Extending an MPEG-21 Viewer to Manage Access Rights

Rickard Lönneborg

January 13, 2004

Abstract

The Internet has opened many opportunities for the distribution of information, both free and copyrighted. A music album is quickly transferred with high quality between Internet users, peer-to-peer networks enable simple trading and redistribution of copyrighted music, video and programs. Digital Rights Management (DRM) systems are designed to enable legal trading of copyrighted information, prevent unauthorised distribution, modification and usage of copyrighted digital information.

MPEG-21 is a new standard for multimedia on the Internet. It is described as being the "big picture" in describing multimedia content. It allows for composition of *digital items* from different resources such as pictures, sounds, etc. This standard is still under development by the Moving Pictures Expert Group (MPEG). MPEG-21 Intellectual Property Management and Protection (IPMP) is a framework for enabling DRM in MPEG-21. IPMP enables a MPEG-21 terminal to communicate through messaging with various IPMP-tools to protect MPEG-21 content.

A Rights Expression Language is a language capable of describing users rights to some digital information. One part of MPEG-21 is the Rights Expression Language, which will be based on *The eXtensible rights Markup Language*.

The SMICL viewer is a viewer for MPEG-21, it is under development and is supposed to be a research platform for security related research.

In this thesis we review some of the current systems for DRM and present some Rights Expression Languages. Further we describe MPEG-21, and the SMICL viewer and its architecture. We discuss refinements of the IPMP messaging for the SMICLV viewer and the required behaviour of a rights enforcing IPMP tool.

Finally we provide an implementation of a rights enforcing IPMP tool for the SMICL viewer, and propose future extensions for the SMICL viewer and the rights enforcing tool.

Acknowledgements

I would like to thank my supervisors Professor Reihanheh Safavi-Naini, Dr. Nicholas Sheppard, and my fellow student Qiong Liu, for supporting me with a lot of constructive criticism and suggestions. The work of the author is partially supported by the CRC for Smart Internet Technlogy.

I would also like to thank my supervisor Ola Ågren at Umeå University for the positive feedback given.

Contents

1 Introduction **1**
1.1 Objective 1
1.2 Introduction 1
1.3 Contributions and structure of this thesis 2

2 Basic Concepts **4**
2.1 XML 4
2.1.1 Syntax 4
2.1.2 XML Schema 5
2.1.3 XML Namespaces 5
2.2 Cryptographic Systems 5
2.2.1 Symmetric key systems 6
2.2.2 Asymmetric key systems 7
2.2.3 Digital signatures 8
2.3 User authentication 8
2.3.1 Password systems 8
2.3.2 Challenge-Response systems 8
2.3.3 Machine vs User Authentication 9
2.4 Digital Watermarking 9
2.5 Digital Rights Management Systems 10
2.6 Copyright Laws 10
2.6.1 Fair Use 10
2.6.2 USA 11
2.6.3 Australia 11
2.6.4 European Union 12
2.6.5 Sweden 12

3 Digital Rights Management **14**
3.1 Overview 14
3.2 DRM Systems 15
3.2.1 Microsoft DRM 15
3.2.2 IBM Cryptolope 17
3.2.3 IBM Electronic Media Management System 19
3.2.4 Intertrust DRM 20

3.2.5 Summary 21

4 MPEG-21 **22**
4.1 Digital Item Declaration 23
4.1.1 Container 23
4.1.2 Item 23
4.1.3 Descriptor 23
4.1.4 Component 23
4.1.5 Resource 24
4.2 Intellectual Property Management and Protection 24
4.2.1 Terminal 24
4.2.2 Tool 24
4.2.3 Tool manager 25
4.2.4 Messaging Interface 25
4.3 Digital Item Identification and Description 26
4.3.1 Goals 26
4.3.2 Syntax 26
4.4 Rights Data Dictionary and Rights Expression Language . . . 27
4.5 Digital Item Adaption 27
4.6 MPEG-21 Example 27

5 Rights Expression Languages **30**
5.1 XrML 30
5.2 XMCL 32
5.3 ODRL 32

6 SMICL MPEG-21 Viewer **34**
6.1 Introduction 34
6.2 Functionality 35
6.3 Architecture 35
6.3.1 wxWindows 36
6.3.2 libXML 36
6.3.3 DIDL Interpreter 36
6.3.4 IPMP 36
6.3.5 IPMP Tools 36
6.3.6 GUI 37
6.4 Classes 37
6.4.1 smiclvApp 38
6.4.2 smiclvStringTable 38
6.4.3 smiclvFrame 38
6.4.4 ipmpTerminal 38

