEG-21 REL as its rights expression langrage. In this thesis, we will focus on the MPEG-21 REL.

MPEG-21 is a standard produced by the Moving Picture Expert Group (MPEG) and it defines "a multimedia framework to enable transparent and augmented use of multimedia resources across a wide range of networks and devices used by different communities."[5] There are two parts in MPEG-21 that are highly related to DRM. MPEG-21 Part 4 IPMP [6] defines the high level schema which protects and manages digital items. Because all definitions of MPEG-21 IPMP are mostly concepts, we use the MPEG-4 IPMP Extension [7] system, a Digital Rights Management interface and architecture, as the vehicle for implementation.

MPEG-21 Part 5 REL [2] is an XML-based language that can declare an authorized distribution for the use of any content, resource, or service owned by specific users. It provides flexible, exact, and rich representation of rights. It can be used by various applications because of the interoperability of this language. Besides, people who use REL to express rights management can define their own extension and create a specific profile according to their applications.

The goal of this thesis is to study the MPEG-21 IPMP and REL standards and to design an implementation of a DRM system which protects digital assets and performs rights management. To fulfill this objective, we integrate the MPEG-21 IPMP and REL into the MPEG-4 IPMPX framework, and choose the MPEG-21 Testbed [8] [9], a multimedia test bed for resource delivery developed by MPEG group, as the streaming platform with client-server architecture. Because the MPEG-4 IPMPX has a concrete definition of APIs and DRM messaging operations, we use it as the platform for developing IPMP Tools with the MPEG-21 IPMP and REL specifications.

Our thesis is organized into five parts. First, we introduce the concepts and specifications of MPEG-21 REL in Chapter 2. Second, in Chapter 3, we give an overview of the MPEG-21 DID and IPMP. Chapter 4 gives the structure of the MPEG-4 IPMPX and the MPEG-21 Testbed. Then, we describe the details of our design and implementation of the MEPG-21 IPMP and REL within the MPEG-4 IPMPX framework on the MPEG-21 Testbed. Finally, we design several application examples to demonstrate the functionalities of our designed DRM system.

Chapter 2

MPEG-21 Rights Expression Language

The ISO/IEC MPEG, Moving Picture Experts Group, issued a Call for Proposals during its 57th meeting in Sydney for a Rights Data Dictionary and a Rights Expression Language in July 2001. After processing responses to this call during the 58th meeting in Pattaya in December 2001, MPEG-21 Part 5 'Rights Expression Language (REL)' advanced to FDIS [Final Draft IS] status as ISO/IEC FDIS 21000-5. On 2004-04-01, MPEG-21 Part 5 was published as an ISO Standard: *ISO/IEC 21000-5:2004, Information technology — Multimedia framework (MPEG 21) — Part 5: Rights Expression Language [REL]*.

In the following sections, we will give an overview of MPEG-21 Part 5 'Right Expression Language' (abbreviated as the MPEG REL), which contains objectives, data model, authorization model, and the mechanism of extensibility and profiling.

2.1 Objectives

The MPEG REL is an XML-based rights expression language that can declare an authorized distribution for the use of any content, resource, or service owned by specific users. Three goals have to be satisfied. First, it must be a machine-interpretable language with unambiguous syntax and semantics. Second, an authorization model is needed to determine if a principle has the right to act on a resource according to REL expressions. Third, it has to be rich enough to express a wide variety of business models in the end-to-end distribution value chain and to enable multimedia distribution and usage of all types of digital resource.

The MPEG REL provides flexibility and interoperability to support transparent, augmented use of digital resources in publishing, distributing, and consuming digital content while protecting such content and exercising the rights, conditions, and fees specified for the content. The MPEG REL also describes access and usage controls for digital content when financial exchange is not part of the terms of use, and it supports the exchange of sensitive or private digital content. For enterprises and individuals, this language can be used to enable the authorized distribution and persistent protection of valuable data, content, and resources in accordance with privacy and confidentiality requirements.

2.2 Data Model

The primary function of the MPEG REL is to specify rights relating to digital resources (such as content, services, or some other applications). Using this language, people can easily distribute their digital resources to identified principals (such as users, groups, devices, and systems). An identified principal has specific rights for exercising on those resources under the terms and conditions.

For example, consider a song “wifi_audio”, distributed by Music Station to John, an AAC player’s owner. A typical REL instance would be like this: “under the authority of Music Station, John is granted with the right to play ‘wifi_audio’ during June 2006.” Figure 2-1 shows the structure of this REL expression.

```
license
    grant
        John
        play
        wifi_audio.aac
        during June 2006
    issuer
        Music Station
```

Figure 2-1 Example of a right expression language (REL) expression

According to the semantics of MPEG REL, “John” is a principal; “play” is a right; “wifi_audio.aac” is a resource; “during June 2006” is a condition; and “Music Station” is an issuer of the right.

This example has demonstrated the essence of an MPEG REL expression, a statement that an issuer states that a principal has some rights to a resource under some condition. The right-granting portion of this statement (e.g., “John is granted with the right to play ‘wifi_audio’ during June 2006”) is called a grant. The left-granting portion of this statement (e.g., “Music Station”) is called an issuer. The whole statement is called a license.

The basic data model of an MPEG REL license is shown in Figure 2-2. In this data model, a license can contain none or many grants and issuers. A grant is constructed by a principal, a right, a resource, and a condition. We will describe each component of a license in the following sections.

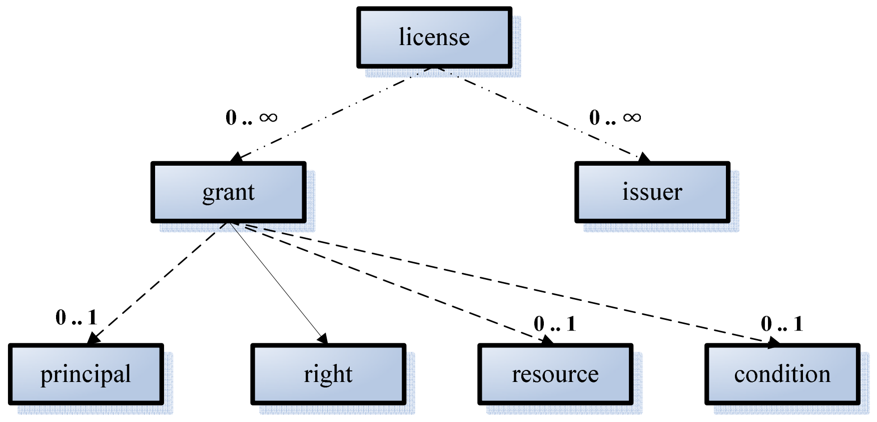


Figure 2-2 Data model of a license

2.2.1 License

The most important concept in the MPEG REL is the License. Basically, a license contains the following:

- a set of grants describing that certain principles have certain rights corresponding to specific resources under certain conditions
- a set of issuers that identify one or more principles in this set of grants
- some other related information, such as encrypted licenses.

Conceptually, a license tells us that a set of grants for specific is identified by such a set of issuers.

2.2.2 Grant

A grant contains four basic components as follows:

- the principles to whom the grant is issued
- the right which the grant specifies
- the resource to which the right in the grant applies
- the condition which has to be satisfied for exercising the rights

2.2.3 Principal

A principal is the "subject" in a grant. In order to identify this entity, the MPEG REL uses information to denote it uniquely. This information includes some associated authentication mechanism by which the principle can prove itself. For example, an X.509 certificate is one of authentication proves. The MPEG REL defines three elements to represent principals, as illustrated below:

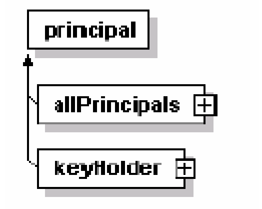


Figure 2-3 Principle Modal [10]

Each element is described in the following items:

- Principle: An abstract element from which al other principal element are derived. This element is the substitution head for all principal elements.
- AllPrincipals: A container of other principal elements that represent a set of principals acting as a single element. Each principal element of allprincipals is necessary to authorize for executing a right.
- keyHolder: A principal is identified as the possessor of a secret key, such as the private key of a public/private key pair.

Besides, extension to the MPEG REL can define additional principal elements that use other technologies to authenticate principals. In other words, there are three way to declare a principle: (1)AllPrincipals, (2)KeyHolder, (3)Use extention to MPEG REL.

Figure 2-4 shows an example for declaring a principal with a keyHolder element. In Figure 2-4, if a principal is identified as this key holder, it must present a cryptographic key that match the content of the keyHolder element.

```
<r:keyHolder licensePartId="Alice">
   <r:info>
      <dsig:KeyValue>
         <dsig:RSAKeyValue>
            <dsig:Modulus>AliM4ccyzA==</dsig:Modulus>
            <dsig:Exponent>AQABAA==</dsig:Exponent>
         </dsig:RSAKeyValue>
      </dsig:KeyValue>
   </r:info>
</r:keyHolder>
```

Figure 2-4 Alice is identified as an RSAKey

2.2.4 Right

A right is the "verb" that a principal can exercise against some resource under some conditions. Typically, a right specifies an act or a set of acts that a principal can perform on or using the associated resource. A set of commonly used rights has been provided by MPEG REL, such as play, print, revoke, issue, and obtain. The flowing figure demonstrates how to declare a right. Besides, the prefix "mx" means this right is belong to Multimedia Extension. The extensibility of MPEG REL will be introduced in section 2.4.

```
<mx:play/>
```

Figure 2-5 A play right

2.2.5 Resource

A resource is the "object" to which a principal can be granted a right. A resource can be a digital work (such as a video or audio file, or an image), a service (such as an email service or B2B transaction service), or even a piece of information that can be owned by a principal (such as a name, an email address, a role, or any property or attribute). In addition, a resource can be a grant or a grantGroup in conjunction with issue and obtain Rights. The MPEG REL provides mechanisms to encapsulate the information necessary to identify a particular resource or a collection of resources with some common characteristics. As illustrated in Figure 2-6, an audio file is represented as an URI.

```
<digitalResource>
    <nonSecure
        IndirectURI="http://www.onlinemusic.com/mySong.mp3"/>
</digitalResource>
```

Figure 2-6 An audio file is represented as a digitalResource with an URI

2.2.6 Condition

A condition specifies the terms, conditions, and obligations under which rights can be exercised. A simple condition is a time interval within which a right can be exercised, and a more complicated condition may require the existence of a valid, prerequisite right that has been issued to some principals by some trusted entities. Moreover, a condition can specify a logical "AND" operation of several conditions by the "allConditions" element. A logical "OR" operation of several other conditions also needs to be specified for a grant. For example, to represent an "OR" operation of three conditions in a grant, a grant has to be duplicated three times except the condition element. Then, each duplicate grant is inserted with different condition.

Figure 2-7 shows a conjunction of two conditions, validityInterval and sx:exerciseLimit. Therefore, a right can be exercised only when two conditions are both satisfied.

```
<allConditions>
  <validityInterval>
    <notBefore>2000-12-24T23:59:59</notBefore>
    <notAfter>2002-12-24T23:59:59</notAfter>
  </validityInterval>
  <sx:exerciseLimit>
    <sx:count>5</sx:count>
  </sx:exerciseLimit>
</allConditions>
```

Figure 2-7 A conjunction of two conditions

2.2.7 Issuer

An issuer is an element within a license that identifies a principal and issues the license. The issuer can also contain a digital signature of the license signed by the issuer to signify that the principal does indeed bestow the grants contained in the license.

In Figure 2-8, an example illustrates how to describe an issuer which contains a digital signature and the issued time.

```
<issuer>
     <dsig:Signature>
          <dsig:SignedInfo>
               <dsig:CanonicalizationMethod
                    Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
               <dsig:SignatureMethod
                    Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
               <dsig:Reference>
                    <dsig:Transforms>
                         <dsig:Transform
                              Algorithm="urn:mpeg:mpeg21:2003:01-REL-R-NS#license"/>
                    </dsig:Transforms>
                    <dsig:DigestMethod
                         Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                    <dsig:DigestValue>PB4QbKOQCo941tTExbj1/Q==</dsig:DigestValue>
               </dsig:Reference>
          </dsig:SignedInfo>
          <dsig:SignatureValue>
          alIDoedpLvDzDWqZU+7Tdt4kg7DRt9bu5K9I6p5C32wsNb6kUNJ4q9sH/2OLPTHsUnTuPdaGncWh
     JRyoYRjJqA==
          </dsig:SignatureValue>
          <dsig:KeyInfo>
               <dsig:KeyValue>
                    <dsig:RSAKeyValue>
                       <dsig:Modulus>
                       g8NRYMG307NqJgmZG8TlUOp+9sQjsAai+hlLpkBiLaf4RhvjS3pD0dvy1YosEjK
                    L8mk/KTGniC+pY4ia5kLByQ==
                       </dsig:Modulus>
                       <dsig:Exponent>AQABAA==</dsig:Exponent>
                    </dsig:RSAKeyValue>
               </dsig:KeyValue>
```

Figure 2-8 An Issuer

2.3 Authorization Model

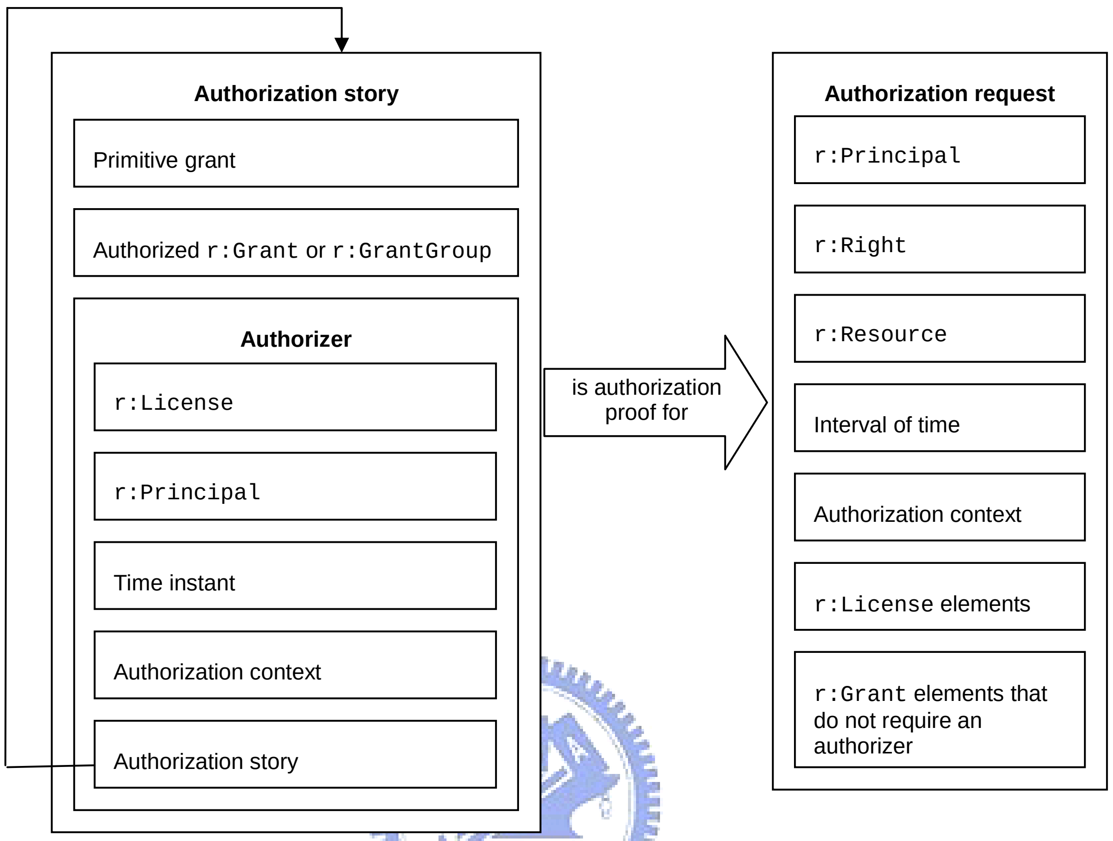


Figure 2-9 Authorization model [2]

MPEG REL defines an authorization model as shown in Figure 2-9 for clearly defining the semantics of Licenses. An authorization model includes an authorization request, an authorization context, an authorization story, and an authorizer. The most important concept of an authorization model is to generate an authorization proof according to an authorization story when an authorization request is processed.

2.3.1 Authorization request

An authorization request contains several items as illustrated in Figure 2-9, and uses these items to ask the following question ”is it permitted for the given Principle to perform the given Right upon the given Resource during the given time interval based on the given authorization context, the given set of Licenses, and the given trusted root?”

2.3.2 Authorization context

An authorization context is constructed by a set of properties, each of which is described by one name, one value, and one statement. Therefore, each property has an unique name in the same authorization context. A statement in one property can be used to either verify an issuer or validate a condition.

In Table 1, we can see that a property name is composed by one Qualified name and zero or more parameters.

Table 1— Example of Authorization Context properties [2]

Property name	**Property value**	**Statement represented**
r:issueTime(*l*, *p*)	*i*	*l* is an `r:License`, *p* is an `r:Principal`, *i* is a time instant, and *p* issued *l* at *i*. NOTE 1: The r:timeOfIssue field in an r:IssuerDetails can be useful in determining when *p* issued *l*. However, it is wise to give consideration as to whether the issuer is trustworthy and, when in doubt, to seek additional proof, such as in the form of signatures and countersignatures. NOTE 2: The r:Issuer can be useful in determining whether *p* issued *l*. However, it is wise to give consideration as to whether the information given in the r:Issuer is trustworthy and, when in doubt, to seek additional proof, such as in the form of signatures and countersignatures.
r:issueContext(*l*, *p* ,*h* ,∑)	true	*l* is an r:License, *p* is an r:Principal, *h* is either an r:Grant or an r:GrantGroup, ∑ is an authorization context, and the statements represented by the properties in ∑ are all true for the purposes of

		establishing the permission for the Principal identified by *p* to include an r:Grant or r:GrantGroup that is Equal to h as *l*/r:grant or *l*/r:grantGroup when issuing *l*.

2.3.3 Authorization story

An authorization story contains a primitive grant, an r:Grant or an r:GrantGroup, and an authorizer in order. Their meanings are stated as following:

- A primitive grant means that no item in it was represented as a variable and is also a necessary element that an authorization request has to match.
- An r:Grant or an r:GrantGroup is authorized by an authorizer and is part of a License by some Principle at some time instant based on some authorization context.
- An authorizer contains five ordered items, an r:License, an r:Principle, a time instant, an authorized context, and an authorization story.

In other words, an authorization story possesses all necessary information to describe the following fact:

"A given primitive grant may be derived from a given grant authorized by a given authorizer, which identifies a principle from a license at a time instance based on an authorization context and supported by an (additional) authorization story."

2.3.4 Authorization proof

An authorization proof is the result of an authorization request for an authorization story. If the result is true, several rules have to be satisfied:

- the principle, right, and resource in the authorization request must match those in the primitive grant in the authorization story;
- the authorization context in the request must satisfy the conditions in the authorization story or the conditions in the authorization story is absent;
- the primitive grant indeed is derived from the grant;
- the issuance of the grant in the first authorization story is recursively authorized via the additional authorization story in the authorizer in the first authorization story;

and

- the recursive authorization eventually terminates in one of the trust roots given in the authorization request.

When the authorization proof is true, it means that there is at least one authorization story in a License that provides an authorization proof for the authorization request. [11]

2.4 The mechanism of extensibility and profiling

Various applications require different levels of complexity and flexibility in the REL, and specific industries and user communities may need to modify the MPEG REL language to better meet their specific needs. Hence, the MPEG REL is designed to be extensible and is itself specified in extensions.

The syntax and grammar of the MPEG REL are described using the XML Schema. It means that the MPEG REL can offer a high degree of flexibility in its extensibility. The MPEG REL is organized into several architectural parts as shown in Figure 2-10, and each part is defined as its own XML namespace:

◆ A core schema defines general concepts that form the basic architecture of the language, particularly those who have to do with evaluation of a trust evaluation. The namespace of this schema is identified with the prefix, **r:**. In some cases, this prefix may be omitted.

◆ A standard extension schema defines concepts that are generally and broadly useful and applicable to DRM usage scenarios. The namespace of this schema is identified with the prefix, **sx:**.

◆ A multimedia extension schema defines DRM concepts (e.g., rights, resources, and conditions) specifically related to multimedia content such as books, video, and audio. The namespace of this schema is identified with the prefix, **mx:**.

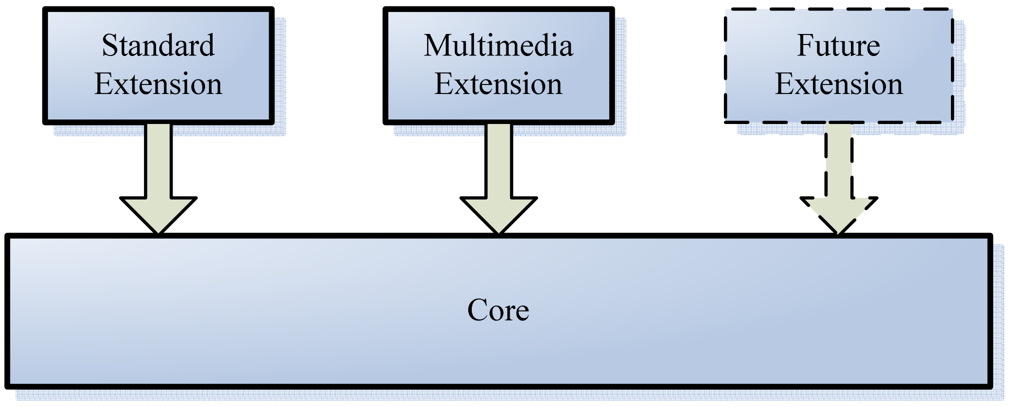


Figure 2-10 Extensibility model [10]

There may be some parties who want to make domain-specific extensions to the MPEG REL and its future extensions. This can be achieved by using the existing standard XML Schema and XML Namespace mechanisms [12] [13]. In general, an extension to the MPEG REL can define its own principals, rights, resources, and associated conditions according to specific usage models and technical applications. For example, the MPEG REL multimedia extension defines specific rights, *mx:play* and *mx:move*, for using digital resources, and the MPEG REL standard extension defines conditions, *sx:FeePerUse* and *sx:FeePerInterval*, for specifying sufficient payment of using digital resources.

In addition to add extensions to the baseline REL, users can remove unnecessary items in this baseline language. This is called profiling. It is a process to choose REL items for a specific purpose, and thus to form a subset of the language. The idea of profiling is very important as content moves to many various devices in different ways. When it comes to a resource-limited device, such as a smart phone, power consumption must be as small as possible for maintaining longer battery-life. Therefore, a full REL support is not sufficient, and a compact version is enough to satisfy the requirement of its usage.

2.5 The relationship between REL and other parts in MPEG-21

There are three parts in MPEG -21 related to the MPEG REL:

- The MPEG-21 Part2 Digital Item Declaration (DID) defines a language which describes a digital item. Rights expressions can also be included in digital item declarations. (see

3.1.5)

- The MPEG-21 Part3 Digital Item Identification (DII) provides a normative way to express how this identifications can be associated with DIs, containers, components, and/or fragments thereof by including them in a specific place in the DID. The MPEG REL can use the identification mechanism defined in DII to identify digital items described by DID.
- The MPEG-21 Part6 Rights Data Dictionary (RDD) specifies a dictionary of terms that can be used by the MPEG REL to describe rights. It also provides mechanisms for extending the dictionary to add new terms.

Chapter 3

MPEG-21 Part 2 DID and Part 4 IPMP

Digital Item (DI) is the core concept of MPEG-21 Multimedia Framework. In addition, DIs are also the basic transaction unit in MPEG-21. A DI, as defined specifically by MPEG-21 [5], consists of resources, metadata, and structure.

There are two important features of a DI: [14]

- A DI will either operate in application spaces where there are agreed “rules” for presentation or may contain presentation descriptions as resources.
- The configuration of a DI includes factors such as the usage environment, terminal capabilities, and network conditions. This assists in enabling transparent and augmented use of DIs across a wide range of networks and devices used by different communities.

The representation and protection of DIs are both important issues for our consideration. Hence, the declaration of a DI, MPEG-21 Part 2 DID, and the DI protection, MPEG-21 Part 4 IPMP, will be introduced in the following sections.

3.1 Overview of MPEG-21 DID

In this section, we discuss DIs and the structure of them. We will describe the model for declaration a DI and it’s representation in XML. Then, we will analyze an example DI and the mechanism of validation.

3.1.1 DIs and Declaration

The DID formally represents and identifies the constituent resources (e.g., video and audio files) and the metadata (e.g., Dublin Core [15] descriptions) Further, the DID binds together individual and groups of resources and metadata. This is further extended by the capability to allow metadata to be anchored to certain fragments in a media resource. [14]

The DID technology can be divided into three normative parts: [14]

- *Model*: The Digital Item Declaration Model describes a set of abstract terms and concepts to form a useful model for defining Digital Items. Within this model, a DI is the digital representation of “a work”, and as such, it is the ting that is acted upon (managed, described, exchanged, collected. etc.) within the model.

- *Representation*: The description of the syntax and semantics of each of the Digital Item Declaration elements, as represented in XML.
- *Schema*: An XML schema [12], [13] comprising the entire grammar of the Digital Item Declaration represented in XML.

It should be noted that the DID Model is an abstract model of fundamental materials useful for declaring DIs. This abstract model can be expressed in many ways. MPEG-21 Part2 defines a normative XML expression of the DID Model. This XML-based expression is the Digital Item Declaration Language (DIDL).

3.1.2 DIDL

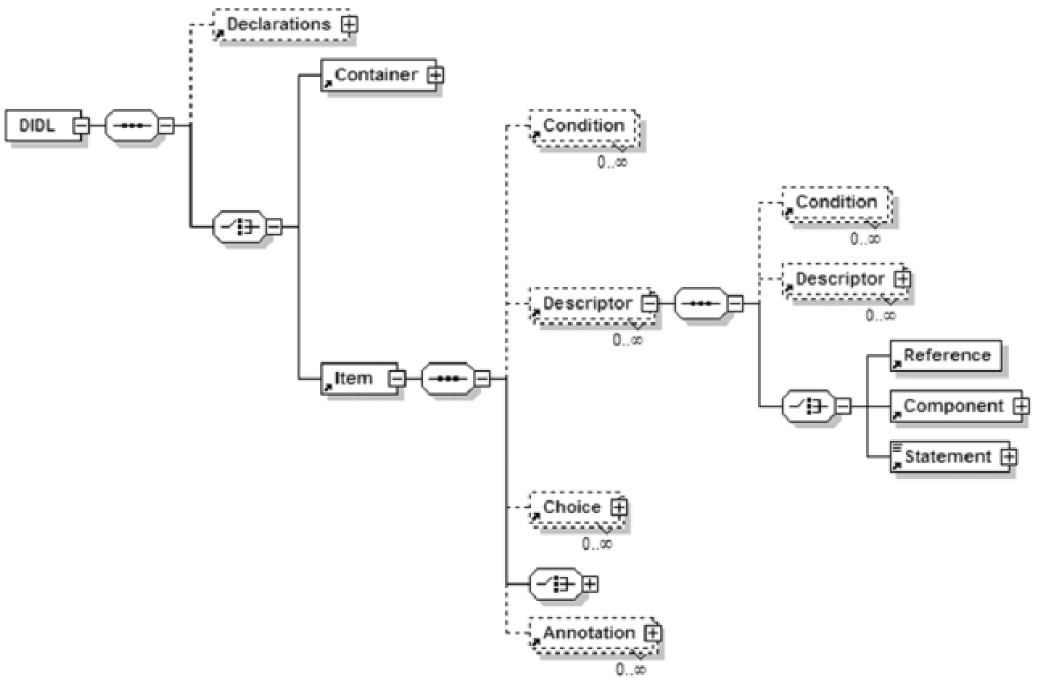


Figure 3-1 Partial Graphical representation of the DID Schema [14]

DIDL is an XML based language. The syntax of DIDL is based on an abstract structure defined in the DID Model. Figure 3-1 shows the partial graphical representation of the DIDL schema. In this section, we discuss the fifteen elements that are related to express a DI. It should be noted that in the descriptions below, the names of DIDL elements and attributes appear in **bold**, and *italics* are used when we use the reference to the fundamental materials of the DID Model.

The basic constituents of the declaration of a DI are listed below:

- The **DIDL** element is the root element of any other DIDL elements. The **DIDL** element contains either a **Container** or an **Item** as a child and it may contain an optional child, a **Declarations** element, before the **Container** or **Item**. Note that it is not part of the DID abstract Model, it exists only in the representation and schema.
- A **Container** element represents a grouping of **Item** elements and/or possibly other **Container** elements. It is also bound with a set of **Descriptor** elements. For instance, "books" represented by **Item** elements can be placed on spaces provided by "a shelf" represented by a **Container** element.
- An **Item** element represents a grouping of possible sub **Item** elements and/or **Component** elements. It may also contain **Descriptor** elements containing descriptive information about the *item* represented.
- A **Declarations** element is used to define other DIDL elements without instantiating them. The declared elements can be used later in a DID via an internal reference (see the **Reference** element below). The **Declarations** elements exist only in the representation and schema.
- A **Component** element groups a **Resource** element with a set of **Descriptor** elements. Hence, a **Component** element is the basic building block of digital content within a DID.
- A **Resource** element defines an individually identifiable resource such as a video or audio clip, an image, or a textual asset. In the DID Model, there is only one *resource* in a *component*. However, in DIDL, there may be one or several **Resource** elements in a **Component** element. It makes multiple references (for example at different resource locations) for the represented resource possible. Each **Resource** element in a **Component** element must refer to a bit equivalent resource.
- A **Descriptor** element describes information to be associated with its parent element. The information can be contained either in a **Component** or a **Statement**. The latter contains textual information that can be bound to other elements.

After introducing the basic constituents of the declaration of a DI, the next four elements affect the configuration of a DI:

- A **Choice** element groups a set of related **Selection** elements that can affect the configuration of an Item. There may be no, one, or several selected **Selection** elements in a **Choice** element dependent on the values of two attributes, **minSelections** and **maxSelections**.

- A **Selection** element defines a specific decision about a particular **Choice** element which represents an associated predicate that will affect one or more **Condition** elements in an **Item** element. At configuration time, if a **Selection** element is chosen, its predicate becomes true; if it is refused, its predicate becomes false; if it is left undecided, its predicate is left undecided.
- A **Condition** element indicates that the inclusion of the parent elements is dependent on a defined set of *predicates* associated with the **Selection** elements. In order to satisfy the **Condition** element, the **except** attribute of the **Condition** element lists the *predicates* must be false, and the **require** attribute lists the *predicates* must be true. A set of **Condition** elements can express a Boolean combination of predicate tests. For example, a conjunction (an AND relationship) can be represented by a single **Condition** element, and a disjunction (an OR relationship) can be represented by Multiple **Condition** elements within a given parent element.
- An **Assertion** element allows the partial or full configuration state of a **Choice** element to be defined by asserting true, false, or undecided values for a number of predicates associated with the **Selection** element in a parent **Choice element**.

The following two elements are correlated with the description of resources and other elements, and the other one represents a linkage between two DIDL elements:

- An **Anchor** element binds a set of **Descriptor** elements to a fragment, a specific location or range, within the resource identified by the **Resource** element within the parent **Component** element.
- An **Annotation** element allows additional information, such as **Descriptor** and/or **Anchor** elements, to be associated with an identified element without changing the original content of the element.
- A **Reference** element represents a reference to one of the following DIDL elements: **Container**, **Item**, **Component**, **Descriptor**, or **Annotation**. Using a **Reference** element, the parent element of the Reference element can include a reference DIDL element located either internally or externally. The **Reference** element exists only in the representation and schema.

3.1.3 DIDL Example

We take a music album as one simple example of a DI, as shown in Figure 3-2. This DI is constructed by one root DIDL element. This DIDL element has a single **Item** element which has an id attribute, used for internal or external referencing of an Item. This Item element is composed of two **Descriptor**/**Statement** combinations, one **Choice** element, and one “sub” **Item** element in order.

1) The first **Descriptor**/**Statement** combination represents a MPEG-21 Part 3 DII Identifier [16]. This identifier identifies the root **Item** element, the “subject” of the DID. Other content in the DID can be identified with different identifier.

2) The second **Descriptor**/**Statement** combination expresses a human readable text including the title and artist. This information can be represented as the XML metadata form, e.g. the MPEG-7 standard [17].

3) The **Choice** element is identified for reference purpose as “**BR**” and represents a choice in the DID between content as a bit rate of 128 kb/s or 192 kb/s. One, and only one, of these bit rates must be selected (the configuration state of the **Choice**) and thus both **minSelections** and **maxSelections** are set to 1. Each of the **Selection** elements has an appropriate **select_id** that can be used to resolve the state of **Condition** elements elsewhere in the DID. [18]

4) The subsequent **Item** element contains the album tracks. The first inside element is a **Descriptor** element which includes a **Conponent**/**Resourece** combination representing a JPEG image of the cover art of this CD: “debutcd.jpg”. Each track is represented as a sub **Item** element of the album tracks **Item**, and contains one **Descriptor**/**Statement** combination with the track name and then two **Component** elements. Only one **Component** will be available depending on the state of the **Condition** elements which correspond to the **Selection** elements in the bit rate **Choice**. However, only one track is shown in Figure 3-2. The last of further sub **Item** elements of the root **Item** expresses supplementary information. There is a **Component** element bound in this sub **Item**. The **Component** element includes a **Descriptor**/**Statement** combination and a **Resource** element which links to a web site. The **Descriptor**/**Statement** combination contains descriptive information about the **Resource**. In this case, it describes the purpose of the **Resource**, i.e. where the CD of the album can be purchased. [14]

```
<?xml version="1.0" encoding="UTF-8"?>
<DIDL xmlns="urn:mpeg:mpeg21:2002:01-DIDL-NS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:mpeg:mpeg21:2002:01-DIDL-NS didl.xsd"
xmlns:dii="urn:mpeg:mpeg21:2002:01-DII-NS">
  <Item id="TONY_EYERS_YIG001">
    <Descriptor id="Item_Identifier">
      <Statement mimeType="text/xml">
        <dii:Identifier>urn:mpeg21:dii:trc:AU-000-YIG001</dii:Identifier>
      </Statement>
    </Descriptor>
    <Descriptor>
      <Statement mimeType="text/plain">Black Mountain Harmonica - Tony Eyers</Statement>
    </Descriptor>
    <Choice choice_id="BR" default="BITRATE_192k" maxSelection="1" minSelection="1">
      <Selection select_id="BITRATE_128k"/>
      <Selection select_id="BITRATE_192k"/>
    </Choice>
    <Item id="ALBUM_TRACKS">
      <Descriptor>
        <Component>
          <Resource mimeType="image/jpeg" ref="images/debutcd.jpg"/>
        </Component>
      </Descriptor>
      <Item id="TRACK1">
            <Descriptor>
                  <Statement mimeType="text/plain">Billy in the Lowground</Statement>
            </Descriptor>
            <Component>
                  <Condition require="BITRATE_128k"/>
                  <Descriptor>
                        <Statement mimeType="text/plain">Billy(128kbit)</Statement>
                  </Descriptor>
                  <Resource mimeType="audio/mpeg" ref="128k/Billy_128k.mp3"/>
            </Component>
            <Component>
                  <Condition require="BITRATE_192k"/>
                  <Descriptor>
                        <Statement mimeType="text/plain">Billy(192kbit)</Statement>
                  </Descriptor>
                  <Resource mimeType="audio/mpeg" ref="192k/Billy_192k.mp3"/>
            </Component>
      </Item>
      <!--........-->
      <!-- Track 13 -->
    </Item>
    <Item id="TONY_INFO">
      <Component>
            <Descriptor>
                  <Statement mimeType="text/plain">Purchase CD</Statement>
            </Descriptor>
            <Resource mimeType="text/html" ref="http://www.cdbaby.com/cd/tony"/>
      </Component>
    </Item>
  </Item>
</DIDL>
```

Figure 3-2 Example DID for an (abbreviated) one track music album (black mountain harmonica by Tony Eyers). [14]

3.1.4 DIDL Validation

DID creates a DI hierarchy which gives powerful structure semantics that take DIDL beyond a simple descriptive schema. Validating a DID is more complex than validating an XML document is. So, to validate a DID document with respect to DIDL, the following two conditions have to be satisfied simultaneously:

1. A DID document is validated against the DIDL schema.
2. A DID document is subjected to each of additional validation rules.

Only two validating rules are considered in this section:

- **Item** *Validation Rules*: The **Item** element has a single validation rule which is intended to prevent that **Item** from the effective deletion due to the state of a **Condition** as a result of a **Selection** included somewhere in the hierarchy of that **Item**. The exact wording of the validation rule in 21000-2 is:

  "An **Item** element cannot be conditional on any of its descendant **Selection** elements. In other words, an **Item** cannot contain a **Condition** element specifying a **select_id** value that identifies any descendant **Selection** element within the element."

- **Condition** *Validation Rules*: **Condition** elements are intrinsically linked to **Selection** elements. When a **Condition** is found during the process of parsing a DID document, it is needed to look for any associated **Selection** elements according to this Condition's attributes. [14] To facilitate the search, there are two simple validation rules listed in 21000-2 for the Condition element:

  "Each ID value specified in the require and except attributes must match a select_id attribute value defined in a Selection element in a Choice that is a child of an Item that is an ancestor of the Condition" [14]

  "Empty Conditions are not permitted. Therefore, it is not valid for a Condition element to have neither a require attributes nor an except attribute" [14]

3.1.5 Integration role of DID: *REL and DIA*

The information in a DI is not only a collection of metadata and resource but also other configuration data, such as rights data. The DID defines a building structure that makes a DI a function virtual object. Therefore, we can consider how other parts of MPEG-21 fit into the

DID structure. In this section, our discussion focuses on the integration of Rights Expressions [2] and Digital Item Adaptation (DIA) [19] into the DID. The key elements for integration are the Descriptor/Statement combinations. Because these combinations represent a container of metadata relevant to a resource, they can also be used to contain the Right Expressions or DIA metadata. In Figure 3-3, a simple example of an integration of an REL license is shown. Using this method, a DID can even include the Right Expressions by reference.

```
<Descriptor>
        <Statement>
                <REL:license>elements of a license</REL:license>
        </Statement>
</Descriptor>
```

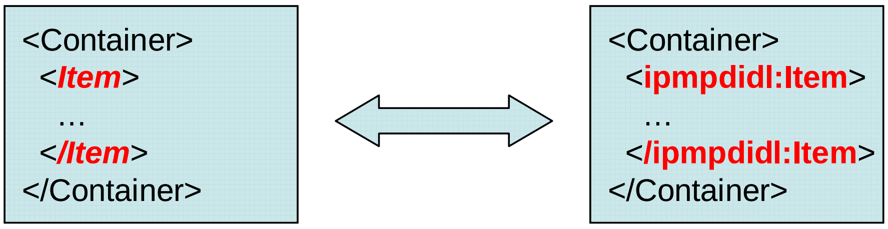
Figure 3-3 Integration of a license in a Descriptor/Statement combination

3.2 MPEG-21 IPMP

The goal of the MPEG-21 IPMP is to provide the management of rights and intellectual property through the use of protected Digital Items. However, DIDL is defined as cleartext XML. The valuable contents of the Digital Item are exposed to view, and unauthorized perusal of a Digital Item may happen. Therefore, a protected Representation of DID Model Structure is proposed as IPMP DIDL.

3.2.1 IPMP DIDL

To protect a specific part of the Digital Item structure by encapsulating it in IPMP DIDL, IPMP DIDL elements must be interchangeable with DIDL equivalents within a DID document. Figure 3-4 demonstrates how an **Item** within a **Container** may be represented as either DIDL (left) or IPMP DIDL (right). [20]



```
<Container>
  <Item>
    …
  </Item>
</Container>
```

```
<Container>
  <ipmpdidl:Item>
    …
  </ipmpdidl:Item>
</Container>
```

Figure 3-4 Element interchangeability [20]

In order to fulfill the scenario, the IPMP DIDL elements must be in the substitution group as their DIDL equivalents. Hence, we can recognize that DIDL and IPMP DIDL are both Representations of the DID Model. As shown in Figure 3-5, both the IPMP DIDL elements and the DIDL elements extend abstract types defined for the DID model. They are interchangeable within a Digital Item. As the IPMP DIDL schema imports the DIDL schema as defined in ISO/IEC 21000-2, all documents conforming to the DIDL schema are also conformable to the IPMP DIDL schema. [6]

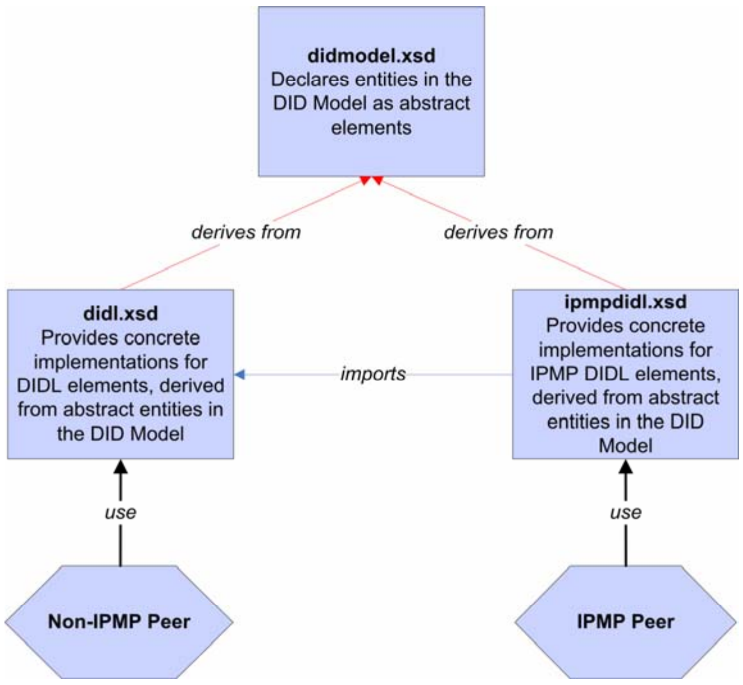

didmodel.xsd
Declares entities in the DID Model as abstract elements
derives from
derives from
didl.xsd
Provides concrete implementations for DIDL elements, derived from abstract entities in the DID Model
imports
ipmpdidl.xsd
Provides concrete implementations for IPMP DIDL elements, derived from abstract entities in the DID Model
use
use
Non-IPMP Peer
IPMP Peer

Figure 3-5 Schema relationship between DID model, DIDL and IPMP DIDL

3.2.1.1 IPMP DIDL Elements for the DIDL model

Each of the IPMP DIDL elements below represents the corresponding entity in the DID model, and corresponds to an element defined in the DIDL Representation as defined in ISO/IEC 210002 DID. Each of the following IPMP DIDL elements has the same semantics as its DIDL counterpart. [6]

- <ipmpdidl:Container>
- <ipmpdidl:Item>

- <ipmpdidl:Descriptor>
- <ipmpdidl:Statement>
- <ipmpdidl:Component>
- <ipmpdidl:Anchor>
- <ipmpdidl:Fragment>
- <ipmpdidl:Condition>
- <ipmpdidl:Choice>
- <ipmpdidl:Selection>
- <ipmpdidl:Resource>
- <ipmpdidl:Annotation>
- <ipmpdidl:Assertion>

Since the IPMP DIDL elements listed above encapsulate the protected DIDL structure with corresponding information about the protection, the structure for carrying protected DIDL elements is shown in Figure 3-6. This structure is identical to each IPMP DIDL element. The components in this structure are discussed below.

(i) At most one ipmpdidl:Identifier element, into which an appropriate identifier for the protected Representation may be placed.

(ii) One ipmpdidl:Info element, into which information about the governance is placed.

(iii) At most one ipmpdidl:ContentInfo element, into which information about the governed contents is placed.

(iv) One ipmpdidl:Contents element, into which the governed contents is placed.

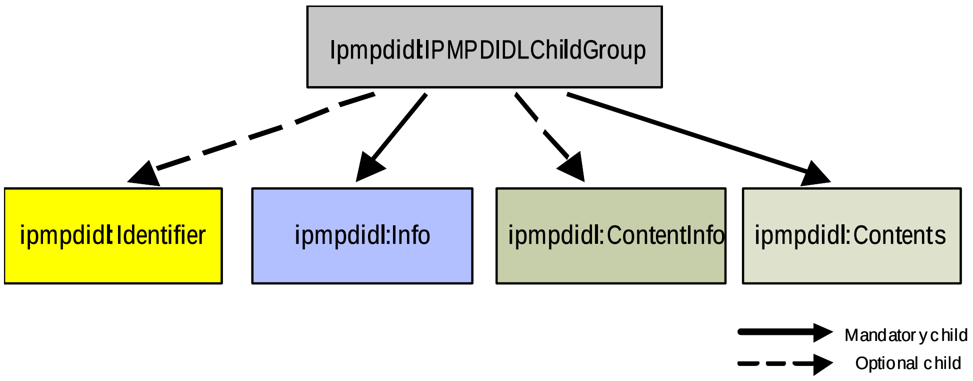


Figure 3-6 Structure of IPMP DIDL elements for the DID model [6]

3.2.1.2 IPMP DIDL elements particular to the IPMP DIDL Representation

There are five elements particular to the IPMP DIDL representation.

- <ipmpdidl:ProtectedAsset>: This element is defined to provide for the communication of IPMP governance on a specific asset - i.e. content referenced from or included inline to didl:Resource.
- <ipmpdidl:Identifier>: This element acts as a placeholder for an Identifier from an appropriate namespace, to be associated with the protected Representation of the Contents.
- <ipmpdidl:Info>: This element acts as a placeholder for IPMP Information from an appropriate namespace, to be associated with the protected Representation of the Contents. It will be used by processes seeking to make the Contents available for consumption.
- <ipmpdidl:ContentInfo>: This element acts as a placeholder for metadata, to be associated with the protected contents.
- <ipmpdidl:Contents>: This element contains Contents which is protected by the parent IPMP DIDL element. This may be a protected element of the DIDL structure, for example a protected didl:Item or didl:Resource, or (if the parent IPMP DIDL element is ipmpdidl:ProtectedAsset) an asset such as an image or an audio file. In any case, the contents may be inline or referred, but not both. This contents will generally be obfuscated or protected in some form (e.g. encryption).

3.2.2 IPMP Information Schemas

IPMP Information schemas define structures for expressing information relating to the protection of content including tools, mechanisms and licenses. The IPMP Information Descriptor and the IPMP General Information Descriptor are covered in IPMP Information schemas.

3.2.2.1 IPMP Information Descriptor

The description of IPMP governance and tools is required to satisfy intellectual property management and protection for accessing a Digital Item or its parts. The root element for IPMP Information Descriptor is "IPMPInfoDescriptor", which contains a range of information related to IPMP governance and tools.

In a IPMPInfoDescriptor element shown in Figure 3-7, there are

- IPMP Tool information (section 3.2.2.1.1) includes ToolDescription and initialization settings, either directly described within the descriptor or through a reference,
- different level Rights description subjective to the usage for IPMP Tool or content (section 3.2.2.1.2), and
- the associated digital signature.

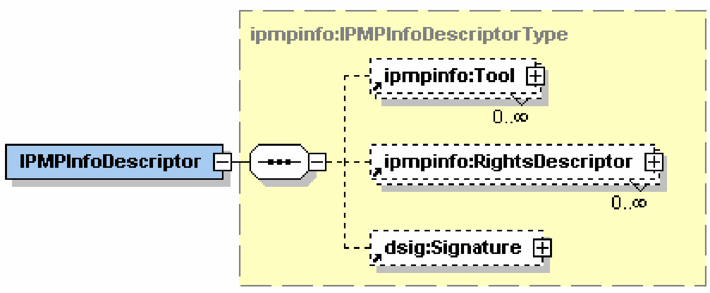


Figure 3-7 Structure of an IPMPInfoDescriptor element [6]

When used for the DIDL protection, the IPMPInfoDescriptor should be inserted into any IPMP DIDL elements and be the child element of ipmpdidl:Info. The ipmpinfo:IPMPInfoDescriptor also allows to be used to signal for other non-DIDL multimedia declaration unit protection.

3.2.2.1.1 Tool

The ipmpinfo:Tool element describes an IPMP tool. IPMP tools are modules that perform (one or more) IPMP functions such as authentication, decryption, watermarking, etc. A given IPMP tool may coordinate other IPMP tools. An IPMP Tool has the granularity that it can be single protection module, such as a single decryption tool, and can also be a collection of tools, i.e. a complete IPMP system. Tools can be executed in any order if the order attribute is not present. Similarly, the tools with the same order number can be executed in any order. If an IPMP tool is being used to protect a resource fragment, the fragment should be identified using the ipmpdidl:fragment element. [6]

3.2.2.1.2 Rights Descriptor

The ipmpinfo:RightsDescriptor element contains information about the license that governs the IPMP tool. The existence of the ipmpinfo:RightsDescriptor element under the ipmpinfo:ToolBaseDescription element indicates that there is governance for the usage of the IPMP tool. The ipmpinfo:RightsDescriptor may contain an ipmpinfo:IPMPInfoDescriptor element and a License, ipmpinfo:LicenseReference or ipmpinfo:LicenseService elements. The ipmpinfo:IPMPInfoDescriptor element contains the IPMP information related to the protected license(s). [6]

3.2.2.2 IPMP General Information Descriptor

The IPMPGeneralInfoDescriptor element represents the general control and the global governance information about IPMP tools and rights expressions relating to a complete DID. It should be carried at the outmost place of the protected DIDL. It can also be used for signaling for other non-DIDL general multimedia declaration unit protection.

The ipmpinfo:IPMPGeneralInfoDescriptor element is the root element of an IPMPGeneralDescriptor instance document. It may contain:

- an ipmpinfo:ToolList element, which describes a list of the necessary tools to access the protected content,

- an ipmpinfo:LicenseCollection element, a collection of licenses each of which identifies its own target, and/or

- dsig:Signature element.

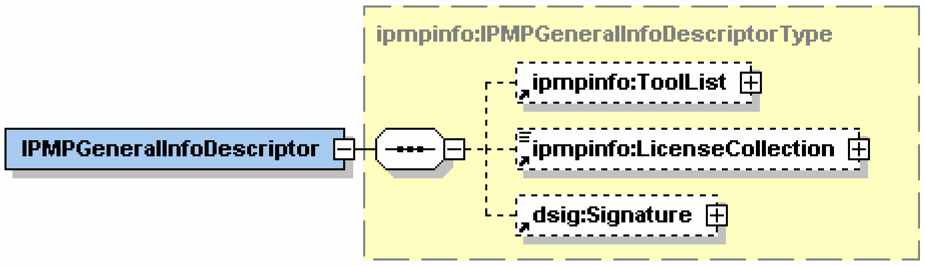

ipmpinfo:IPMPGeneralInfoDescriptorType
IPMPGeneralInfoDescriptor
ipmpinfo:ToolList
ipmpinfo:LicenseCollection
dsig:Signature

Figure 3-8 Structure of an IPMPGeneralInfoDescriptor element [6]

Each element in the IPMPGeneralDescriptor element is used to communicate general information relating to a complete Digital Item.

3.2.3 Processing IPMP DIDL Elements

In this section, we illustrate the ways to process the IPMP DIDL elements. We introduce the encapsulation of the DIDL element and the retrieval of the DIDL element from an IPMP DIDL element in the following paragraphs.

Figure 3-9 shows a block diagram with two inputs (DIDL element and IPMP Information) and single output (IPMP DIDL element). The IPMP DIDL element are processed according to the IPMP Information defined in the ipmp:Info child element. This system can encapsulate the DIDL element with/without encryption in an IPMP DIDL element. In Figure 3-10, a music album is processed by encapsulation in an ipmp:Contents child element of the IPMP DIDL element.

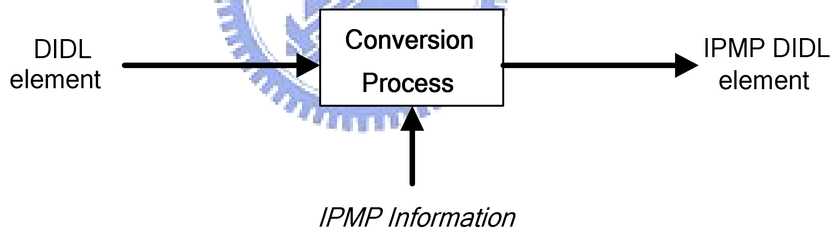


Figure 3-9 IPMP DIDL processing [6]

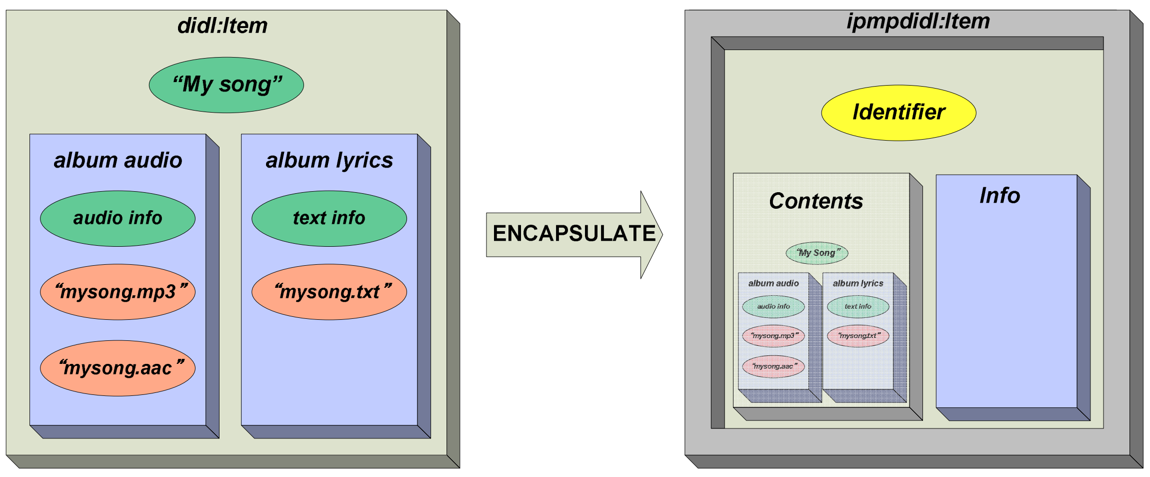


Figure 3-10 IPMP DIDL processing example [6]

A DIDL element, as shown in Figure 3-11, is encapsulated in IPMP DIDL by applying the following procedure.

1) Replace the DIDL element with the corresponding IPMP DIDL element. In this example, the DIDL Item element is replaced by an IPMP DIDL ipmpdidl:Item element.
2) Encapsulate the identifier in an IPMP DIDL Identifier element if the DIDL element has an identifier. In this example, the DII Identifier of the DIDL Item is encapsulated in an IPMP DIDL ipmpdidl:Identifier element.
3) If any information is required to retrieve the DIDL element content from the IPMP DIDL Contents element, encapsulate it in an IPMP DIDL Info element. In this example, the content of the DIDL element is simply encapsulated in an IPMP DIDL Contents element at this stage.
4) Finally, encapsulate the content of the DIDL element in an IPMP DIDL Contents element.

After encapsulation, an IPMP DIDL element which encapsulates the unprotected DIDL element is shown in Figure 3-12.

```
<Item>
     <Descriptor>
          <Statement mimeType="text/plain">
               <dii:Identifier>IPMPid0001</dii:Identifier>
          </Statement>
     </Descriptor>
     <Component>
          <Resource ref="myPicture.png" mimeType="image/png"/>
     </Component>
</Item>
```

Figure 3-11 Example of an unprotected DIDL structure [6]

```
<ipmpdidl:Item>
     <ipmpdidl:Identifier>
          <dii:Identifier>IPMPid0001</dii:Identifier>
     </ipmpdidl:Identifier>
     <ipmpdidl:Info>...</ipmpdidl:Info>
     <ipmpdidl:Contents>
          <Item>
               <Descriptor>
                    <Statement mimeType="text/plain">
                         <dii:Identifier>IPMPid0001</dii:Identifier>
                    </Statement>
               </Descriptor>
               <Component>
                    <Resource ref="myPicture.png" mimeType="image/png"/>
               </Component>
          </Item>
     </ipmpdidl:Contents>
</ipmpdidl:Item>
```

Figure 3-12 Example of an IPMP DIDL element which encapsulates the unprotected DIDL

element [6]

However, if the DIDL element is valuable, the ipmpdidl:Contents can be encrypted with a tool described in ipmpdidl:Info to prevent from unauthorized access. The tool information would be encapsulated in the IPMP DIDL Info element at step 3) above, and the encryption would occur during step 4) above. In this case the structure and contents of ipmpdidl:Contents will not be visible. Figure 3-13 shows an IPMP DIDL element which encapsulates the protected DIDL element.

```
<ipmpdidl:Item>
     <ipmpdidl:Identifier>
          <dii:Identifier>IPMPid0001</dii:Identifier>
     </ipmpdidl:Identifier>
     <ipmpdidl:Info>...</ipmpdidl:Info>
     <ipmpdidl:Contents>3E674F632A56BD56...</ipmpdidl:Contents>
</ipmpdidl:Item>
```

Figure 3-13 Example of an IPMP DIDL element which encapsulates the protected DIDL element [6]

After receiving the IPMP DIDL element, the following is used to retrieve the original content.

1) Parse the tool information encapsulated in the IPMP DIDL Info element.
2) Obtain an instance of the tool and configure the tool as required.
3) Apply the tool to the content of the IPMP DIDL Contents element. The result should be the original content of the encapsulated DIDL element.

Replace the IPMP DIDL element with the corresponding DIDL element with the content obtained from step 3) above.

3.2.4 Relationship between IPMP and other parts of MPEG-21

The fundamental unit of MPEG-21 Multimedia Framework is the Digital Item. While the existing parts of ISO/IEC 21000 deal with different aspects of the Digital Item, they together facilitate to construct the complete MPEG-21 Multimedia Framework. It is important to understand the relationship between the parts to be able to achieve an interoperability framework. The relationship is described below: [20]

A. Relationship between IPMP Components and MPEG-21 Part 2 Digital Item

Declaration (DID)

- Because a DI expressed in DIDL is a cleartext, the document of a DI represented entirely in DIDL (the representation for Digital Items) is unprotected. IPMP Components provide *an alternative representation* for parts of Digital Items that require protection. This representation is termed the IPMP Digital Item Declaration Language (IPMP DIDL). Each of these IPMP DIDL elements is identical to link a corresponding DIDL element (which may be encrypted) with information about the governance, to that the Digital Item (or part of if) is used in accordance with the Digital Item author's wishes.

B. Relationship between IPMP Components and MPEG-21 Part 3 Digital Item Identification (DII)

- The Digital Item identifiers associated with Digital Items, and parts thereof can be placed in a specific place within the Digital Item structure. When the IPMP DIDL is used for protecting a DI, it may hide or prevent access to identifiers located within that hierarchy. However, some information in DIDL can not be hidden (for example, without a REL License that references the governed content, the verification can not be executed).

C. Relationship between IPMP Components and MPEG-21 Part 5 Rights Expression Language (REL)

- REL specifies the syntax and semantics of a Rights Expression Language expressing the rights which a user may have to act on assets, such as Digital Items or parts thereof. One important concept in REL is a License. A License is defined as an expression that is created by Principals to conditionally or unconditionally permit the same or other Principals to perform Rights upon resources.
- IPMP defines four different ways about how REL can be correctly associated with their target:
  1) Be included in a DI
  2) Be referenced from within a DI
  3) Be referenced from within a DI via a license service
  4) Reference the DI from the rights expression

Chapter 4

MPEG-4 IPMPX Framework on MPEG-21 Testbed

In this chapter, we introduce that a content protection framework, MPEG-4 IPMP Extension (IPMPX) [24], can be integrated into a multimedia streaming platform, MPEG-21 Test Bed [8]. Thus, this multimedia streaming platform has the ability to prevent digital content from unauthorized access.

4.1 MPEG-4 IPMPX Extension

The first version of the MPEG-4 IPMP system [24] is called IPMP Hook. To enhance the inter-operability, the second version of the MPEG-4 IPMP system is defined as IPMPX [24]. The MPEG-4 IPMPX system was finalized in 2004. In the 62nd MPEG meeting in October 2002, the MPEG-4 IPMPX system was declared to become a new part, part 13, of MPEG-4 standard [21]. The IPMP Hook is out-dated by the IPMPX and thus we consider the IPMPX in rest of this thesis.

Although, the MPEG-4 IPMPX system is designed to safeguard the intellectual property in the form of MPEG-4, it can provide not only the MPEG-4 terminal but also other platforms the management and protection of intellectual property.

An MPEG-4 IPMPX system is a message-based system. The standard specifies the interface and protocol for these messages. The communication among Tools and Terminal becomes easier. By the proper use of messages, different Terminals can authenticate each other. The whole structure of MPEG-4 IPMPX is shown in Figure 4-1. The basic concept of MPEG-4 IPMP system is a Virtual Terminal, which contains a Tool Manager (TM) [24] and a Message Router (MR) [24]. The Message Router is to pass the message to the corresponding receiver. All IPMP Tool Messages are routed through this entity. The Tool Manager is used to manage and connect all the IPMP Tools to perform the specific IPMPX function. We will describe the details later.

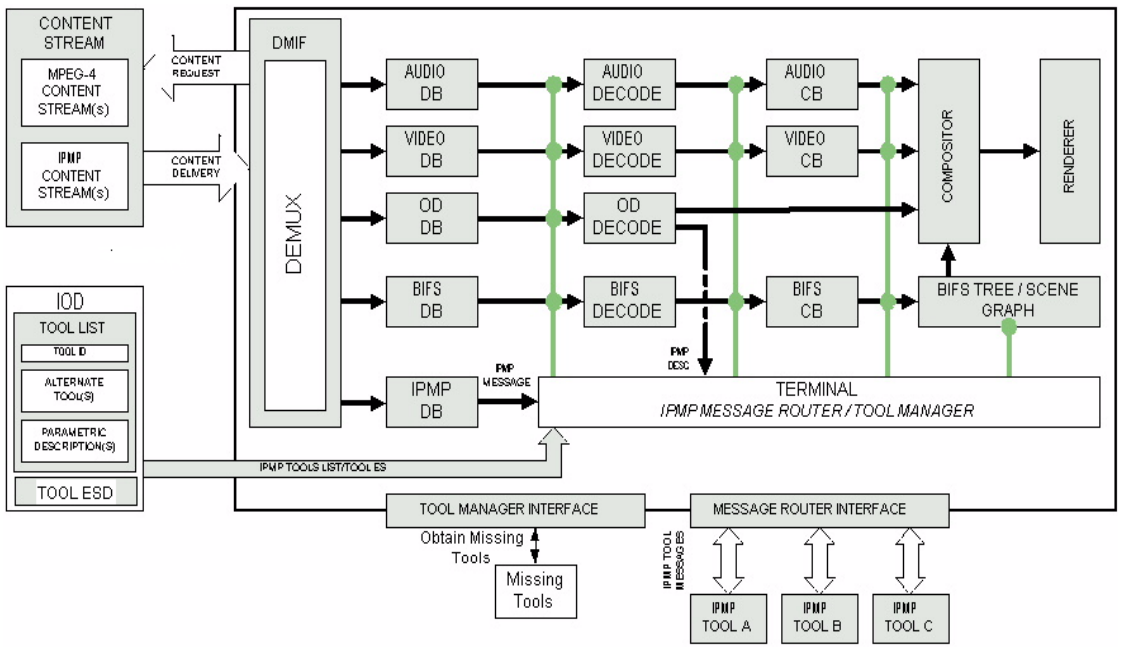


Figure 4-1 Architecture Diagram of MPEG-4 IPMPX system [24]

4.1.1 IPMPToolManager

The Tool Manager [24] mainly has three jobs. First of all, it processes an IPMP Tool List Descriptor. The IPMP Tool List Descriptor is contained in the IOD (Initial Object Descriptor) and is passed to the Tool Manager when this IOD is parsed by the Terminal. After the IPMP Tool List Descriptor is processed by the Tool Manager, the listed IPMP Tools are resolved.

In addition to resolve the listed IPMP Tools, the IPMP Tool Manager also attaches these listed IPMP Tools to corresponding IPMP Filters when the Message Router requests the Tool Manager to connect a tool. This action requires the associated IPMP Tool Descriptor for initial configuration including the specific location for the Tool to connect to. However, the removal of a tool is initiated by the Message Router and requires only the pointer of the tool. [22]

4.1.2 IPMP Message Router

The IPMP Message Router [24] performs the message routing. The locally delivered messages are called IPMP Tool Message, and the messages delivered among devices are called the IPMP devices Message (Annex. H). In addition to routing the IPMP messages, the IPMP Message Router should also process the information dispatched from the Terminal. The information may come from the IOD or the bitstream. When the terminal receives an IOD, it parses the IOD and then passes the IPMP Tool Descriptor, the ES Descriptor, and the IPMP Tool Descriptor Pointer to the Message Router for processing.

The IPMP Message Router processes the IPMP Tool Descriptor that contains some information for IPMP Tool initialization. The IPMP Message Router also processes the ES (Elementary Stream) Descriptor for parsing the IPMP Tool Descriptor Pointer. An IPMP Tool Descriptor Pointer indicates the existence of an IPMP Tool in its residing ES by the descriptor ID field in its associated IPMP Tool Descriptor. The location of an IPMP Tool is specified by the control point and the sequence code of the IPMP Tool Descriptor. After processing the IPMP Tool Descriptor Pointer, the IPMP Message Router informs the IPMP Tool Manager to connect the IPMP Tools to specific location specified by the IPMP Tool Descriptor. Also, the IPMP Message Router handles the IPMP elementary stream, which contains the dynamic IPMP information for updating the IPMP system.

4.1.3 Terminal

The Terminal [24] is an environment where the IPMP system performs its functionalities. For example, it may be an IM1 [25] terminal or any compatible terminals that could perform content consumption 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. It also receives the IPMP Tool List from bitstream and dispatches this to the IPMP Tool Manager. In addition to the above IPMP-related work, there are still some works done by the Terminal such as decoding the bitstream data, displaying, etc.

4.1.4 IPMP Tool

The IPMP Tool [24] is the component that really performs the functionalities of IPMP, such as encryption, decryption, watermark insertion, watermark extraction, authentication, etc. The IPMP Tool may be instantiated inside the IPMP Control Point to perform the data processing or may be instantiated outside all the IPMP Control Points to perform other functionalities that are not related to the data stream, 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 received bitstream. When an IPMP Tool receives the IPMP message, the process of the message is implementation dependant and is not defined in the specification.

4.1.5 IPMP Control Point (IPMP Filter)

IPMP Control Points [24] are the place where an IPMP Tool can connect for data processing. IPMP Control Points are just like the Filters with one or more IPMP Tools plugged inside. For example, there are one IPMP Control Point between the decoder and decoding buffer, and one between the decoder and the composition buffer. The IPMP Tool that decrypts the stream data will be inserted into the IPMP Control Point between the decoder and decoding buffer; and, the IPMP Tool that extracts the watermark will be plugged into the IPMP Control Point between the decoder and composition buffer. In the MPEG-4 IPMP Extension specification, there are four IPMP Control Points defined: the two described above, the one between the composition buffer and the compositor, and the one on the the BIFS tree. The specifications also reserve several user-defined position codes for future extension of control points.

4.1.6 IPMP Control Information

The IPMP control information may be contained in the Initial Object Descriptor (IOD) [24], Object Descriptor update command [24], or IPMP elementary stream [24]. It specifies how the IPMP system works. The IPMP control information is composed of IPMP Tool List Descriptor [24], IPMP Tool Descriptor [24], and IPMP Tool Descriptor Pointer [24].

- Initial Object Descriptor:
  IOD contains the three important components, Tool List Descriptor, Tool Descriptor, and Tool Descriptor Pointer. There can be multiple Tool Descriptors and multiple Tool Descriptor Pointers, but only one Tool List Descriptor within an IOD. In the beginning of the terminal, an IOD must be accessed for initial configuration of the IPMP system. Those configurations we considered are only about Tool List Descriptor, Tool Descriptor, and Tool Descriptor Pointer.
- IPMP Tool List Descriptor:
  The IPMP Tool List Descriptor is to transport the list of required IPMP Tools for content consumption. An IPMP Tool List can contain 256 IPMP Tools at most. A Tool is specified

in the IPMP Tool List Descriptor using one of the three modes:

- The Unique mode: each IPMP Tool is described by a unique ID (IPMP_ToolID).
- The Alternates mode: Terminal can choose one of the alternate IPMP Tools specified in the set of ToolIDs.
- The Parametric mode: it is used when an IPMP Tool has the following features.
    1. It is based on 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.

- 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 configuration. If the IPMP_ToolID in this descriptor is zero, the remote IPMP Tool Descriptor is pointed by the URLString. Otherwise, the IPMP_Initailize structure carries the IPMP information for initializing the Tool. The location where the IPMP Tool is connected is specified by the controlPointCode and sequenceCode in the IPMP_Initailize. Here the controlPointCode specified in the specification is shown in the following table [24].

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 IPMP Control Point, the order of invocation is decided by the sequenceCode. An IPMP Tool with larger sequenceCode has higher priority to process the datat.

- IPMP Tool Descriptor Pointer:

The IPMP Tool Descriptor Pointer is a pointer that points to an IPMP Tool Descriptor by specifying an IPMP Tool Descriptor ID within it. The IPMP Tool Descriptor Pointer in an object descriptor indicates that all streams referred to by the embedded elementary descriptor are subject to be protected and managed by the IPMP Tool specified in the pointed IPMP Tool Descriptor. Similarly, 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 be protected and managed by the IPMP Tool specified in the referenced IPMP Tool Descriptor. Every IPMP Tool Descriptor pointer instantiates an unique instance of the corresponding IPMP Tool.

Figure 4-2 shows how an IPMP Tool is resolved according to the Tool Descriptor Pointer. The IPMP Tool Descriptor Pointer within the OD A containing an ESD C indicates that there is one IPMP Tool with descriptor F at the AUDIO ES. A similar resolving process can applied to OD B. There are many cases of use and scope of the IPMP Descriptors and declaration. The details are illustrated in the Annex F of the MPEG-4 IPMP Extension specification [24].

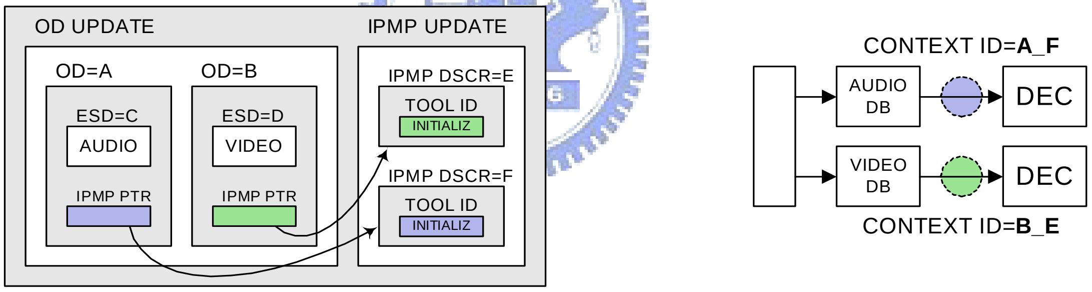


Figure 4-2 Sample of IPMP Tool context ID mapping [24]

4.1.7 Using IPMP

Here, we give a simple example that illustrates how an IPMP system works. Figure 4-3 shows the basic concept of MPEG-4 IPMPX system. We describe the scenario of content consumption step by step as follows. [23]

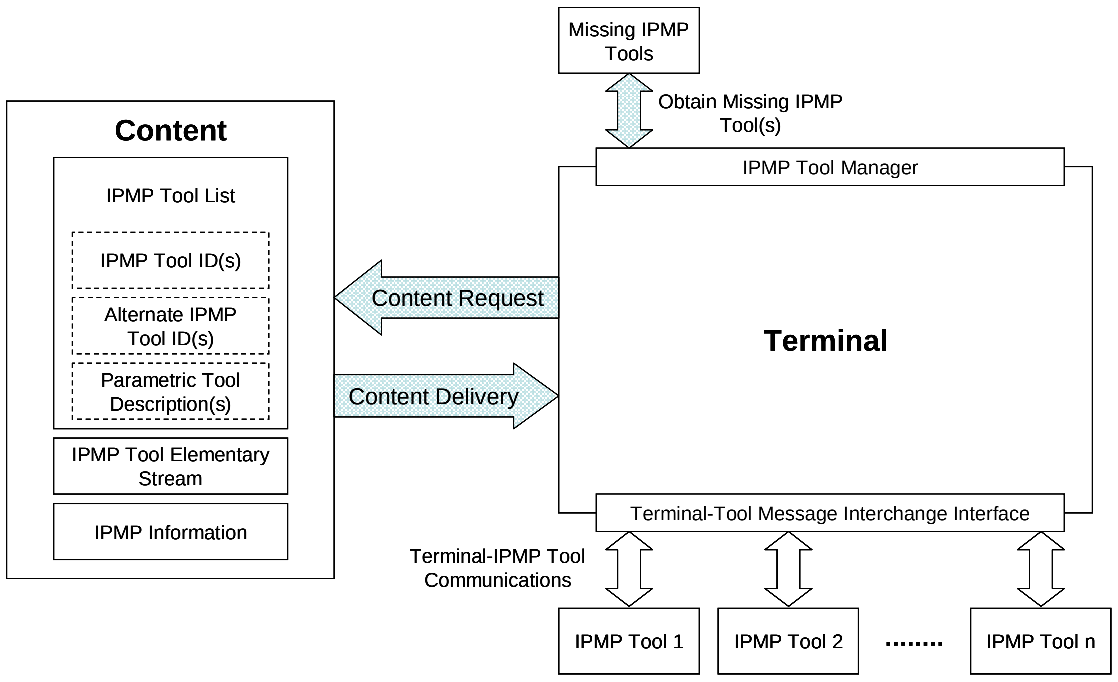

Missing IPMP Tools
Obtain Missing IPMP Tool(s)
Content
IPMP Tool List
IPMP Tool ID(s)
Alternate IPMP Tool ID(s)
Parametric Tool Description(s)
IPMP Tool Elementary Stream
IPMP Information
Content Request
Content Delivery
IPMP Tool Manager
Terminal
Terminal-Tool Message Interchange Interface
Terminal-IPMP Tool Communications
IPMP Tool 1
IPMP Tool 2
....
IPMP Tool n

Figure 4-3 MPEG-4 IPMPX basic concept [24]

1. **User request specific content**

   The standard does not specify that how the content is requested. However, the following recommendations are made for the processing priority of each part of the Content. First, the IPMP requirement should be placed with or before media requirement. Second, before delivery of the media content, IPMP information and/or restrictions should have been accessed. To ensure the content is protected as specified, it is reasonable that the terminal should access and own the information that protects the media content before accessing the media data. [23]

2. **IPMP Tool Descriptor access**

   Before the content consumption, the terminal should parse the Initial Object Descriptor and retrieve the IPMP Tool List Descriptor within the IOD to get the list of the required IPMP Tools. The IPMP Tool Descriptor is also conveyed within the IOD, and should be obtained by the terminal here. [23]

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. [23]

4. **Instantiation of IPMP Tools**

   The IPMP Tools required to consume the Content are instantiated locally according to the IPMP Tool List Descriptor received before. The IPMP Tool Descriptor contains the IPMPInitialize information that provides the IPMP Control Point code and the sequence code to inform the Terminal to connect the IPMP Tool at the specified position. [23]

5. **Initialize and update the IPMP system**

   After setting up the whole IPMP system according to the Initial Object Descriptor, the content consumption begins. During consuming the content, 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. There are also IPMP messages for local negotiation. All the steps can be requested more than once during the content consumption, and the requests may originate from the process implicitly or from the user explicitly. [23]

4.2 Overview of MPEG-21 Testbed

The purpose of the MPEG-21 Testbed is to provide a flexible and fair test environment for evaluating delivery technologies for MPEG contents over IP network. It also supports scalable media streaming, which is a crucial topic in MPEG-21 Digital Item Adaptation (DIA). Because each component in this test bed is highly modulized, users can integrate their customized components into this test bed easily. For example, they can replace the media codec to evaluate 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 [9].

The overall architecture of the test bed is shown in Figure 4-4. The entire test bed is divided into three parts: Server, Client, and the Network Emulator.

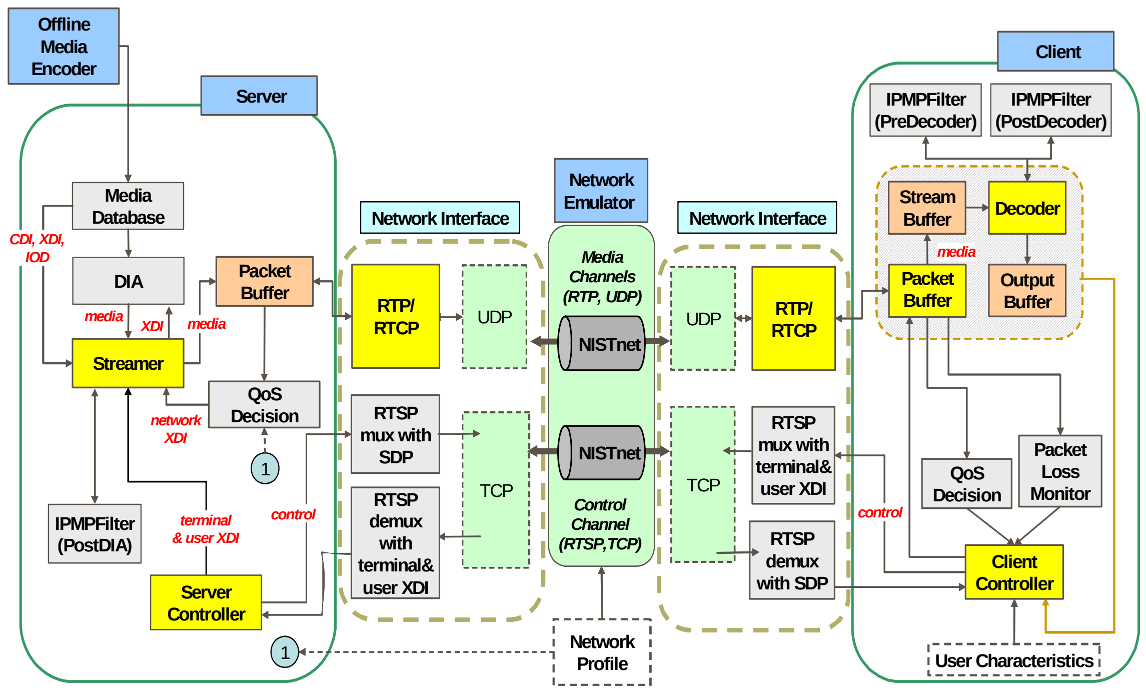


Figure 4-4 Architecture of MPEG-21 Testbed [8]

The Server system accepts the request for a specific media content form the Client. A Client is 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 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. The server sends an acknowledging message called DESCRIBE_ACK to Client indicating that the media data at the Server side is ready. An RTSP message, SETUP, is then received form the Client to request for setting up the transport session. After setting up the RTP channel for content delivery, the Server acknowledged the Client with a message called SETUP_ACK. After 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. 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 [9].

Because there are several common modules in Server and Client, the descriptions below are divided into four parts, Server components, Client components, Common components, and Network Emulator.

4.2.1 Server Components

The functionality of the server is to provide digital media content with streaming technology 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 the QoS Decision will be discussed later.

Media Database

All the Media Contents are offline encoded and stored in the Media Database using a directory tree structure. The Media Database is responsible for opening the file of the Media Content that the Client requests. [9]

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 are required for initialization, and the dynamic XDI will be set during adaptation. The DIA component receives the information from the Server Controller and generates the adapted resource. [9]

Streamer

When a specific Media Content is requested by the Client, the Streamer gets this content from the DIA module. The Streamer accepts commands from the Server Controller, and segments this content into video packets according to the MPEG-4 specification. [9]

Server Controller

Server Controller handles the control messages sent by the Client side through the RTSP channel, such as REQUEST, SETUP, PLAY, and TEARDOWN. After receiving these messages, the Server Controller processes these messages and sets up the system for processing different protocols. The Server Controller also sends the corresponding ACK messages to the Client side. [9]

4.2.2 Client Components

The Client consumes the digital media content coming from the Server. It receives the media data through the Network Interface and playbacks the content including video and audio. 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 Stream Buffer is designed as a circular buffer which temporally stores the bitstream data. The data saved in the Stream Buffer will be accessed by the decoder for decoding. There is a flag that records the current access location in the Stream Buffer. When there is no data in the Stream Buffer, the Decoder waits for the Stream Buffer to get the data from the Packet Buffer and resumes decoding. [9]

Decoder

The Decoder component is the root class for all decoders, including real-time and offline decoders. A Decoder fetches coded units from the Stream Buffer and decodes the data. After decoding, Decoder stores the decoded units into the Output Buffer for display. [9]

Output Buffer

The Output Buffer component holds the decoded data produced by the Decoder. The data will be fetched by the player and returned on the output device. [9]

Packet Loss Monitor

The Packet Loss Monitor handles the lost packet monitoring and retransmission.

The component checks the Packet Buffer for any lost packets. The action is triggered by the Client-side timer. If there are packet losses, Packet Loss Monitor informs the Client Controller to issue the retransmission request to the Server side. [9]

Client Controller

The Client Controller integrates and controls all the components at the Client side. It processes the GUI user's inputs. When there are lost packets detected by the Packet Loss Monitor, the Client Controller is informed to send the retransmission request to the Server side. [9]

4.2.3 Common Components

In this section, we introduce the modules that are common in 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 Server side, the Packet Buffer holds the video packets produced by the Streamer. The stored packets are then sent to the Client side via RTP protocol according to the pre-scheduled time set by the Streamer. At the Client side, the Packet Buffer receives and stores all the packets that coming from the RTP protocol. This component is used at both Server and Client side with different initializations. [9]

QoS Decision

This component estimates the channel condition and provides some QoS information for rate adaptation. In the current implementation, 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. [9]

IPMP Subsystem

The IPMP Subsystem performs the functionalities of intellectual property protection, such as encryption, decryption, and watermarking. The IPMP Subsystem is mainly composed of five parts, namely 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 routing the messages to the 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. It is also responsible to initialize and destroy the IPMP Tools. There can be several IPMP Tools working within one IPMP system. For instance, at Client side, there would be a DES decryption algorithm, a video watermarking extractor, and an authentication tool. The IPMP Filter is the control point where the IPMP Tools can exercise their functions. The Terminal is provided as the interface between the test bed system and the IPMP system. The details about the IPMP system in the test bed will be discussed in section 4.3.

4.2.4 Network

In order to connect the Server and the Client via 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 shown in Figure 4-5.

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

Figure 4-5 Network protocol [9]

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: DESCRUBE, SETUP, PLAY, and TEARDOWN. The DESCRIBE message is sent from Client to Server for requesting a specific media content. It also contains the information such as terminal capability or user characteristics. 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 of starting to transport the media content. Finally, the TEARDOWN message ends the transport and closes the session.

In the test bed, an IP network emulator named NISTnet [29] is used to provide repeatable network environments. NISTnet can simulate practical wide-area heterogeneous network environments. It is a LINUX based IP network emulator developed by the National Institute of Standard Technology, USA. The testbed provides a GUI to parse a network profile and controls the time-varying network conditions via the NIST Net kernel module.

4.3 Integrate MPEG-4 IPMP Extension into MPEG-21 Testbed

To integrate the MPEG-4 IPMPX subsystem into MPEG-21 Test Bed, some additional design and implementation is required. The major components are Context, IPMP Tool, IPMP Filter and Terminal. All details of these modifications are introduced in the following subsections.

4.3.1 Context

Because there is the information shared among the IPMP modules, we put the shared information in the Context structure rather than in individual modules. All the IPMP modules, including Tool Manager, Message Router, Terminal, IPMP Filters, and IPMP Tools, are associated to the Context in order to retrieve the shared information. The relationships among these modules and the Context are shown in Figure 4-6. [22]

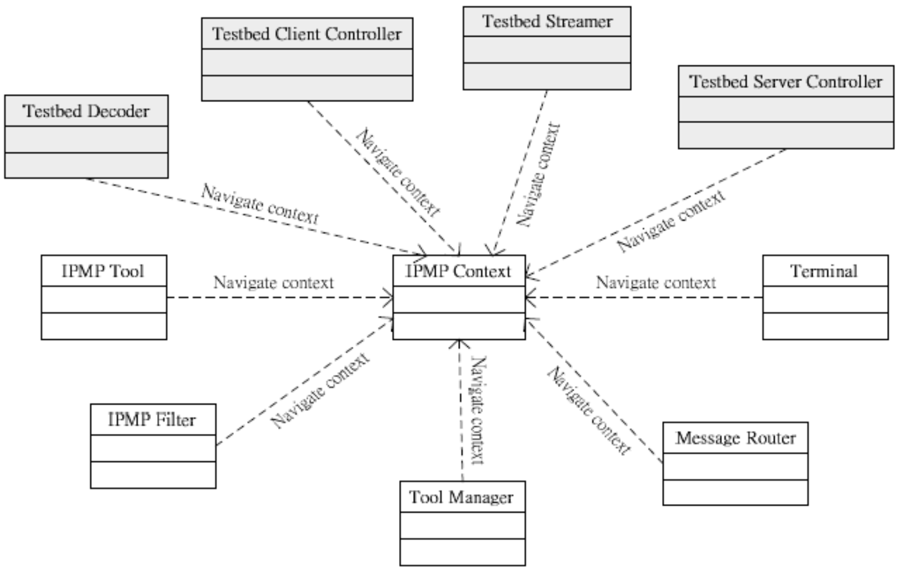


Figure 4-6 Relationship between IPMP Context and other modules [22]

4.3.2 IPMP Tool

In order to make the design and implementation of the IPMP Tools flexible, the IPMP Tools are designed to have uniform interfaces. They are enforced to support a few (minimal) operations required by the architecture. For example, an IPMP Tool should be able to receive and to process IPMP messages. Though this operation is implementation dependent, all tools should be able to perform this operation. Figure 4-7 shows the relationships between the IPMP Tool and the other IPMP modules. [22]

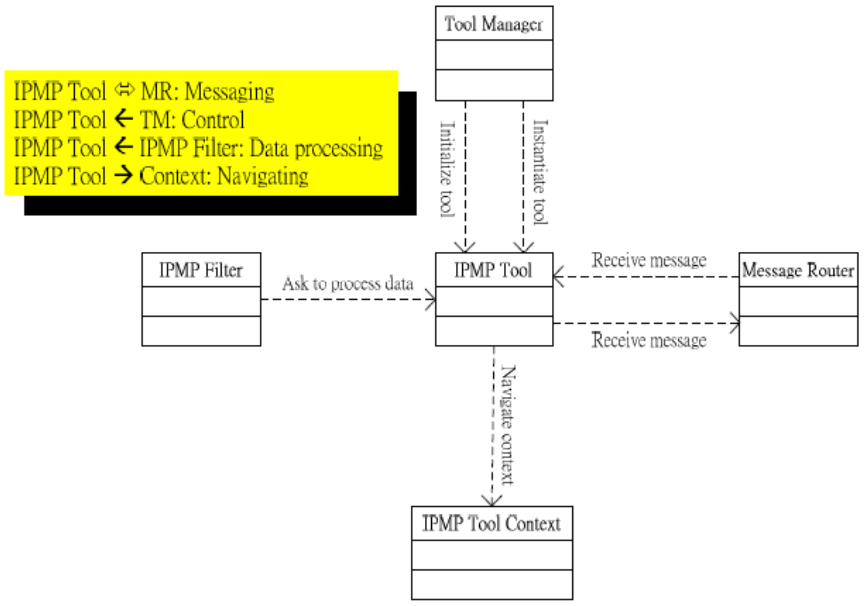


Figure 4-7 Relationship between IPMP Tools [22]

4.3.3 IPMP Filter

Due to the design of the MPEG-21 Testbed, there are only three IPMPX Control Points available on the Test Bed: two on the client side and one on the server side. On the client side, the Control Point located before the Decoder (PreDecoderFilter) corresponds to the Control Point between the Data Buffer and the Decoder; and the CP between the Decoder and the Output Buffer (PostDecoderFilter) corresponds to the one between the Decoder and the composition buffer.

Since the specification of the MPEG-4 IPMPX mainly focuses on the client side only, our design on the server side is inferred from the client side design. The PostDIAFilter is introduced as the counterpart of the PreDecoderFilter (at the client side). Because the MPEG-21 Test Bed requires compressed media files as its sources, we do not have a server counterpart of the PostDecoderFilter. In this system, we can insert an IPMP Tool for decryption in PreDecoderFilter at client side, and insert an IPMP Tool for real-time encryption in the PostDIAFilter at the server side. Figure 4-8 summarizes how the IPMP Filters are related to the other Test Bed components. [22]

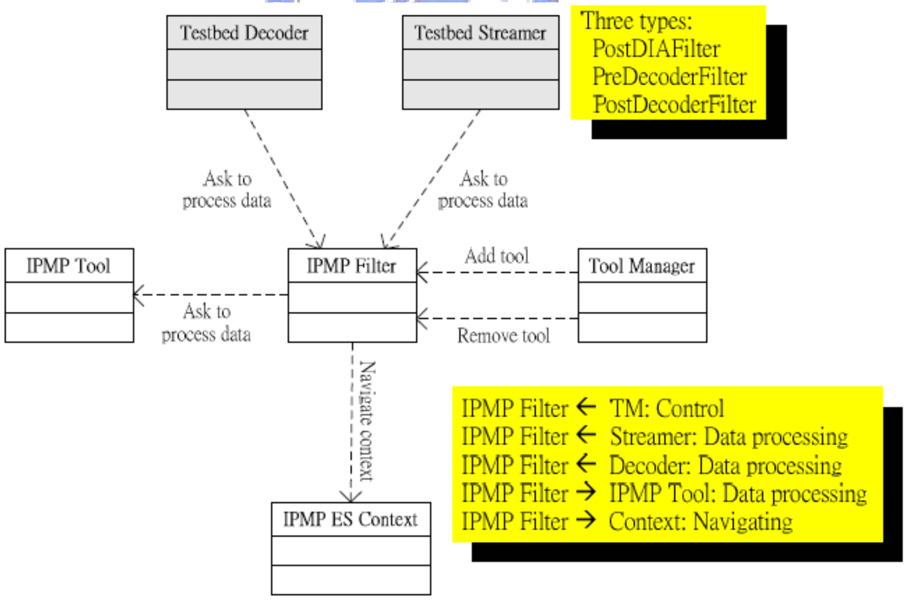


Figure 4-8 Relationship between IPMP Filter and other modules [22]

4.3.4 Terminal

The Terminal is the Interface between the Testbed system and the IPMP system. The relationship between them is shown in Figure 4-9. The Terminal class here can be viewed as the abstraction of the Test Bed system without the IPMP subsystem. The interaction between them includes local messaging and remote messaging.

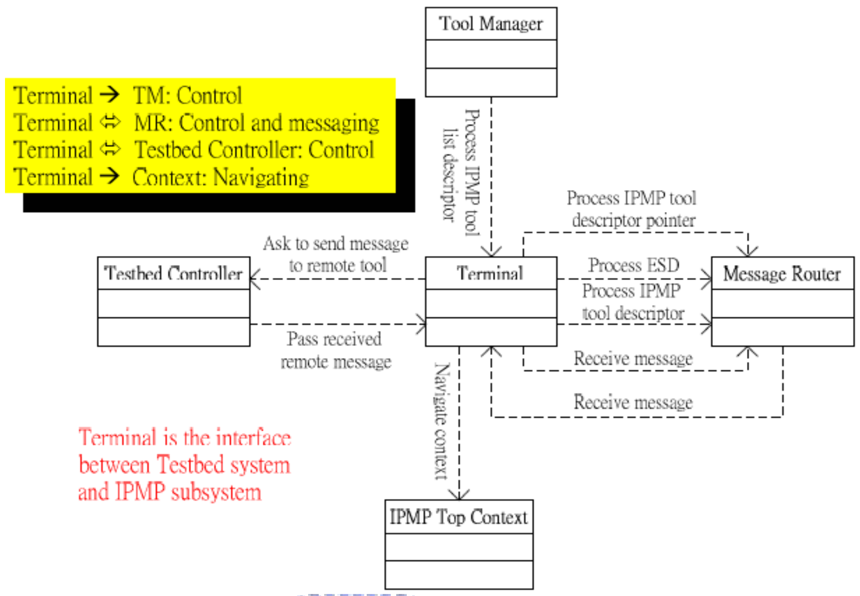

Terminal → TM: Control
Terminal ⇔ MR: Control and messaging
Terminal ⇔ Testbed Controller: Control
Terminal → Context: Navigating
Tool Manager
Process IPMP tool list descriptor
Process IPMP tool descriptor pointer
Ask to send message to remote tool
Testbed Controller
Terminal
Process ESD
Process IPMP tool descriptor
Message Router
Pass received remote message
Receive message
Receive message
Navigate context
Terminal is the interface between Testbed system and IPMP subsystem
IPMP Top Context

Figure 4-9 Relationship between Terminal and other modules [22]

Chapter 5

Implementation of MPEG-21 IPMP and REL with MPEG-4 IPMPX

This chapter describes our implementation of the MPEG-21 IPMP and REL incorporated with the MPEG-4 IPMPX Framework on the MPEG-21 Testbed. First, we modify the computer program of the MPEG-4 IPMPX Framework embedded on the MPEG-21 Testbed [8] to process the IPMP Messages sent from the local (or remote) terminal. Further, we redesign the procedure of the entire system in accordance with the specifications of the MPEG-21 IPMP and REL. Figure 5-1 shows the basic structure of the modified system, containing the server side and the client side. The server side encapsulates the digital resource into the protected format. The client side performs four IPMP related functions, which includes verifying the client, resolving the IPMP DIDL files to get the rights data, authorizing the client's request, and parsing the protected digital resource into the playback format if users are authorized to consume it. Our implementation will focus on how to verify that the user owns the valid licenses.

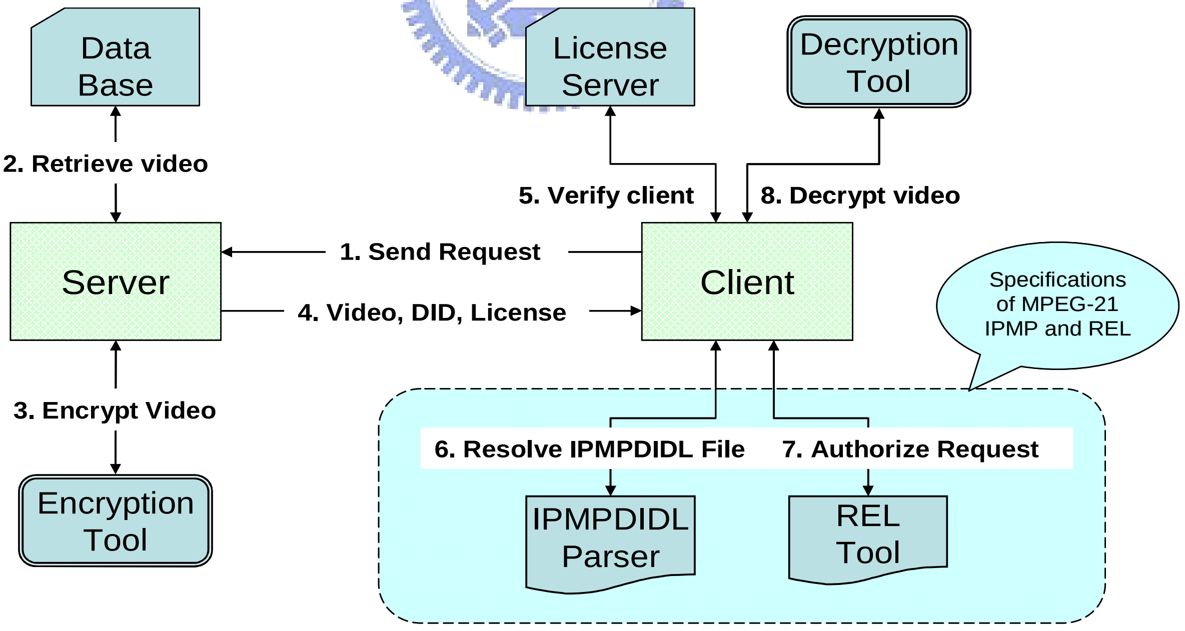


Figure 5-1 Basis structure of a modified content protection system

To construct a system which conforms to the basic structure shown in Figure 5-1, we choose the MPEG-21 Testbed as our streaming base system. Since the Testbed incorporates the IPMPX functionality, we design and implement the MPEG-21 IPMP using the APIs provided by the IPMP modules in the MPEG-21 Testbed.

In the following sections, we will describe first the design of IPMP Messages. Then, the details of redesigned overall system and the software implementation are discussed.

5.1 IPMP Messages

The IPMP Message is an important concept of the MPEG-4 IPMP Extension specification, because the MPEG-4 IPMP Extension is a message-based framework. All real-time information can be delivered to the IPMP Tools or Terminal via the IPMP Messages. All the IPMP Messages can be divided into two types, IPMP Tool Message and IPMP Device Message. In our system, communication between devices is represented as IPMP Tool Message. Hence, IPMP Device Message is not considered here.

The IPMP Tool Message facilitates the delivery of the IPMP Information among the IPMP Tools and the Terminal, and from the bitstream to the IPMP Tools. There are three forms of IPMP Tool Messages. *IPMP_MessageFromBitstream* is used to deliver IPMP stream data; *IPMP_DescriptorFromBitstream* is used to deliver IPMP_Descriptors; and *IPMP_MessageFromTool* is used to deliver messages to either other IPMP tools or the Terminal itself. All the three IPMP Tool Messages are derived from *IPMP_ToolMessageBase*. This message base contains three fields. The *Version (8 bits)* field indicates the version of the syntax. The *Sender (32 bits)* field indicates the context ID of the originator of the message. The *Recipient (32 bits)* field indicates the context ID of the intended recipient of the message. The context ID “0x00” is reserved for the terminal according to the MPEG-4 specifications.

In the MPEG-21 Testbed, the MPEG-4 IPMP terminal is split into the client and the server parts. Because each server links only to one client, we can treat the whole testbed system as an integrated terminal. Hence, to distinguish different side of terminals in the MPEG-21 Testbed, the server’s context ID is defined as “0x00” and the client’s context ID is defined as “0x80000000”. All the elements at the server side have the context ID less than “0x80000000”, and all the elements at the client side have the context ID greater than “0x80000000”.

The life cycle of an IPMP Message is divided into three stages as follows:

1) **Generation**: To create an IPMP Message, users have to fill in each element with

corresponding information, such as the context ID of the recipient and other IPMP Information.

2) **Transmission**: After the creation of an IPMP Message, the Message Router is called by the sender for transferring this message to the intended recipient.

3) **Accessing**: The intended recipient receives and parses this message. If necessary, a response message will be sent back to the originator of this received message.

According to the above description, only the details of the transmission stage are independent of applications. When the Message Router gets an IPMP message, it routes this message to the destination according to the recipient's context ID. Because the whole testbed system has two sides, an IPMP Tool Message may be routed to a local element or to a remote one. That is, if the recipient and the sender are at the same side, the message is routed locally; otherwise, the message is routed remotely.

Figure 5-2 illustrates the process of routing a message. The Message Router first checks the context ID of sender and receiver. If both of them are the same, the Message Router returns error. On the contrary, the Message Router decides the destination for passing the IPMP Message according to the context ID of the sender and the recipient of this message. If the context ID of the sender and the recipient are not identical in the most significant bit (MSB), the message will be sent through the network. If the context ID of the receiver is equal to the context ID of the local terminal, the terminal is called for receiving messages. If none of the conditions is satisfied, the destination is a local tool instance. The Message Router checks if the receiver exists. If the result is true, the message is received by the receiver; if the result is false, the Message Router returns error.

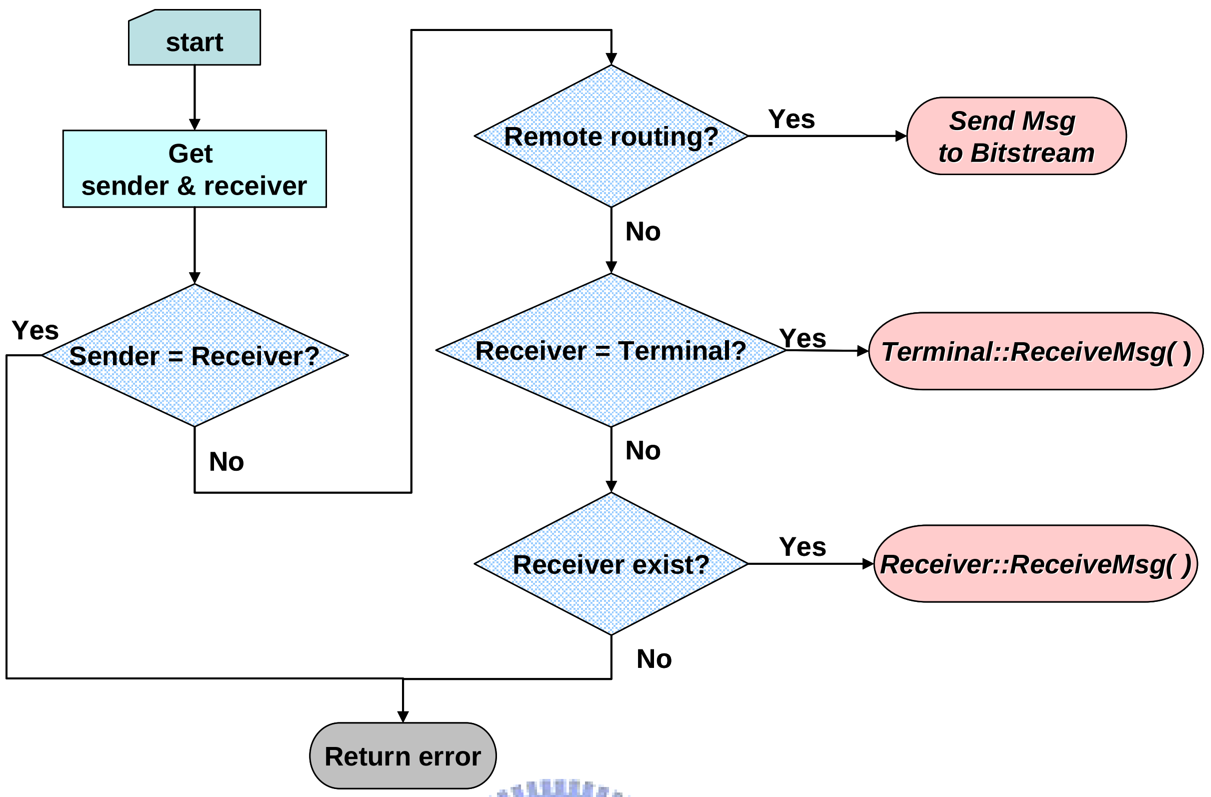


Figure 5-2 Decision of the Message Router for transmitting the IPMP Messages

5.1.1 Remote process

After the Message Router decides to transfer an IPMP Message to a remote terminal, this message will be sent to the remote terminal via the RTSP control channel in the Testbed. The flowchart of transferring an IPMP Message from a server tool to a client tool is given in Figure 5-3.

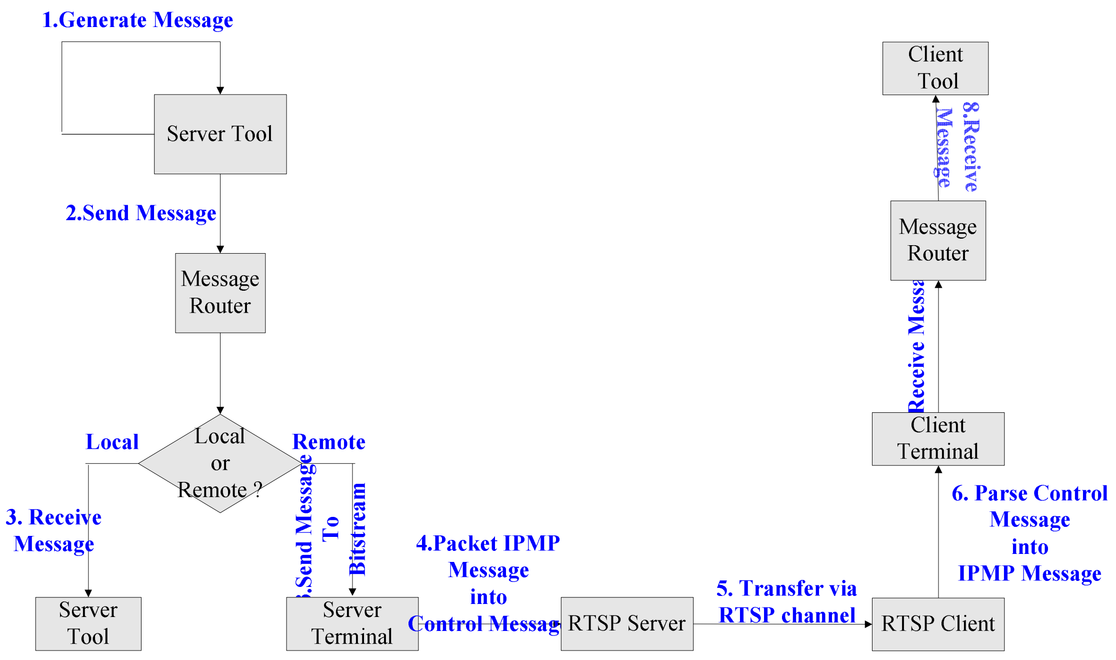


Figure 5-3 Flow chart of transfer an IPMP Message to remote terminal

In Figure 5-3, a tool at the server side wants to send an IPMP Message to a client tool. It generates an IPMP Message, *IPMP_MessageFromToolI,* and passes it to the Message Router. The Message Router receives the message and decides that this message should be transmitted to the client side.

```
class CIPMPMSG_MSG : public CCTRL_MSG{
    string m_message;//IPMP message
}
```

Figure 5-4 Structure of a RTSP control message, *CIPMPMSG_MSG*

Then, the Message Router calls the ServerTerminal’s API, SendMessageToBitstream(), to encapsulate the IPMP Message into the RTSP control message, *CIPMPMSG_MSG* (Figure 5-4). Because the control channel is designed to deliver ASCII data, the binary IPMP Tool Message is encoded using BASE64. The ServerTerminal calls the RTPServer to send the control message to the RTSP channel.

At the client side, the control channel is monitored by the client controller for receiving the control message which carries IPMP Messages. When the control message is received, the client controller calls the RTSP client to handle and parses the control message into the IPMP Messages. Then, the IPMP Message is passed to the client-side Message Router, and is sent to the intended recipient. Finally, the recipient receives and processes the IPMP Messages.

5.2 IPMP Tool design and software implementation

In this section, we discuss the design of our system. At the beginning, we describe the design and implementation of the IPMP Tools with the functionalities of the MPEG-21 IPMP and REL. Then, the designed implementation and the behavior are described.

5.2.1 IPMP Tool with MPEG-21 IPMP and REL functionalities

5.2.1.1 Behavior of MPEG-21 IPMP and REL

The main functionality of the MPEG-21 REL is to provide an authorization proof according to an authorization story and an authorization request file. Every application can use REL to allow or disallow users to consume a specific content under predefined conditions. The most important element in REL is the License.

The goal of the MPEG-21 IPMP is to provide the protection of Digital Items. The MPEG-21 Testbed supports two types of Digital Items, video and audio files. An effective way to protect them is encryption. This kind of protection can be effectively achieved by encryption tools. Because the IPMP information in an IPMP DIDL file can also be stored in an IOD file, only the IPMP information listed in an IOD file is processed in our implementation. In Figure 5-5, a parser is designed for analyzing the IPMPDIDL element.

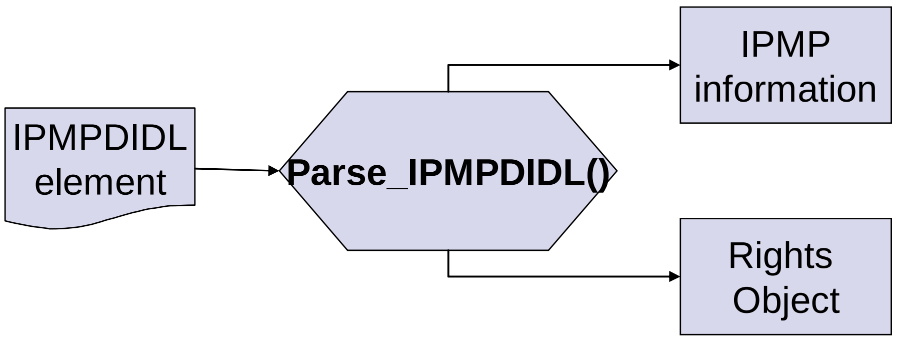


Figure 5-5 An API, Parse_IPMPDIDL()

Therefore, in our implementation, the MPEG-21 IPMP is treated as a central control element which stores all the related information and takes charge of accessing and processing it. For example, the license is reserved by the central control element which has the ability to perform the functionality of the MPEG-21 REL. Hence, we design an IPMP Tool called IPMP_Info_Engine to implement the above specifications.

5.2.1.2 MPEG-21 REL Reference Software

Our implementation of the MPEG-21 REL is based on the REL Reference software developed by Content Guard [30]. The MPEG-21 REL [2] was originally proposed by Content Guard. It now supports one right, “mx:play”, and two conditions, “exerciseLimit” and “validityInterval”.

The reference software contains three components: RELLicAuthzDriverGUI.exe, RELLicAuthzDriver.dll, and RELLicAuthz.dll. “RELLicAuthzDriverGUI” is the user interface for the application. Depending on the run-time environment, this component refers to Win PC or Pocket PC executable files. “RELLicAuthzDriver” is the driver for the MPEG-21 REL, which is responsible for parsing the input License and Query. “RELLicAuthz” contains modules for an MPEG-21 authorization application, to manage the process of authorizing Query according to the input License.

The dataflow of the reference software is shown in Figure 5-6.

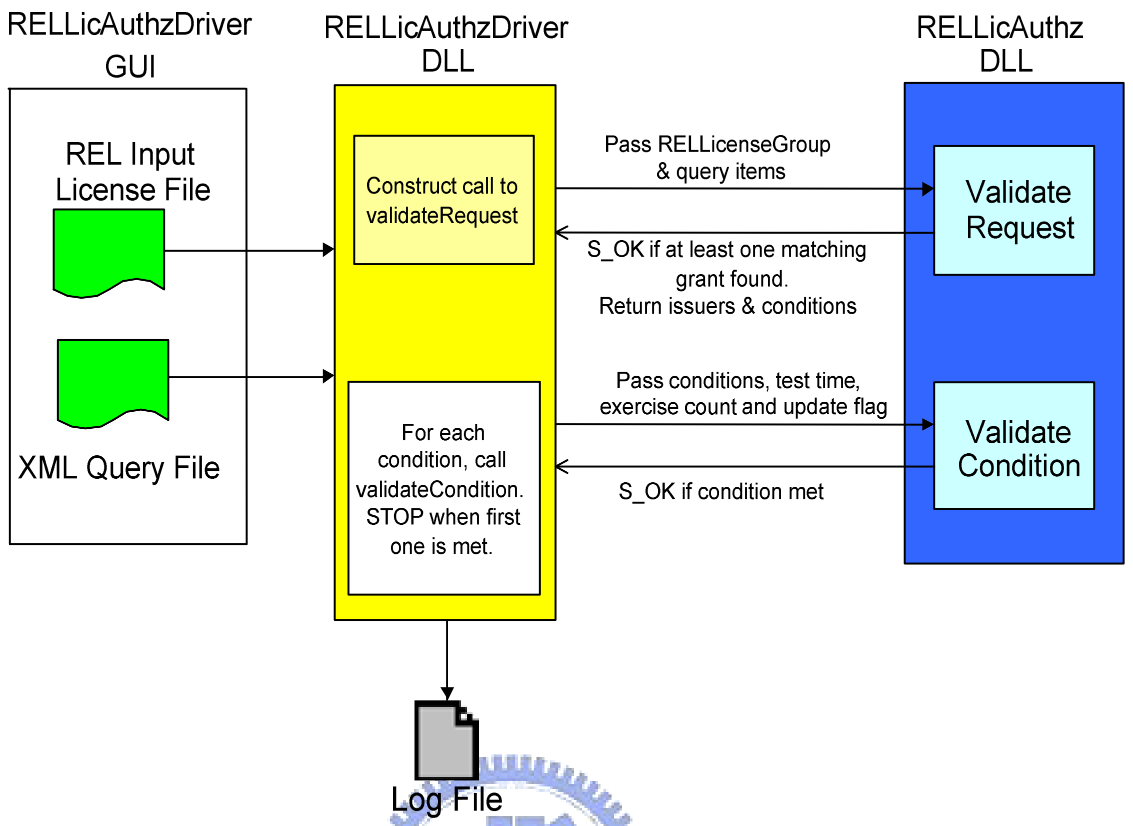

RELLicAuthzDriver GUI
REL Input License File
XML Query File
RELLicAuthzDriver DLL
Construct call to validateRequest
For each condition, call validateCondition. STOP when first one is met.
RELLicAuthz DLL
Validate Request
Validate Condition
Pass RELLicenseGroup & query items
S_OK if at least one matching grant found. Return issuers & conditions
Pass conditions, test time, exercise count and update flag
S_OK if condition met
Log File

Figure 5-6 Dataflow of the MPEG-21 REL Reference Software [30]

The two modules of the RELLicAuthz.dll are described below.

- The “validateRequest” module — It takes in an REL license file, checks to see if there are any matching grants against the supplied query items (keyHolder, right and resource). It then returns vectors of conditions for any conditions found in the matching grants, and vectors of issuers for the issuers of the matching grants. If there are no matching grants, it returns error messages.

- The “validateCondition” module — It checks to see if any of the conditions is met. Since this piece of software only supports the “validityInterval” and the “exerciseLimit” conditions, the input could be either the conditions or an allConditions element. According to the specified condition, other input fields could be a string containing an ISO format date (time) to check, an integer indicating the intended usage count, and a Boolean variable indicating whether to update the state of the usage count.

The execution flow in the REL reference software is described as follows.

1. The user selects an REL license file name and a query file name.
2. RELLicAuthzCEGUI calls the relAuthorizeRight function with the license file name and the query file name as the parameters.
   a) relAuthorizeRight loads the selected REL license file and the query file.
   b) relAuthorizeRight calls validateRequest by passing a licenseGroup, user, right, resource to it.
      I. validateRequest checks for unsupported items.
      II. If no unsupported items are found, validateRequest loops through all the grants in all licenses. If at least one matching grant is found, it will return a positive result. Any conditions found are also returned.
      III. If the call to validateRequest failed, the function RELAuthorizeRight exits.
3. If the call to validateRequest is successful, validateCondition will be called.
   a) validateCondition loops through all the conditions to check whether a condition is satisfied.
   b) If we meet at least one condition, it means that the specified user is allowed to exercise the right on the given resource.
4. The user is informed of the success/failure of the authorization.
5. The user can find further information in the log file, “AuthorizationSession.log”, at the directory which contains the application (if the “Log Authorization Session” option is selected).

5.2.2 Architecture of IPMP_Info_Engine Tool

To incorporate the REL into the Testbed, we integrate the REL’s reference software through a function of the IPMP_Info_Engine Tool, call_REL_authorize(). Because there is no proper IPMP Tool API to implement user interface of a tool, only the RELLicAuthzDriver and RELLicAuthz are included in this REL function. Two input arguments, REL input License and XML Query file, are assigned during initialization. The structure of the REL function is presented in Figure 5-7.

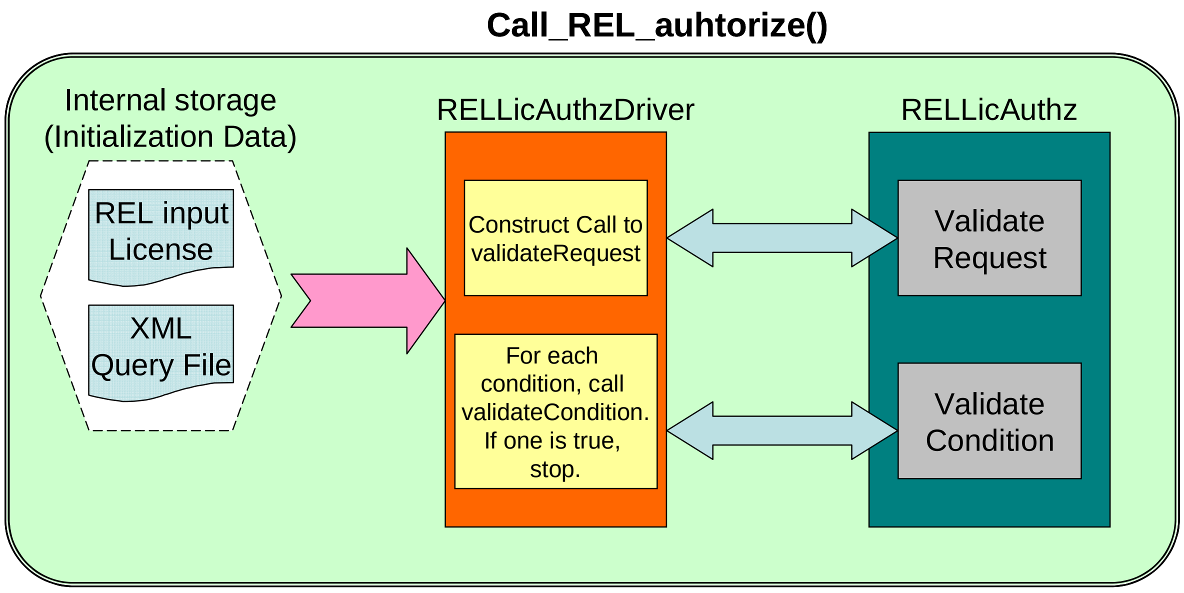

Call_REL_auhtorize()
Internal storage (Initialization Data)
REL input License
XML Query File
RELLicAuthzDriver
Construct Call to validateRequest
For each condition, call validateCondition. If one is true, stop.
RELLicAuthz
Validate Request
Validate Condition

Figure 5-7 Architecture of the REL function

After integrating the REL's reference software, the IPMP_Info_Tool has to retrieve the input License and XML Query File. In our implementation, the XML Query file is stored in the local device. In addition, the rights data (License) is encapsulated in an IPMP DIDL file. Because the initialization data of the IPMP_Info_Engine tool contains the IPMP DIDL file, we can get the License when the IPMP_Info_Engine tool is initialized with the tool descriptor.

The APIs associated with the functionalities of MPEG-21 IPMP and REL in IPMP_Info_Engine Tool are listed below:

Method:

int call_REL_authorize(void)

This method is called by the ReceiveMessage() for getting an authorization proof. The method returns zero when succeed.

void LoadQueryFileFromLocalFile(char *query_path)

This method loads the XML Query File. The parameter query_path indicates the file path.

void IPMPInfoEngine::Initialize(IPMPToolDescriptorD* init)

This method parses the IPMPToolDescriptor, and stores the rights data (License).

5.2.3 IPMP Filter (IPMP Control Point)

After describing the architecture of the IPMP_Info_Engine tool, we now discuss in which IPMP Filter the IPMP_Info_Engine tool should be placed for our implementation. In the MPEG-21 Testbed, three IPMP Filters are available. There are PostDIAFilter at server, PreDecoderfilter and PostDecoderFilter at client. These IPMP Filters stand for the IPMP Control Points.

The IPMP_Info_Engine tool is designed to give the user the consuming right of a specific content and to manage the IPMP DIDL files. It does not handle input video/audio data directly, but it provides the information about which right the user owns. Therefore, all IPMP Filters in the testbed are not suitable for the IPMP_Info_Engine tool. We do not place the IPMP_Info_Engine Tool in any IPMP Filters. In other words, the control point of the IPMP_Inof_Engine is CONTROL_POINT_NO (the ControlPointCode is 0x00).

5.2.4 Relationship between other IPMP Tools

Because the Control Point of the IPMP_Info_Engine is discussed in the preceding section, we consider the relationship between the IPMP_Info_Tool and other IPMP Tools. For instance, Figure 5-8 shows that the communication between the IPMP_Info_Engine tool and other IPMP tools in PreDecoderFilter and PostDecoderFilter. Since the architecture is message-based, we discuss the IPMP Messages in this scenario.

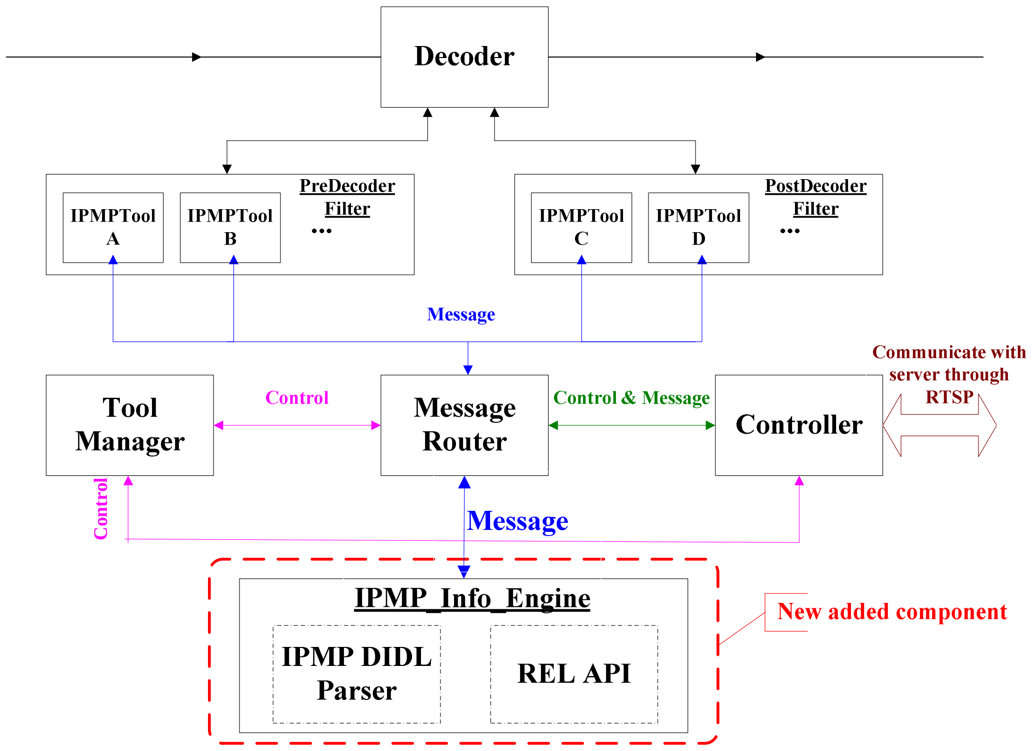


Figure 5-8 Modified block diagram at the client side

IPMP Message can include various kinds of information, such as control information, rights data, and key data. Basic flowchart of generate an IPMP Message is illustrated in Figure 5-9.

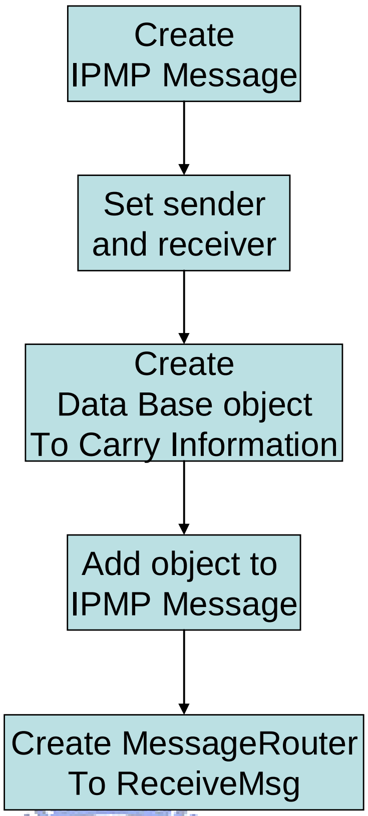


Figure 5-9 Flowchart of generating an IPMP Message

Although there are many types of IPMP Data representing different kinds of information, we choose IPMP_OpaqueData as the data container that can be carried by the IPMP Message for simplicity and flexibility. Its data structure is shown below [1].

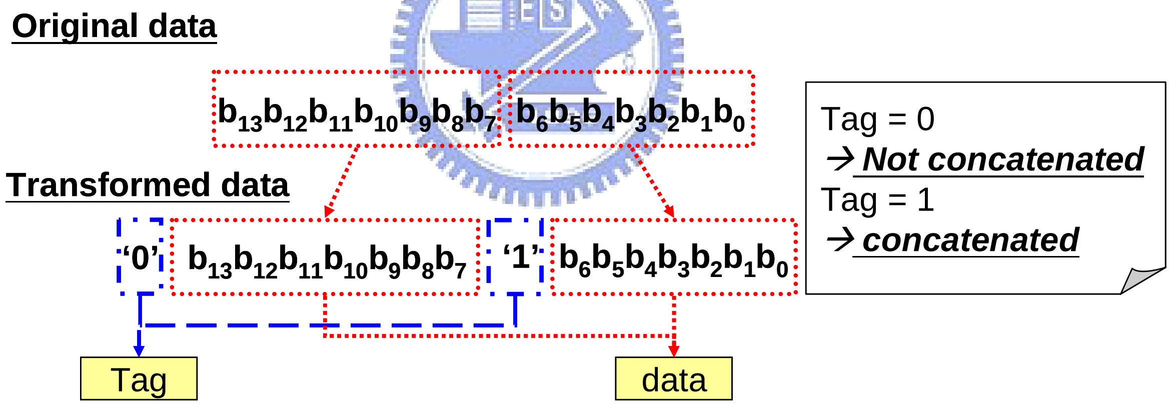
```
class IPMP_OpaqueData extends IPMP_Data_BaseClass
:bit(8) tag = IPMP_OpaqueData_tag
{
            ByteArray opaqueData;
}


class ByteArray
{
            unsigned long(32)        SizeOfArray;
            bit(8)                   Data[SizeOfArray + 1];
}
```

The opaqueData is the opaque IPMP information conveyed to IPMP Tools. Any format of information that is able be converted to the data structure (“ByteArray”) can be included in the opaqueData. Therefore, the procedure for generating an IPMP Message which carries

IPMP_OpaqueData is shown as follows.

1. Create an IPMP_OpaqueData object.
2. Insert information into the IPMP_OpaqueData object.
3. Create an IPMP message.
4. Set the sender and recipient of the IPMP message.
5. Add the IPMP_OpaqueData object into the IPMP message.
6. Call the Message Router to receive the IPMP message.

The implementation of IPMP_Opaque is based on the MPEG-2 IPMP Extension system developed by Panasonic Singapore Laboratories Pte Ltd (PSL) [26]. According to its design, the maximum size of OpaqueData in IPMP_Opaque object is 2^{28} bytes because the size of the length variable varies from one to four bytes and the most significant bit (MSB) of each byte is "1" if there is higher byte to be concatenated with it. The representation of the length variable can be shown in Figure 5-10.



Original data
$b_{13}b_{12}b_{11}b_{10}b_9b_8b_7$ $b_6b_5b_4b_3b_2b_1b_0$
Transformed data
'0' $b_{13}b_{12}b_{11}b_{10}b_9b_8b_7$ '1' $b_6b_5b_4b_3b_2b_1b_0$
Tag
data
Tag = 0
→ Not concatenated
Tag = 1
→ concatenated

Figure 5-10 Representation of ByteArray format

When the coded data is passed to the PreDecoderFilter, it will be processed by each IPMP tool in this IPMP Filter. The processing priority of the IPMP Tools depends on the values of the sequence code. The IPMP tool with a larger sequence code has a higher priority to process data. When an IPMP tool starts to process the given data, it must own the corresponding right to perform such functionalities. Therefore, in our implementation, an IPMP Tool has to send a message to the IPMP_Info_Engine tool for requesting the authorization. The Message Router receives this message and delivers it to the

IPMP_Info_Engine tool. Then, the IPMP_Info_Engine tool performs the verification. The REL license and the user query file stored in the IPMP_Info_Engine tool are used as the input arguments to validate the user's request. After the result of verification is determined, the IPMP_Info_Engine tool sends an IPMP message that carries the result back to this IPMP tool. The transmission of IPMP messages depicted in Figure 5-11 demonstrates the interactions between the DES Tool [23] and the IPMP_Info_Engine Tool in our designed system.

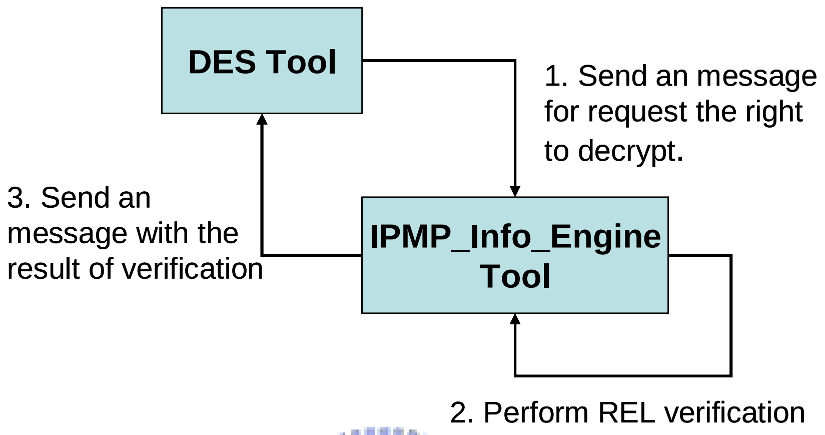

DES Tool
1. Send an message for request the right to decrypt.
3. Send an message with the result of verification
IPMP_Info_Engine Tool
2. Perform REL verification

Figure 5-11 Transmission of IPMP messages between DES Tool and IPMP_Info_Engine Tool

In this example, the communication should be taken place only once (when the first-time innovation). The details of how an IPMP Tool processes the given data is shown in Figure 5-12.

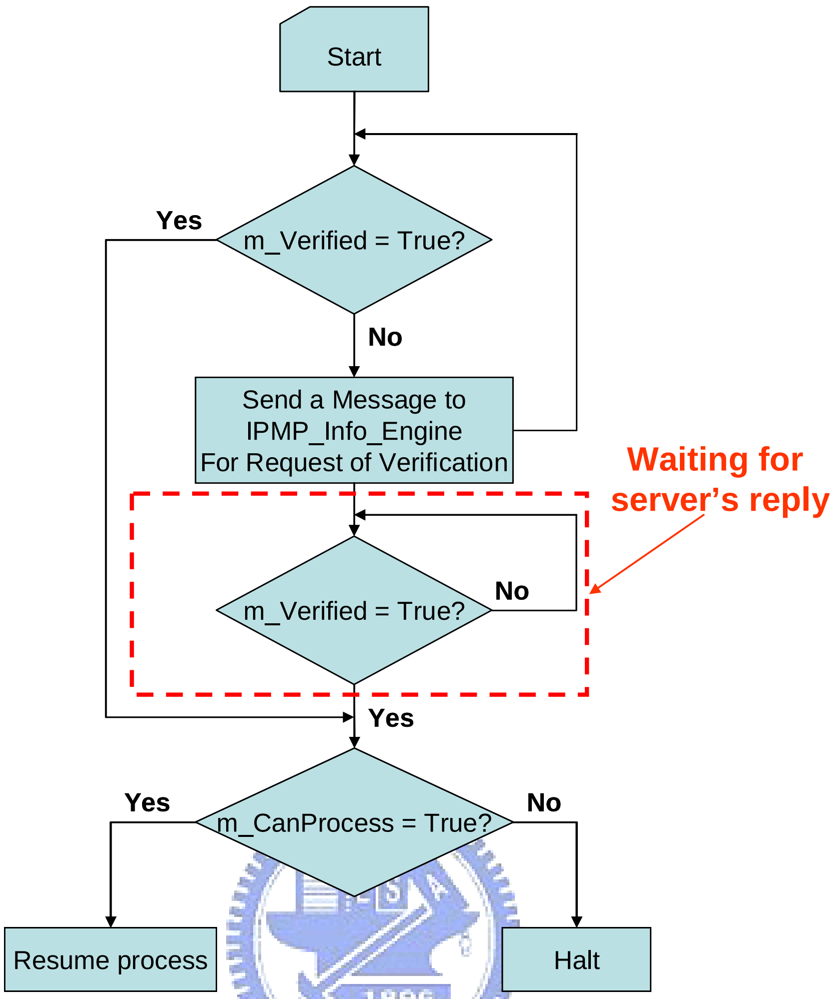


Figure 5-12 Flowchart of processing data in an IPMP Tool

When an IPMP Tool starts to process data, the value of a flag, m_Verified, is checked. If it is false, this IPMP Tool sends an IPMP Message to the IPMP_Info_Engine tool for the request of verification. After the message is received by the Message Router, this Tool will wait for the value of m_Verified becoming true. When the value of m_Verified is true, it means the result of verification sent from the IPMP_Info_Engine tool is received by this IPMP Tool. Then, this IPMP Tool continues to check the value of a flag, m_CanProcess. If its value is true, the process is resumed; otherwise, this IPMP Tool halts this process.

The APIs associated with the communication between other IPMP Tools in IPMP_Info_Engine Tool are listed below.

Method:

int IPMPInfoEngine::ReceiveMessage(IPMPToolMessageBase* *msg*)

This API is called by the Message Router to pass a message to the IPMPInfoEngine object. The parameter msg is the IPMP message to be handled by this tool. The function returns zero when it succeeds.

5.3 Content protection mechanism

In the MPEG-21 Testbed, the encoded bitstream is transported through an RTP channel which can not prevent the bistream from eavesdropping. To increase the safety of transportation, we use an encryption tool at the server side to encrypt the bitstream before transmission. The encryption algorithms can be divided into three types (Figure 5-13):

1) Symmetric cryptography: Uses a single key for both encryption and decryption.
2) Asymmetric cryptography: Uses one key for encryption and the other (different key) for decryption.
3) Hash function: Uses a mathematical transformation to irreversibly "encrypt" the information.

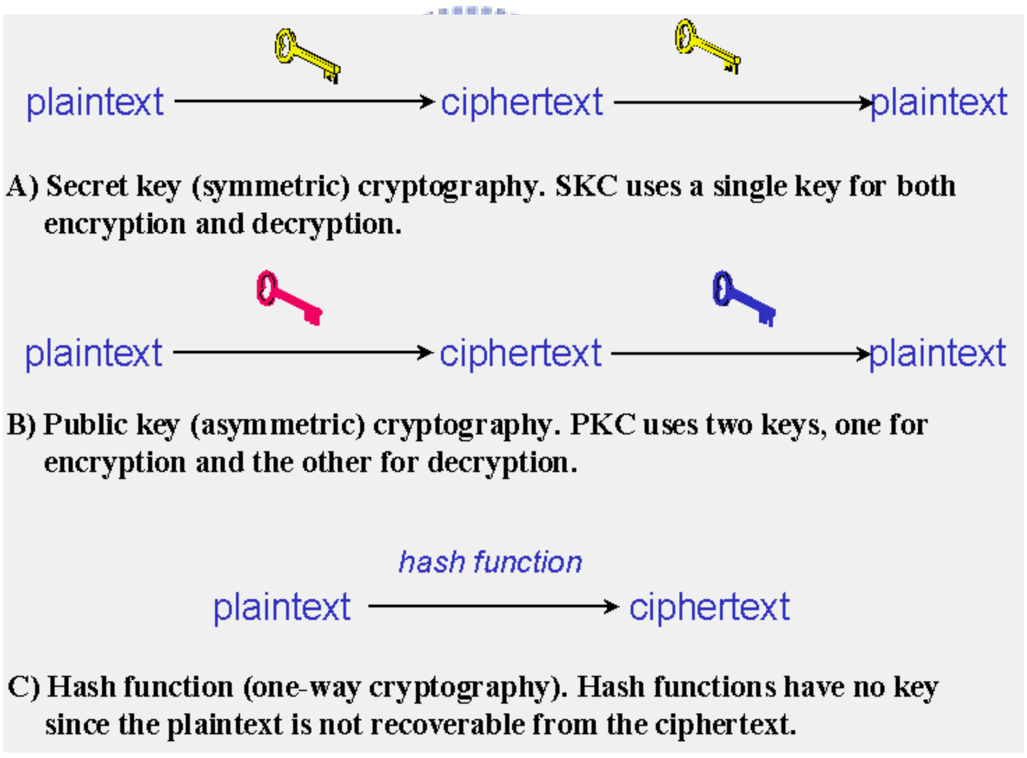

plaintext
ciphertext
plaintext
A) Secret key (symmetric) cryptography. SKC uses a single key for both encryption and decryption.
plaintext
ciphertext
plaintext
B) Public key (asymmetric) cryptography. PKC uses two keys, one for encryption and the other for decryption.
hash function
plaintext
ciphertext
C) Hash function (one-way cryptography). Hash functions have no key since the plaintext is not recoverable from the ciphertext.

Figure 5-13 Three types of cryptography [31]

The original implementation of the MPEG-4 IPMPX on the MPEG-21 Testbed [8] designs a pair of DES tools, one for encryption and the other for decryption. The DES algorithm is a kind of symmetric cryptography, and it is a block cipher algorithm. Because the encryption key at the server side has to be transferred to the client side for decryption, we choose RSA, one of the well-known asymmetric algorithms, to protect the DES encryption key during the transmission. The content protection scheme combined with the key management is shown in Figure 5-14.

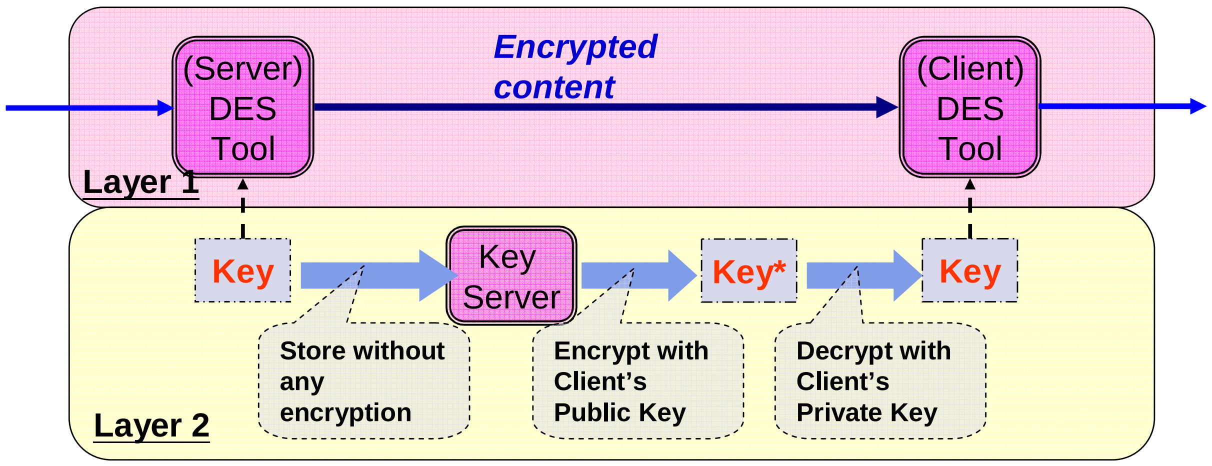

Encrypted content
(Server) DES Tool
(Client) DES Tool
Layer 1
Key
Key Server
Key*
Key
Store without any encryption
Encrypt with Client's Public Key
Decrypt with Client's Private Key
Layer 2

Figure 5-14 Content protection scheme

There are two protection layers. Layer 1 protects the content, and Layer 2 protects the encryption keys. After the content is encrypted by the server-side DES Tool, the protected content is sent to the client. Meanwhile, the encryption key is stored in the Key server's database. When the client receives the encrypted content, it links to the Key server and asks for the decryption key. The Key server uses the RSA algorithm to encrypt the DES key with the client's public key. This additional key protection guarantees that only the authorized client can recover the correct decryption key. As soon as the client gets this protected encryption key, it decrypts this key by using its private key. Consequently, the client can consume the content correctly.

In this scheme, the Key sever is designed to store the encryption key of the specific content and to transfer this key in a protected form to the client. We implement this Key server as a web server. To integrate this Key server with the MPEG-21 REL, we add the validation functionality to the Key server. The flow chart of the Key server is shown in Figure 5-15. The Key Server first receives a request message from the client, and validates that if the client has the right to receive the content decryption key, "key_C." If the client is authorized to get "key_C," the Key server generates the encrypted version (key_R) using the client's public key. Otherwise, the Key server returns an error message.

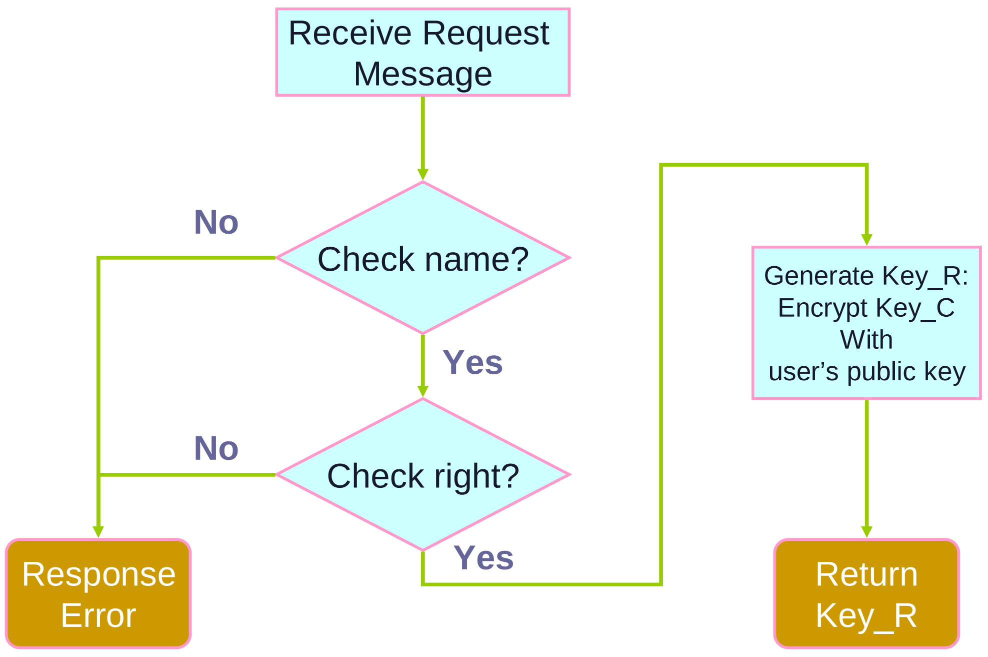


Figure 5-15 Flow chart of the Key server

5.3.1 Add one condition, "ExerciseMechanism"

In the REL reference software, only two conditions and one right are supported. To demonstrate our designed system, we define a new condition, "ExerciseMechanism". This condition provides a description of server which contains server's URI and parameters. An example of this condition is described as follows:

```
<r:exerciseMechanism>
        <r:exerciseService>
                <r:serviceReference>
                        <sx:wsdlAddress>
                               <sx:kind/>
                                <sx:address>
                                        <digitalResource>
                                             <nonSecureIndirect
                                               URI="http://localhost/sxh/EMServer_v1.asp"/>
                                        </digitalResource>
                                </sx:address>
                        </sx:wsdlAddress>
                        <r:serviceParameters/>
                </r:serviceReference>
        </r:exerciseService>
</r:exerciseMechanism>
```

Figure 5-16 Example of a condition, "ExerciseMechanism"

In Figure 5-16, a server is described by <sx:wsdlAddress>, which contains an URI to link to this server, and <r:serviceParameters>, the required information necessary to pass to this server. Here, “http://localhost/sxh/EMServer_v1.asp” is an URI to the server. This condition is fulfilled when the verification result parsed from the response of server is true. To adding this condition to our implementation, we write a condition validator, “validateEMcondition()”, which is integrated into the “validateCondition” module. The dataflow is represented in Figure 5-17. First, the program retrieves the server’s URI and parameters. We use the server’s URI to link to server with server’s parameters. Then, server generates the response to the program according to these parameters. If the connection is successful, the program receives the response and analyzes the state of the response; otherwise, an error message is returned. If the state is true, the program returns the message which represents successful verification; otherwise, the message that represents false verification or connection error is returned.

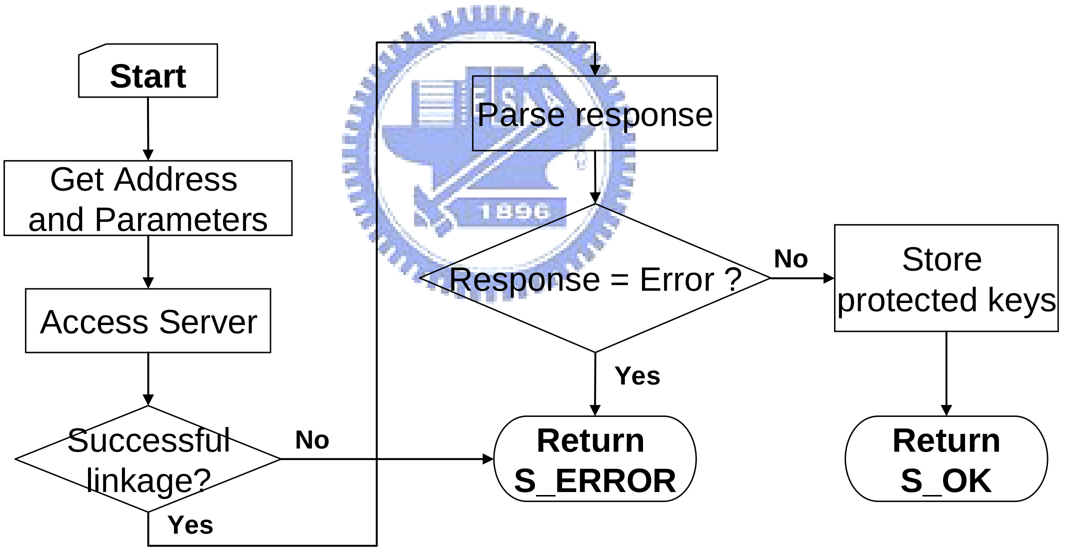


Figure 5-17 Dataflow of validating the condition, “exerciseMechanism”

The APIs associated with the condition “exerciseMechanism” are listed below.

Method:

HRESULT validateEMCondition(CComPtr<IXMLDOMElement> spEMConditionElt)

This function checks to see if the user can exercise the right. This function contacts a web service to verify this. The spEMConditionElt object is content of an “exerciseMechanism” condition. The method returns S_OK when succeed.

HRESULT accessEMServer(TCHAR *tszPath)

This function accesses the exercise mechanism and, depending on the input parameter of tszPath. The tszPath object is the path to the server. The method returns S_OK when the response of server is true.

Chapter 6

Application Examples

In order to demonstrate the functionalities of our implementation, we design three application examples as follows.

- *Online Play*: This application demonstrates how to manage the user's "play" right in a real-time streaming system.
- *Preview*: This application demonstrates how to manage the user's "preview" right in a real-time streaming system.
- *Super distribution*: This application demonstrates how to manage the user's right in a distributed mobile environment.

Figure 5-8 shows our system. The client side has a DES Decryption Tool and an IPMP_Info_Engine Tool. There is a DES Encryption Tool at the server side. All the application examples have a similar system structure. We first describe the initialization settings in an Initial Object Descriptor for the client terminal of our implementation.

There are three main components in an IOD: IPMP Tool List, IPMP Tool Descriptor, and the IPMP Tool Descriptor in the Elementary Stream Descriptor (ESD). The attributes and settings of these components are shown as follows.

```
IPMP_ToolListDescriptor
{
   numTools = 2;
   IPMPToolD[0]
   {
      IPMP_ToolID = [140,113,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1];
      isAltGroup = 0;
      isParametric = 0;
      numURLs = 0;
      ToolURL = NULL;
   }
   IPMPToolD[1]
   {
      IPMP_ToolID = [140,113,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4];
      isAltGroup = 0;
      isParametric = 0;
      numURLs = 0;
```

```
        ToolURL = NULL;
    }
}
```

There is an IPMP Tool List residing in an IOD. Because the client terminal has two IPMP Tools in our system, there must be two IPMPToolIDs on the IPMP Tool Descriptor List and thus *numTools* is set to 2. Each IPMPToolD has a unique *IPMP_ToolID*, because a unique mode is needed 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"};
            };
        }
        numOfData = 0;
        IPMPX_data = NULL;
}
IPMP_ToolDescriptor[1]
{
        IPMP_ToolDescriptorID = 2;
        IPMP_ToolID = [140,113,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4];
        URLString = NULL;
        isInitialize = 1;
        IPMP_Initialize
        {
            controlPointCode = 0x00;
            sequenceCode = 220;
            numOfData = 1;
            IPMPX_data[0]:IPMP_OpaqueData
            {
                opaquedata = {"IPMP DIDL element"};
            };
        }
        numOfData = 0;
        IPMPX_data = NULL;
}
```

Because there are two IPMP Tools instantiated in this demonstration system, we need two IPMP Tool Descriptors for them. Although different IPMP Tools can share the same IPMP Tool Descriptor, the IPMP Tools which appear in the demonstration system can be quite different. There are several parameters which can be set at the users' will, such as *IPMP_ToolDescriptorID*. In both IPMP Tool Descriptors, the *isInitialize* is set to 1 to indicate that the IPMP Tools are newly instantiated. The *controlPointCode*, whose value is 0x01, means that this IPMP Tool are connected to the PreDecoderFilter, and the *controlPointCode*, whose value is 0x00, means that the other IPMP Tool is not connected to anyplace. And the *sequenceCode* indicates the order for processing the data. Finally, the IPMPX_data carried by IPMP_Initialize is set as IPMP_OpaqueData to carry the initialization information for IPMP Tools such as the duration for changing keys and initial keys.

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

The IPMP Tool Descriptor Pointers within the ES Descriptor indicate that the stream described by this ESD is protected by the IPMP Tools listed by the IPMP_ToolDescriptorID. The relationship between the Tool Descriptor and the Tool Descriptor Pointer is shown in Figure 6-1.

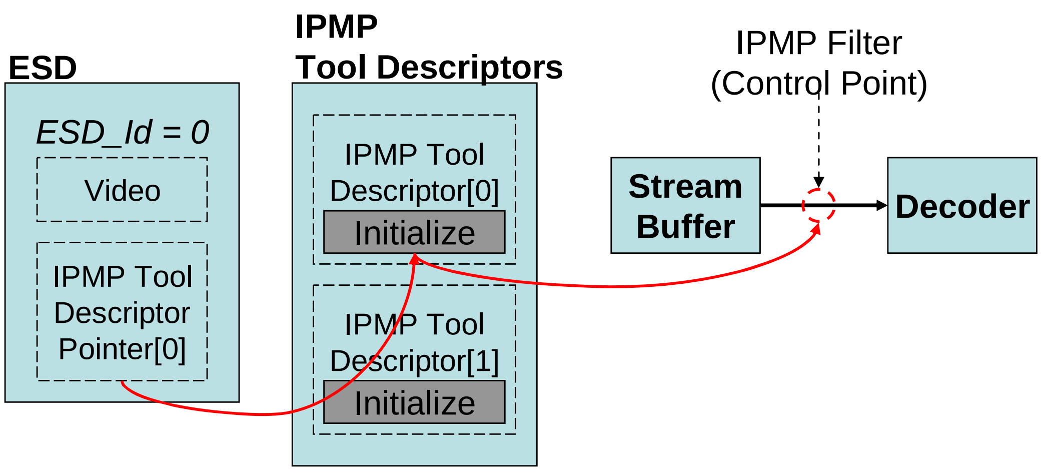


Figure 6-1 Relationship among IPMP Tool Descriptor Pointer and IPMP Tool Descriptor

6.1 Application Example 1 --- Online Play

We design an example, “Online Play,” which demonstrates how to manage the “play” right in a real-time streaming system. In this example, we want to send a request to a remote server if the principal has the right to play the video file. In addition to provide the result of authorization, the remote server is also a key server. Only the result of verification is true, the correct keys are sent back to the client. Therefore, we design a simple license that describes “the principal can play the resource when online verification is true.” To perform online verfication, the license contains the “exerciseMechanism” condition to store the server address and the server parameters. The “serviceParameters” element in the “exerciseMechanism” condition contains the parameters that the remote server needs to verify the client’s right. In this example, the “keyholder” (principal) and the “mx:play” (right) are included. This license is shown below.

```
<license licenseId="Demo1">
        <grant>
                <keyHolder> … </keyHolder>
                <mx:play/>
                <digitalResource>
                        <nonSecureIndirect URI="foreman_qcif"/>
                </digitalResource>
                <exerciseMechanism>
                        <exerciseService>
                                <serviceReference>
                                        <sx:wsdlAddress>
                                                <sx:kind>…</sx:kind>
                                                <sx:address>
                                                        <digitalResource>
                                                                <nonSecureIndirect
URI="http://localhost/sxh/EMServer_v1.asp"/>
                                                        </digitalResource>
                                                </sx:address>
                                        </sx:wsdlAddress>
                                        <serviceParameters>
                                                <datum
                                                        <keyHolder licensePartId="Alice"/>
                                                </datum>
                                                <datum>
                                                        <mx:play/>
                                                </datum>
                                        </serviceParameters>
                                </serviceReference>
                        </exerciseService>
                </exerciseMechanism>
        </grant>
```

```
    <issuer>…</issuer>
</license>
```

6.1.1 Procedure

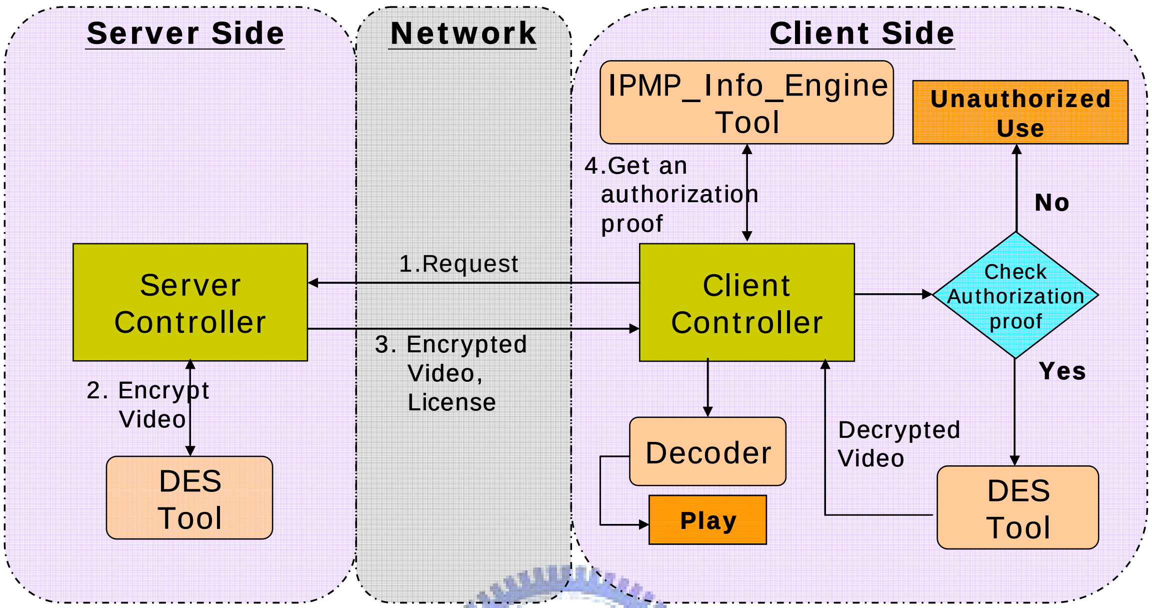


Figure 6-2 System flowchart

The flow of execution is shown in Figure 6-2. First, the client sends a request of playing the video file to the server. Then the server accesses its IOD file, and resolves it to initialize the server's IPMPX system. After sending the client's IOD file to the client, the server performs encryption on the video file and sends these protected packets to the client. Before receiving the server's packets, the client also has to access its IOD file for the initialization of the client's IPMPX system, and the IPMP_Info_Engine tool gets a license file from an IPMP Tool Descriptor. When the client's DES tool starts to decrypt data, the IPMP_Info_Engine tool sends a request to call the REL's API for generating an authorization proof.

The flow of authorization is shown in Figure 6-3 and Figure 6-4. If the user is authorized to play this video, the remote server sends the decryption keys in a protected form to the client. Then, the received keys are stored in a local file. After reading and decrypting the received keys with the client's private key, the DES Tool can decrypt the received video packets. The user can now play this video file. Otherwise, if the user is not authorized, our program will halt and request the user for closing the window.

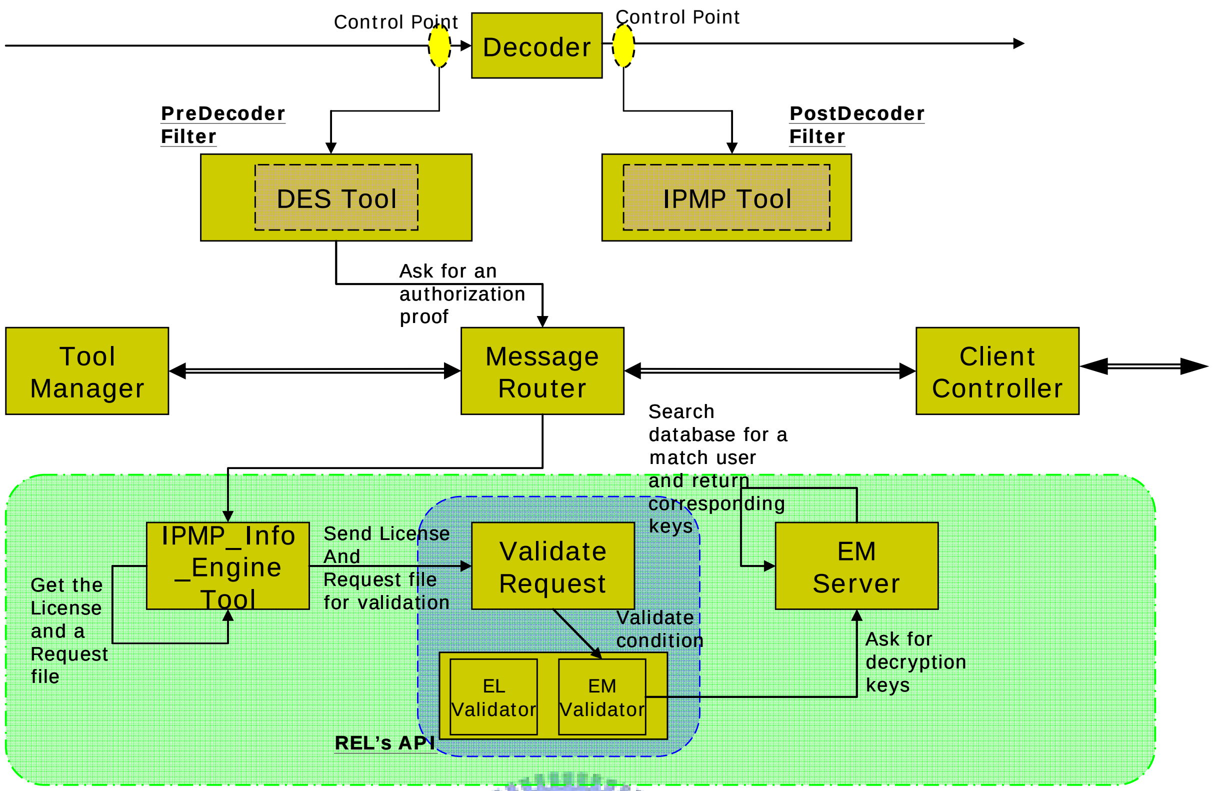


Figure 6-3 Flowchart of authorization (request for verification)

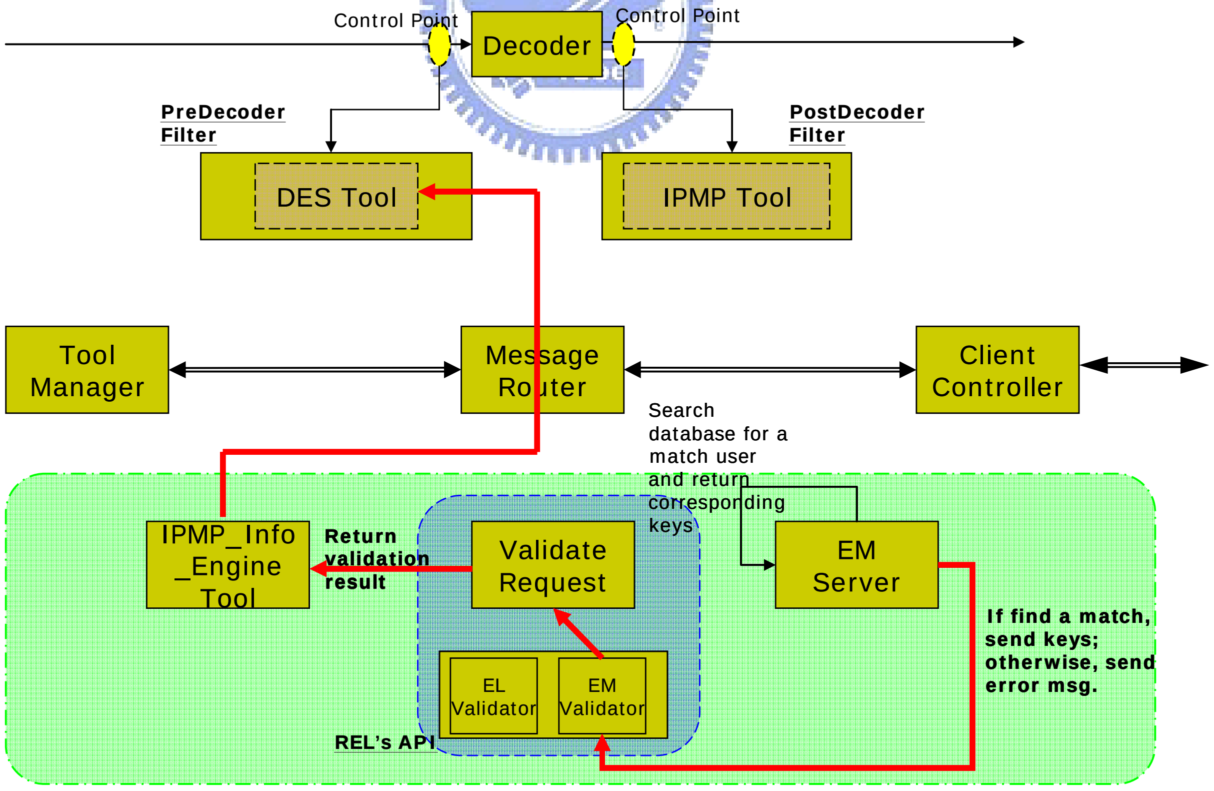
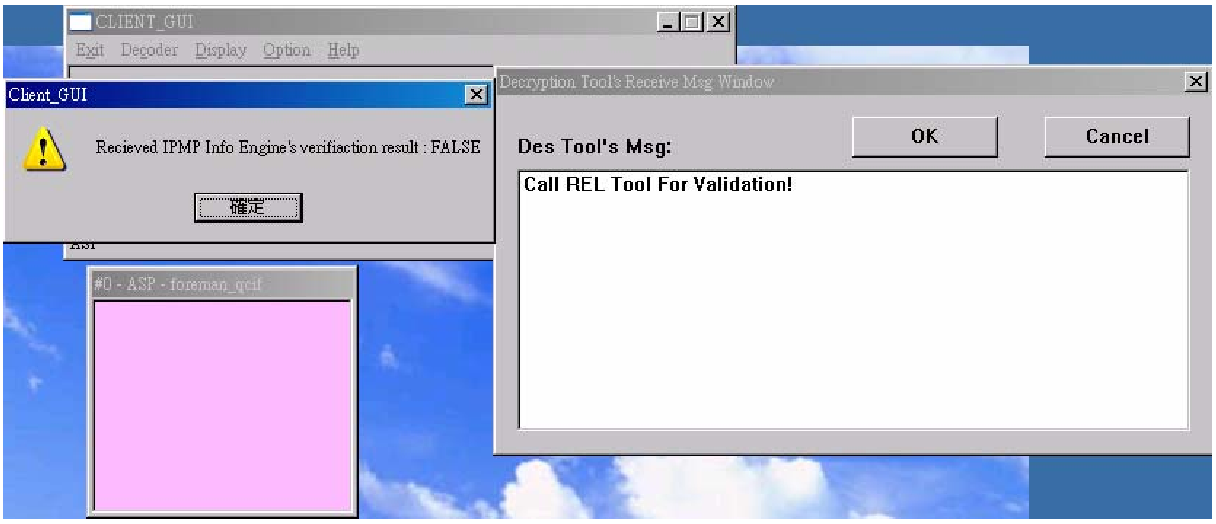


Figure 6-4 Flowchart of authorization (return the result of verification)

The result of unauthorized access is shown in Figure 6-5. The popup window shows an error message, "*Received IPMP_Info_Engine's verification result: False*." The video display window is located at the lower left corner, and it does not show any video.


CLIENT_GUI
Exit Decoder Display Option Help
Client_GUI
確定
Decryption Tool's Receive Msg Window
OK
Cancel
#0 - ASP - foreman_qcif

Recieved IPMP Info Engine's verifiaction result : FALSE

Des Tool's Msg:

Call REL Tool For Validation!

Figure 6-5 Screenshot when the validation result is false

6.2 Application Example 2 --- Preview

In e-commerce, it is popular to allow the consumers to preview video/audio clips in which they are interested. We design an example which demonstrates how to manage the "preview" behavior in a real-time streaming system.

We assume that the consumers can preview a video clip without having any licenses among the first n macroblocks in our system. Instead of using a license to describe the "preview" right, we use a counter in the DES Tool to check if all the first n macroblocks of a video file are processed. We add the counter's information into the opaquedata of the DES Tool's Tool Descriptor.

IPMPX_data[0]:IPMP_OpaqueData //DES Tool
```
{
    opaquedata
    = {"number of preview macroblocks", "duration to change the key","initial key"}
};
```

6.2.1 Procedure

The entire execution procedure is divided into two stages, preview and authorized play.

Their flowcharts are depicted in Figure 6-6 and Figure 6-7, respectively. At the first stage, the user is permitted to play the first n macroblocks freely, and the server sends the unprotected raw data to the client. The client directly plays back the video file without any IPMP process.

Entering the second stage, the server sends the protected packets encrypted by the server's DES Tool. When the client receives these encrypted packets, the client pops up a window to ask the user to enter the correct license and the query file or to terminate the program. Here, our designed license file is identical to the one in the application example 1. When both the license and the query file are accessed, the client's IPMP_Info_Engine Tool is requested to perform authorization. The details of the correct license and the query file have been described in section 6.1.1. If the result of authorization is true, the client program continues to play back the video; otherwise, the program will halt itself, and ask the user to exit.

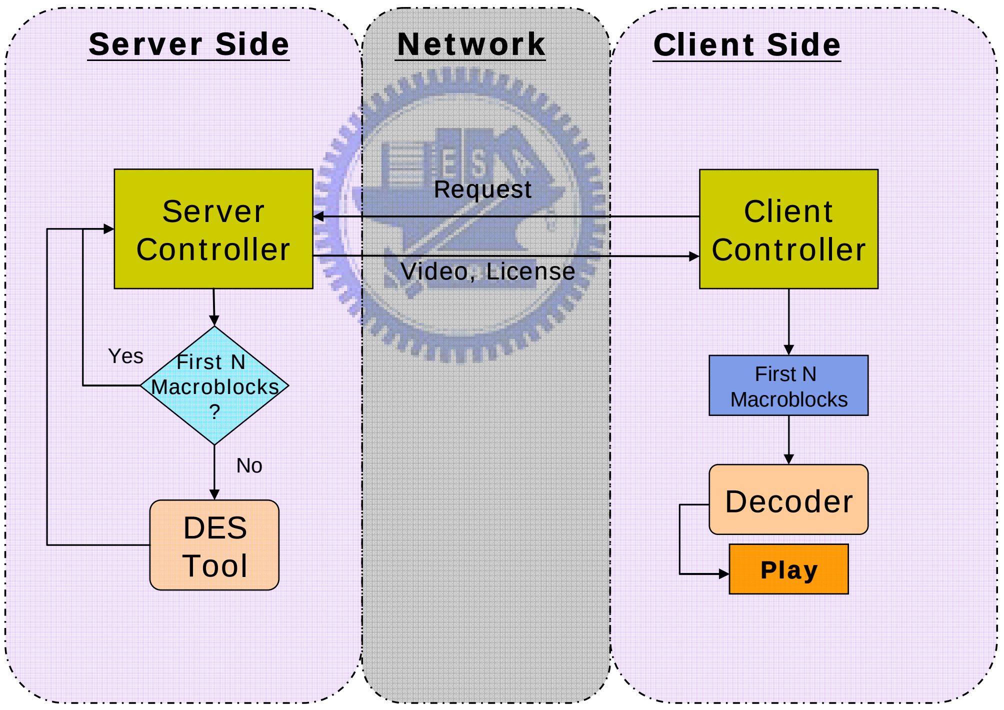


Figure 6-6 Flowchart of stage 1 (preview)

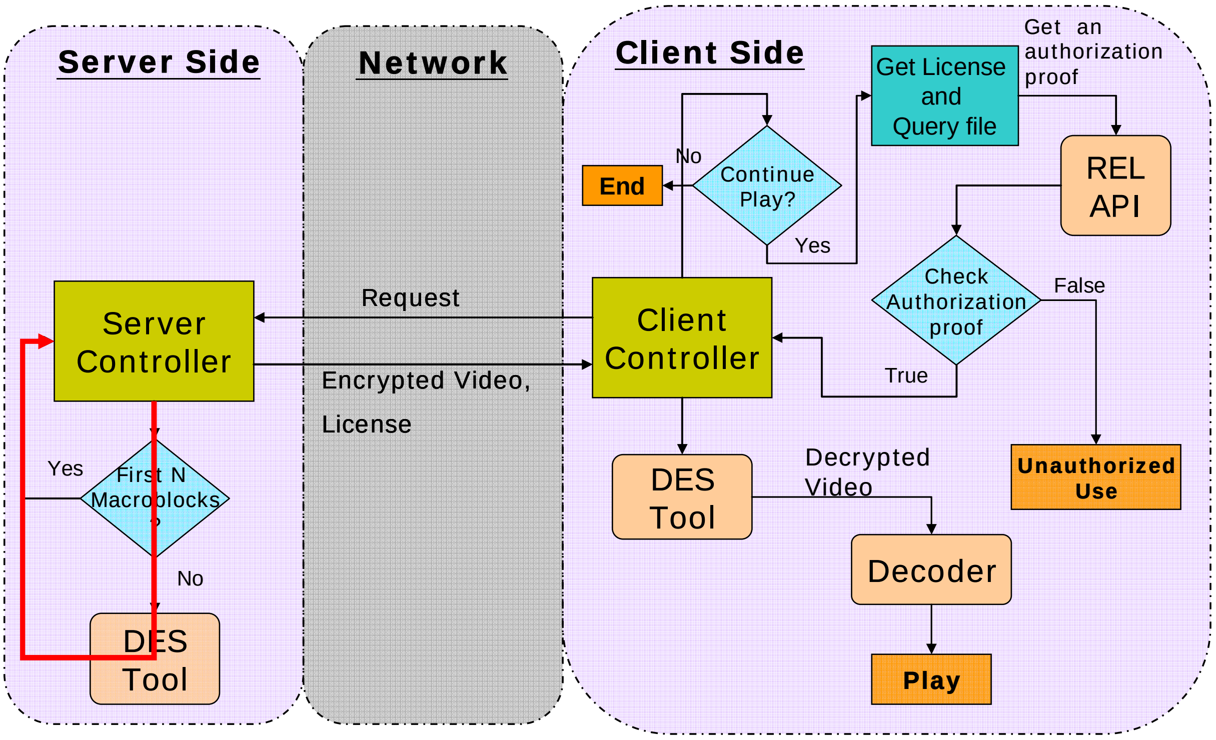
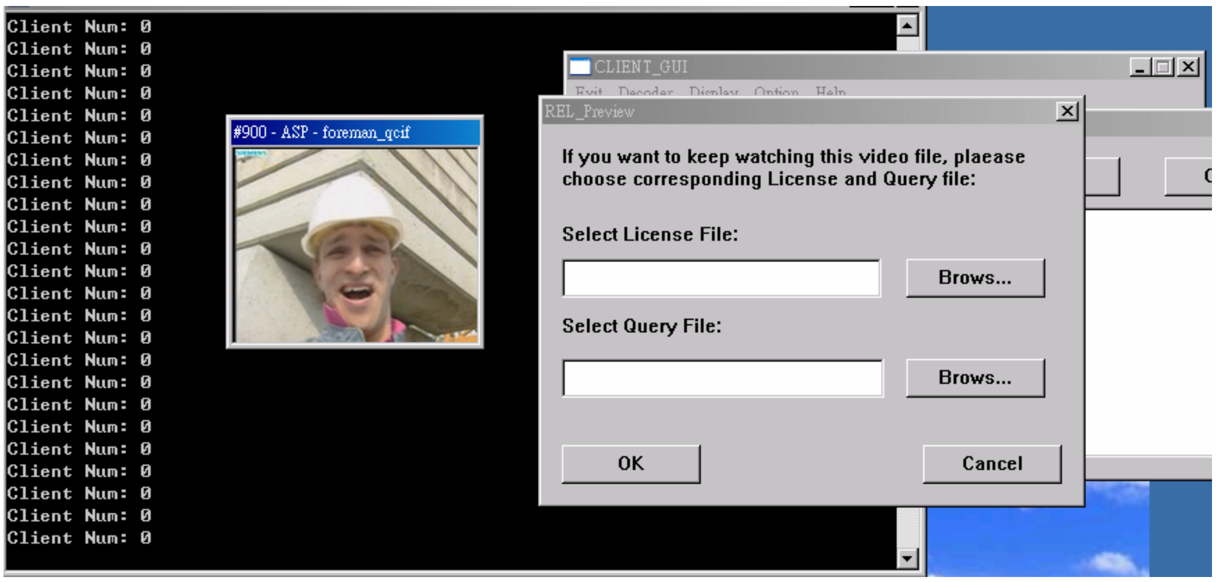


Figure 6-7 Flowchart of stage 2 (authorized play)

The screenshots are shown in Figure 6-8 and Figure 6-9. In Figure 6-8, the left small window is the video display window, which is playing back the first n macroblocks. The right side window is the "REL_Preview," which sends requests for entering the correct license and query file. In Figure 6-9, because users choose not to continue watching this video, a caution message is shown in the "Decription tool's receiving message window," and the user is asked to close the window.



```
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
```

#900 - ASP - foreman_qcif

CLIENT_GUI

REL_Preview

If you want to keep watching this video file, pleaase choose corresponding License and Query file:

Select License File:

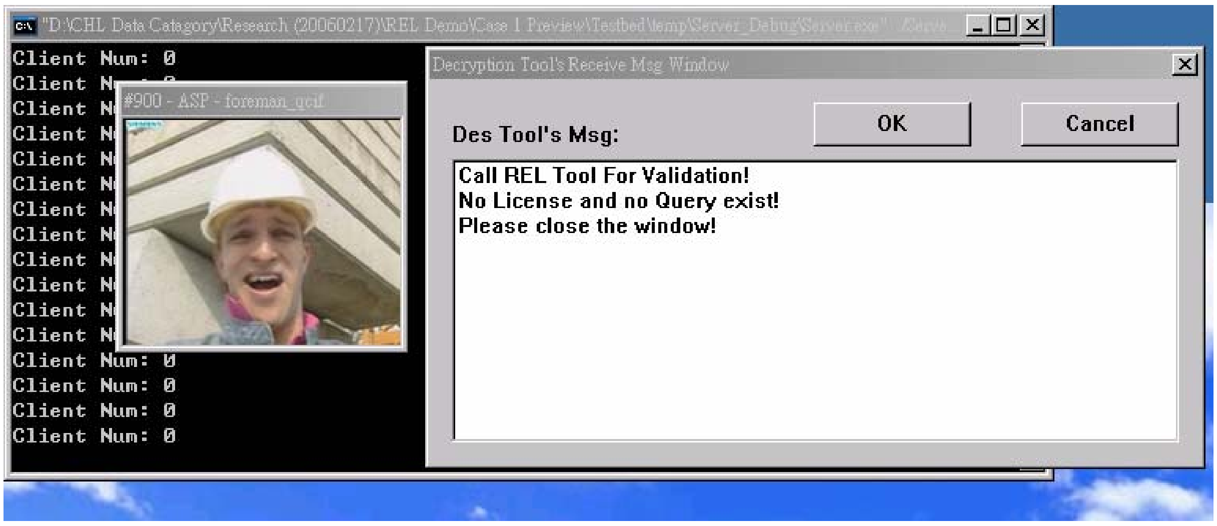
Brows...

Select Query File:

Brows...

OK Cancel

Figure 6-8 Screenshot_1 in application example 2 (“Preview”)


"D:\CHL Data Catagory\Research (20060217)\REL Demo\Case 1 Preview\Testbed\temp\Server_Debug\Server.exe"
#900 - ASP - foreman_qcif
Decryption Tool's Receive Msg Window
OK
Cancel

```
Client Num: 0
Client N
Client N
Client N
Client N
Client N
Client N
Client N
Client N
Client N
Client N
Client N
Client Num: 0
Client Num: 0
Client Num: 0
Client Num: 0
```

Des Tool's Msg:

Call REL Tool For Validation!
No License and no Query exist!
Please close the window!

Figure 6-9 Screenshot_2 in application example 2 (“Preview”)

6.3 Application Example 3 --- Super-Distribution

We design an example, “super-distribution” scenario, which demonstrates how to manage the user’s right in a distributed mobile environment. The idea of this scenario is originated from the OMA DRM v2.0. Figure 6-10 shows that the system is divided into the content provider and the user. The protected content and the rights object can be delivered to the user separably. Each rights object is restricted to one user, and the protected content can be distributed without any constraint. When a user receives the protected content, he or she has to purchase his/her own rights object (license) for consuming the content.

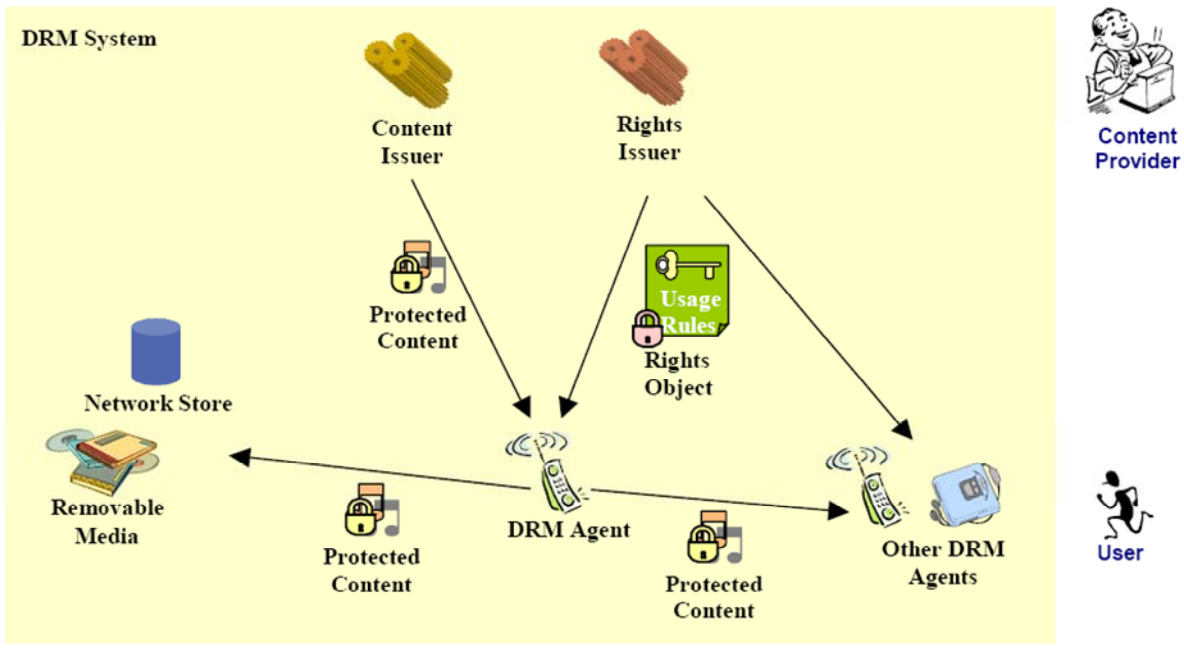


Figure 6-10 Concept of OMA DRM v2.0 [3]

A mobile device may not be always connected to the network. To simulate this situation, we design a license that supports both online and offline verification. The online verification is achieved by a remote server, and the offline verification is achieved by local validation. In Figure 6-11, this license includes two grants, one is for online situation and the other is for offline situation. In this license, Grant 1 describes that “the principal can play back the resource when the online verification is true and within a specific time interval.” Grant 2 describes that “the principal can play back the resource when the offline verification is true and the playing counts doe not exceed five times.”

The condition elements in Grants 1 and 2 are encapsulated in the “allConditions” element, an “and” logic operation, that every condition element inside has to be satisfied. Each “allConditions” element represents two condition elements. In Grant 1, one requests the key server for verification and the other validates execution tine. In grant 2, one performs the offline verification and the other validates the counts of offline consumption. The verification flow of this license is drawn in Figure 6-12. Grant 1 is validated first. If one condition in this grant is not satisfied, Grant 1 is unauthorized. Then, Grant 2 is checked. If both Grants are unauthorized, the result of authorization is false.

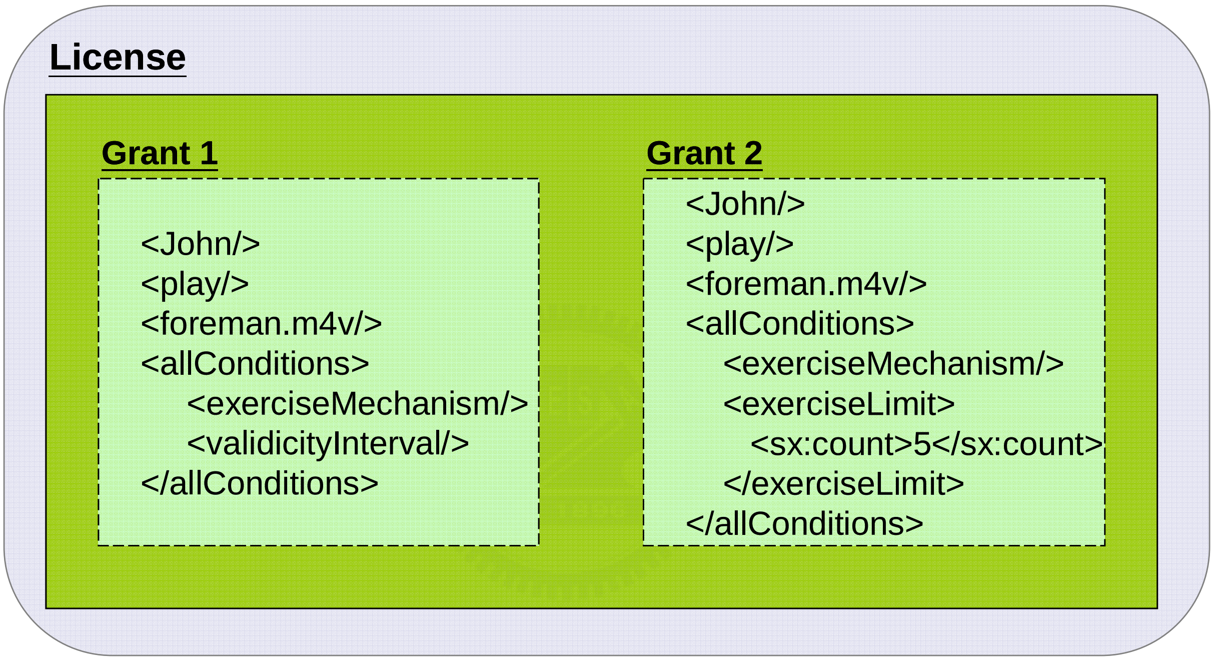


Figure 6-11 Structure of the license of “Super Distribution”

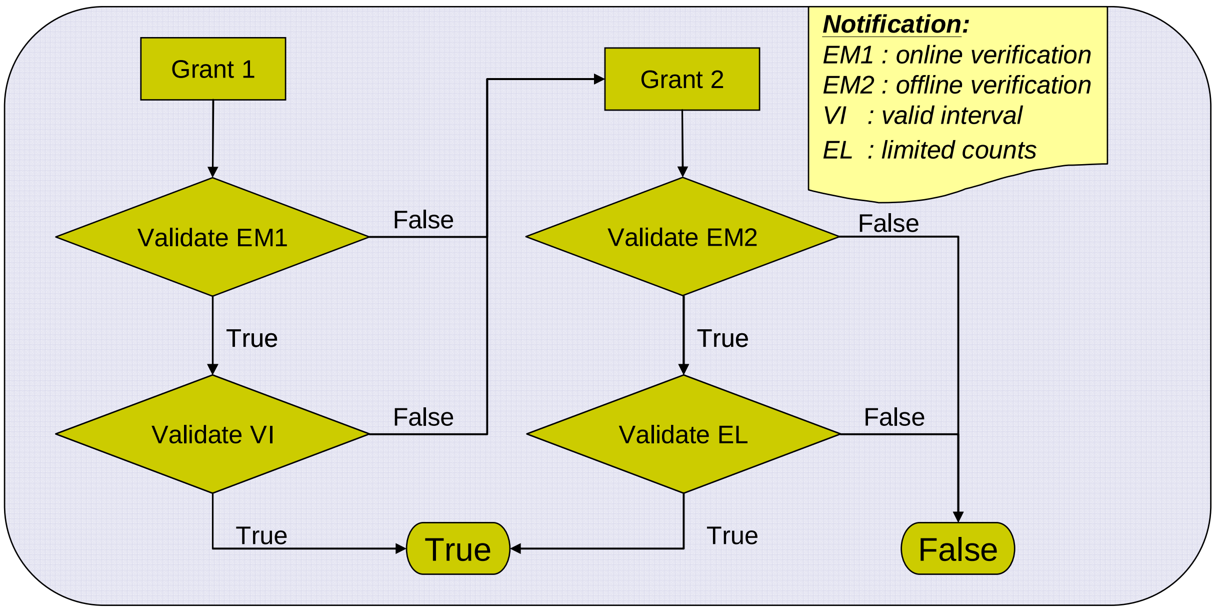


Figure 6-12 Verification flow of conditions in the License of “Super Distribution”

Because the online verification is more secure than the offline verification, we assume that the status of the online verification is a necessary condition of the offline verification. So, our program has to link to the remote server for the verification first, and the remote server returns the decryption keys if the client is authorized. This decryption keys exist only after the remote server has authorized client. We use the decryption keys as a certificate of the online verification. Therefore, the designed offline verification can check the existence of decryption keys to know the status of the online verification. However, this certificate has a finite time span. If Grant 2 is not granted, the online certificate will be deleted.

The designed license is shown as follows:

```
<license licenseId="Demo 3">
<grant>
        <keyHolder> ...   </keyHolder>
        <mx:play/>
        <digitalResource>
                <nonSecureIndirect URI="foreman_qcif"/>
        </digitalResource>
        <allConditions>
        <exerciseMechanism>
                <exerciseService>
                        <serviceReference>
                                <sx:wsdlAddress>
                                        <sx:kind> ... </sx:kind>
                                        <sx:address>
                                                <digitalResource>
                                                        <nonSecureIndirect URI="http://localhost/sxh/EMServer_v1.asp"/>
                        </digitalResource>
                    </sx:address>
                </sx:wsdlAddress>
                <serviceParameters>
                    <datum>
                        <keyHolder licensePartId="Alice"/>
                    </datum>
                    <datum>
                        <mx:play/>
                    </datum>
                </serviceParameters>
            </serviceReference>
        </exerciseService>
    </exerciseMechanism>
    <validityInterval>
        <notBefore>2006-04-30T23:59:59</notBefore>
        <notAfter>2002-06-31T23:59:59</notAfter>
    </validityInterval>
    </allConditions>
</grant>
<grant>
    <keyHolder>      ...      </keyHolder>
    <mx:play/>
    <digitalResource>
        <nonSecureIndirect URI="foreman_qcif"/>
    </digitalResource>
    <allConditions>
        <exerciseMechanism>
            <exerciseService>
                <serviceReference>
                    <sx:wsdlAddress>
                        <sx:kind> ... </sx:kind>
                        <sx:address>
                            <digitalResource>
                                <nonSecureIndirect URI="file://c:/DecryptedKeys.key"/>
                            </digitalResource>
                        </sx:address>
                    </sx:wsdlAddress>

                </serviceReference>
            </exerciseService>
        </exerciseMechanism>
        <sx:exerciseLimit>
            <serviceReference>
                <sx:wsdlAddress>
                    <sx:kind> ...        </sx:kind>
                    <sx:address>
                        <digitalResource>
                            <nonSecureIndirect URI="file:///./RELExLimitService.dll"/>
                        </digitalResource>
                    </sx:address>
                </sx:wsdlAddress>
                <serviceParameters>
                    <datum>
                        <sx:stateDistinguisher>User_Alice</sx:stateDistinguisher>
                    </datum>
                </serviceParameters>
            </serviceReference>
            <sx:count>5</sx:count>
        </sx:exerciseLimit>
        </allConditions>
</grant>
<issuer>
        <dsig:Signature> ... </dsig:Signature>
        <details>
                <timeOfIssue>2006-01-27T15:30:00</timeOfIssue>
        </details>
</issuer>
</license>
```

6.3.1 Procedure

To realize the “Super-Distribution” scenario, we modify the client’s computer program to read local files. The modified program structure is shown in Figure 6-13. When the user inputs the protected content (video file), the IOD file having an identical file name will be opened, and the program’s IPMPX system is initialized. The IPMP_Info_Engine Tool is requested for authorizing that the user can play this protected content or not. The details of the authorization process have been described at the beginning of section 6.3. In the following, we explain the run-time processing of our program with several screenshots.

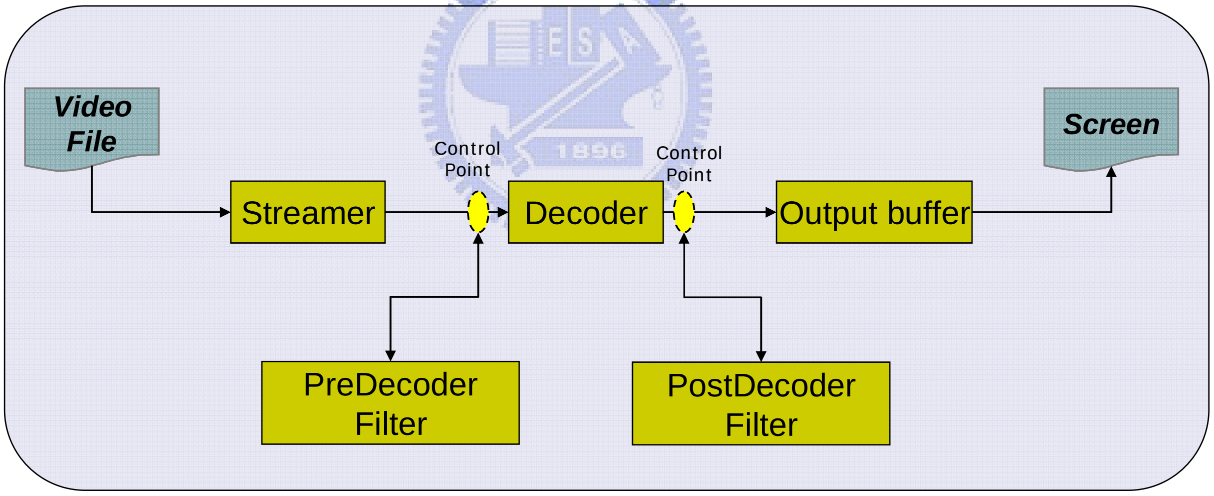
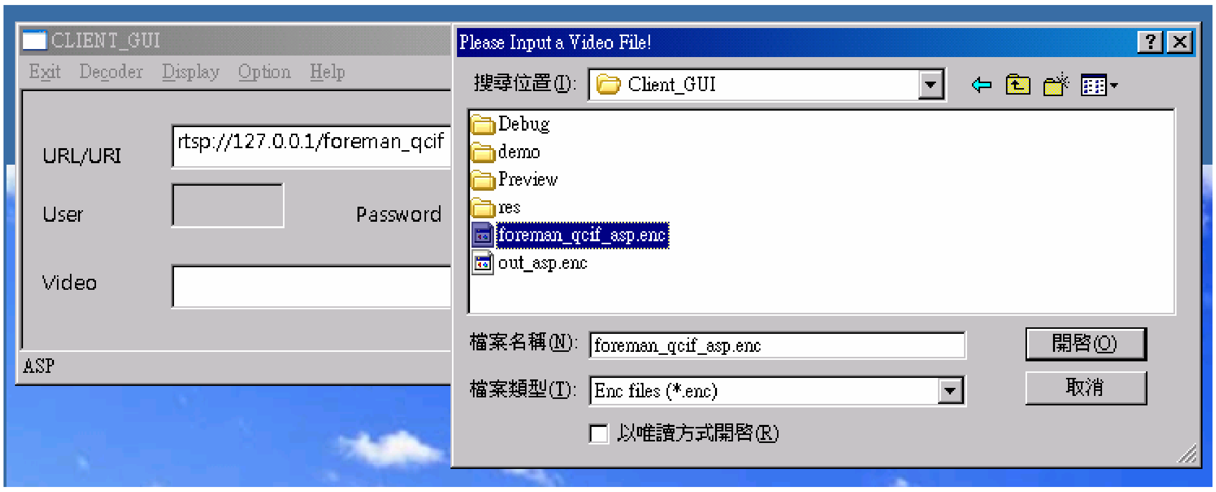


Figure 6-13 Modified program structure

First, the user has to choose a piece of local protected content to play. Figure 6-14 shows that “foreman_qcif_asp.enc” is selected. The “connect” button is pressed for starting playing this video. When the DES Tool starts to decrypt data, it has to check the result of authorization. In Figure 6-15, the result of authorization is true when the online verification is successful. Then, the DES Tool also gets decryption keys to decrypt data. Figure 6-16 shows that the user can successfully play back this piece of protected content.

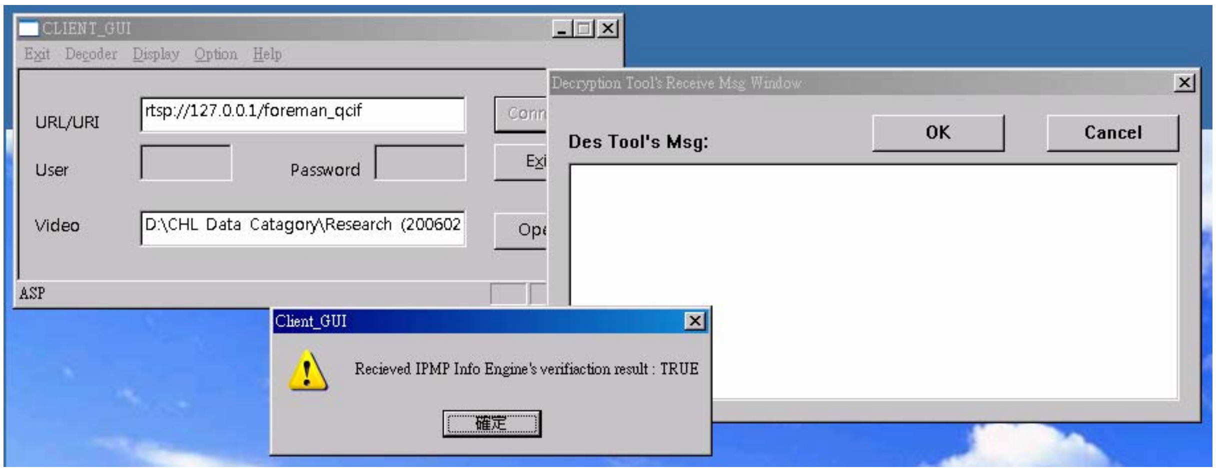


rtsp://127.0.0.1/foreman_qcif

foreman_qcif_asp.enc

Enc files (*.enc)

Figure 6-14 Screenshot_1 in application example 3 ("Super-Distribution")

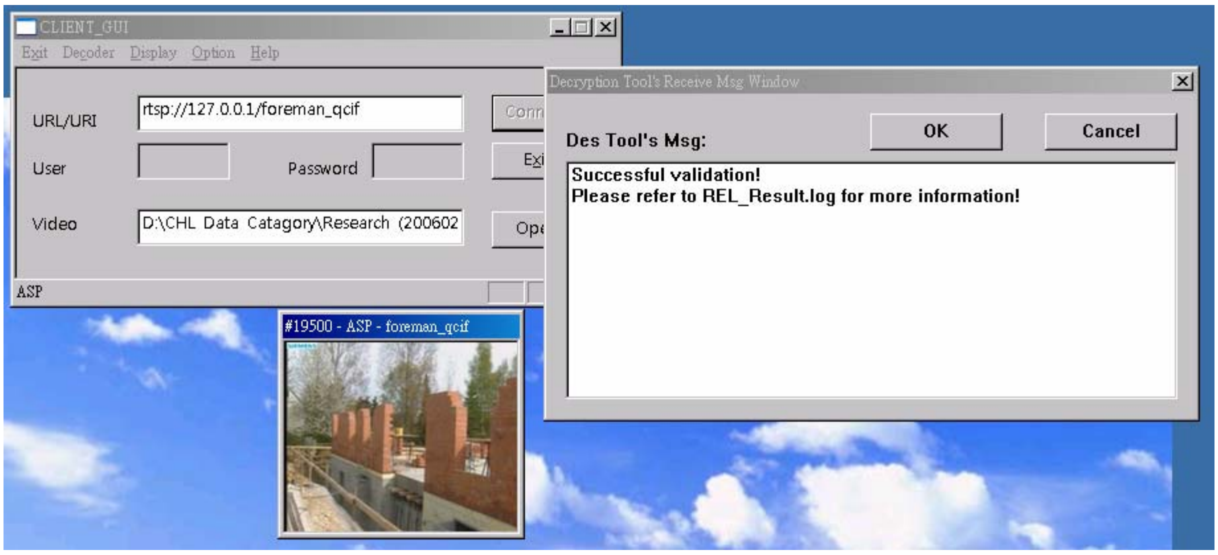


rtsp://127.0.0.1/foreman_qcif

D:\CHL Data Catagory\Research (200602

Recieved IPMP Info Engine's verifiaction result : TRUE

Figure 6-15 Screenshot_2 in application example 3 ("Super-Distribution")


CLIENT_GUI
Exit Decoder Display Option Help
URL/URI
User
Password
Video
ASP
Decryption Tool's Receive Msg Window
Des Tool's Msg:
OK
Cancel
#19500 - ASP - foreman_qcif

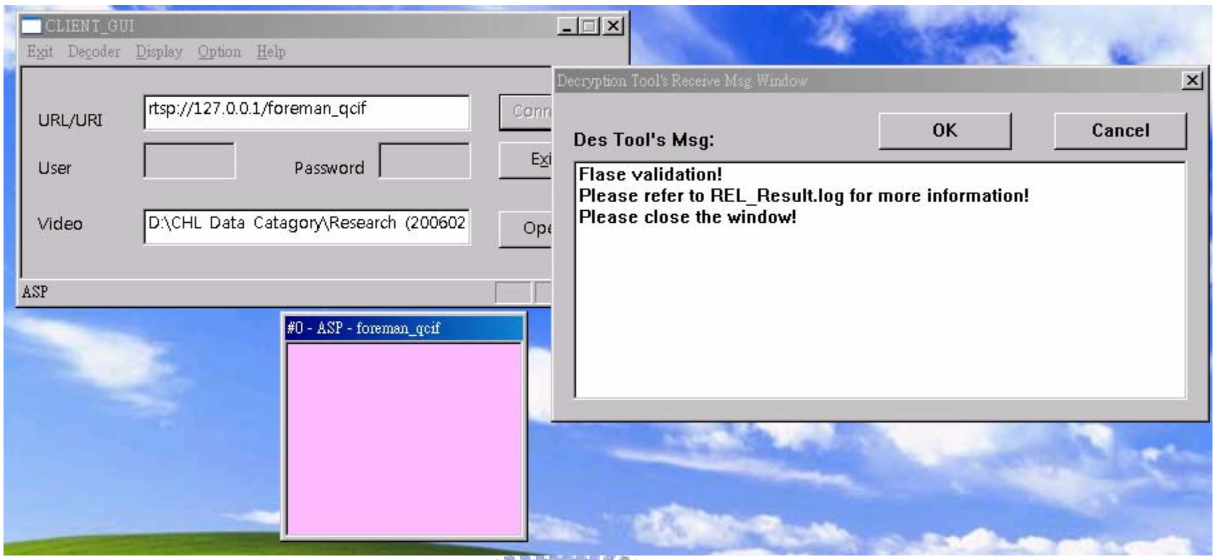
rtsp://127.0.0.1/foreman_qcif

D:\CHL Data Catagory\Research (200602

Successful validation!
Please refer to REL_Result.log for more information!

Figure 6-16 Screenshot_3 in application example 3 ("Super-Distribution")

When the network is temporally unavailable, the user wants to play this video again. But, he has been played five times already without connecting to the network. Although the result of the offline verification is true, the “exerciseLimit” condition is not satisfied. The execution result is shown in Figure 6-17.


CLIENT_GUI
Exit Decoder Display Option Help
URL/URI
rtsp://127.0.0.1/foreman_qcif
User
Password
Video
D:\CHL Data Catagory\Research (200602
ASP
#0 - ASP - foreman_qcif
Decryption Tool's Receive Msg Window
OK
Cancel
Des Tool's Msg:

Flase validation!
Please refer to REL_Result.log for more information!
Please close the window!

Figure 6-17 Screenshot_4 in application example 3 (“Super-Distribution”)

Chapter 7

Conclusions

DRM is a critical element in many multimedia systems, such as mobile communications. As various content delivery applications are getting popular, many DRM systems have been proposed. For example, Microsoft has a DRM system for the “wma” and “wmv” compression formats, and Apple creates a DRM system for its IPOD music player and Itunes software. But, these DRM systems are private and can not communicate with each other. Hence, in our project, we study the MPEG-4 IPMPX system, which is a Digital Rights Management interface defined by MPEG. Our goal is to design a DRM system that can provide functionalities of Content Protection and Rights Management.

Hence, we first study the MPEG-21 IPMP and REL. MPEG-21 IPMP provides a high level protection of Digital Items. MPEG-21 REL is able to describe various rights expressions, and to provide right authorization and control. Because both IPMP and REL are conceptual entities, we choose MPEG-4 IPMPX system as a set of concrete interface specifications for our system. We design MPEG-4 IPMPX compatible tools incorporated with characteristics of MPEG-21 IPMP and REL. We also design an encryption/decryption tool to perform content protection. In addition, we propose a content protection mechanism that combines with the key management for higher security.

Finally, we design three application examples to demonstrate the use of our DRM system. The example shows the management of play right in a real-time streaming system. The second one shows a system that allows users to preview a multimedia program securely. The last one shows the rights management incurred in the distributed mobile device. In this case, we also design a license to perform online authorization as well as offline verification.

Although we have developed a DRM system which safeguards content and manages the rights transactions, there are quite a few related DRM topics can be further studied, such as to scalable media protection and the interoperable DRM solution.

References

[1] The open Digital Rights Language Initiative, last updated 2006-06-05. http://odrl.net/

[2] Text of ISO/IEC 21000-5 FCD－Part 5: Rights Expression Language, ISO/IEC JTC 1/SC 29/WG 11/N5349, December 2002, Japan.

[3] The Open Mobile Alliance. http://www.openmobilealliance.org/

[4] The Innovative Rights and Access Management Inter-platform Solution. http://www.tiramisu-project.org/

[5] J. Bormans, and K. Hill, "MPEG-21 Overview v.5," ISO/IEC JTC 1/SC 29/WG 11/N5231, Shanghai, October 2002.

[6] Study of ISO/IEC 21000-4 FCD－IPMP Components, ISO/IEC JTC 1/SC 29/WG 11/N7426 July 2005, Poznan, Poland.

[7] C.A. Schultz, "Study of FPDAM ISO/IEC 14496-1:2001 / AMD3," ISO/IEC JTC 1/SC 29/WG11 N4849, Klagenfurt, July 2002.

[8] 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.

[9] 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.

[10] ContentGuard, *"MPEG REL SDK for JavaTM Software Development Kit User's Guide Release 1.0."*

[11] X. Wang, T. DeMartini, B. Wragg, M. Paramasive, and C. Barlas, "The MPEG-21 Rights Expression Language and Rights Data Dictionary," *IEEE Multimedia*, vol. 7, no. 3, pp. 408-417, June 2005.

[12] W3C. (1999) Namespaces in XML, http://www.w3.org/TR/1999/REC-xml-names-19990114/

[13] W3C. (2001) XML Schema, http://www.w3.org/TR/2001//REC-xmlschema-1-20010502/

[14] I. S. Burnett, S. J. Davis, and G. M. Drury, "MPEG-21 Digital Item Declaration and

Identification — Principles and Compression", IEEE TRANSACTION ON MULTIMEDIA, VOL. 7, NO.3, JUNE 2005.

[15] *Information and Documentation — The Dublin Core Metadata Element Set*, ISO 15836:2003, Nov. 2003.

[16] *Information Technology — Multimedia Framework (MPEG-21)-Part 3: Digital Item Identification*, ISO/IEC 21000-3:2003, Mar. 2003.

[17] B. S. Manjunath, P. Salembier, and T. Sikora, *Introduction to MPEG-7.* Chichester, U.K.: Wiely, 2002.

[18] *Information Technology — Multimedia Framework (MPEG-21)-Part 2: Digital Item Declaration*, ISO/IEC 21000-2:2003, Mar. 2003.

[19] *Information Technology — Multimedia Framework (MPEG-21)-Part 7: Digital Item Adaptation*, ISO/IEC 21000-7:200X.

[20] S. Lauf, and I. Burnett, "A Protected Digital Item Declaration Language for MPEG-21," IEEE Computer Society, AXMEDIS, 2005.

[21] ISO/IEC 14496-1:2004, Information technology -- Coding of audio-visual objects -- Part 1: Systems Nov. 2004.

[22] C.W. Fan, F.C. Chang, and H.M. Hang, "An MPEG-4 IPMPX Design and Implementation on MPEG-21 Test Bed", ISCAS, Vol. 5, May. 2005

[23] C.W. Fan, "MPEG-4 IPMPX Design and Implementation on MPEG-21 Test Bed," M.S. thesis, Dept. Electrical Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., June 2004.

[24] 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.

[25] 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.

[26] 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.

[27] J. Bormans and K. Hill, "MPEG-21 Overview v.4," ISO/IEC JTC1/SC29/WG11 N4801, Faifax, May 2002.

[28] J. King, et al. "MOSES Progress report on MPEG-4 IPMPX," ISO/IEC JTC 1/SC 29/WG11 M9161, Awaji, Dec 2002.

[29] M. Carson and D. Santay, “NIST Net – A Linux-based Network Emulation Tool,” ACM SIGCOMM Computer Communications Review, Volume33, Number3, July 2003.

[30] X. Wang, et al., “An Exmple Implementation of MPEG-21 REL Reference Software,” ISO/IEC JTC1/SC29/WG11 MPEG2003/M9581, March 2003

[31] An Overview of Cryptography. http://www.garykessler.net/library/crypto.html

自　　傳

呂家賢，西元 1982 年出生於台北縣。2004 年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研究所攻讀碩士學位。以智財權保護及管理系統與權利描述語言為研究主題。