6.4.5 ipmpToolManager . . . 38
6.4.6 messageRouter . . . 38
6.4.7 diWindow . . . 38
6.4.8 diDeclaration . . . 39
6.4.9 diElement . . . 39
6.4.10 smiclURL . . . 39
6.4.11 diTransport . . . 39

7 Extensions to the SMICL Viewer 40
7.1 Introduction . . . 40
7.2 XrML Software Development Kit . . . 40
7.3 System Architecture . . . 42
7.4 License Interpreter plugin . . . 42
7.5 Condition Validation . . . 43
7.6 Class Overview . . . 43
7.6.1 License handler class . . . 44
7.6.2 License interpreter plugin . . . 45
7.6.3 Condition Listener . . . 45
7.7 Extending The Viewer . . . 46
7.7.1 XrML license transfer . . . 46
7.7.2 Effects of including license in DID . . . 46
7.7.3 IPMP communication . . . 47
7.8 IPMP Messages . . . 48
7.8.1 Messages of the SMICLV IPMP Terminal . . . 49
7.8.2 REL Plugin handled messages . . . 50
7.9 Algorithms . . . 52
7.9.1 Intent-request handler . . . 52
7.9.2 License Interpretation . . . 52
7.9.3 Condition validation . . . 53

8 Conclusion 54
8.1 Functionality . . . 54
8.2 Security . . . 54
8.3 Future work . . . 55

9 Glossary 57

A SMICLV MPEG-21 IPMP 62
A.1 MPEG21-IPMP.h . . . 62

B REL Plugin **76**
B.1 Rights.h 76

List of Figures

2.1 Small XML example 5
2.2 XML Namespace example 6

3.1 Microsoft DRM overview 16
3.2 Example of a cryptolope, from [2] 18
3.3 IBM EMMS components 19

4.1 MPEG-21 IPMP Architecture [10] 25
4.2 MPEG-21 example, including license entry 28

5.1 Example XrML license 31
5.2 ODRL simplified example 33

6.1 SMICLV User interface 34
6.2 SMICL Viewer overview 35
6.3 Simplified class overview of the SMICL viewer 37

7.1 The XrML SDK architecture 41
7.2 XrML Namespaces 42
7.3 REL plugin overview 42
7.4 REL plugin classes 43

Chapter 1

Introduction

1.1 Objective

We will look at current systems for Digital Rights Management (DRM) and observe the main common features to gain a better understanding of DRM systems. Further, we want to see whether or not a DRM system can be implemented with the MPEG-21 framework. The focus is on the Rights Expression Language in MPEG-21, to describe different users rights to a content.

The objective is to design a DRM system that conforms with the MPEG-21 framework and to extend an existing viewer of *Digital Items* to enforce some basic rights.

1.2 Introduction

Huge amounts of information is easily available on the Internet. This information can be free or copyrighted. The digital nature of information makes it very easy to copy and redistribute an object, both legally and illegally. Digital Rights Management (DRM) systems are created to help stop the unauthorised distribution of digital information and to encourage the legal trading of information on the Internet.

MPEG-21 is a new standard for multimedia on the Internet. It is described as being the "big picture" in describing multimedia content. It allows for composition of *digital items* from different resources such as pictures, sounds, etc. This standard is still under development by the Moving Pictures Expert Group (MPEG).

The *digital item* is central to MPEG-21 and is an abstraction and extension of the actual resources. A digital item may consist of a film clip, with selections of picture quality, or it may be a picture with associated information about the producer and the origin.

MPEG-21 is meant to be open and platform independent and will allow for digital items to be adapted to the viewing environment.

> The vision for MPEG-21 is to define a multimedia framework to enable transparent and augmented use of multimedia resources across a wide range of networks and devices used by different communities [7].

The SMICL Viewer[1] has been developed by Nicholas Sheppard in the SMIC Lab of the University of Wollongong.

It has functionality for viewing items, and support for the MPEG-21 framework for Digital Rights Management called *Intellectual Property Management and Protection (IPMP).*

1.3 Contributions and structure of this thesis

In this thesis we review some existing systems for Digital Rights Management, some languages for describing rights, and further we provide an implementation of a tool for enforcing rights for digital items. We discuss the impact of including rights expressions in a digital item, and how the communication may be defined between a rights enforcing tool and a digital item viewer.

We propose a behaviour rule for a rights enforcing tool, and suggest definitions for some MPEG-21 IPMP messages. The implementation part of this thesis is a rights enforcing tool for the SMICL viewer, and extensions to the current implementation of the IPMP. The rights enforcing tool will interpret licenses in the eXtensible right Markup Language (XrML), and enforce the right in these licenses.

The first chapter describes some basic concepts in relation to this thesis and the Markup Language called the *the Extensible Markup Language (XML)* is introduced. This chapter also describes some cryptographic and watermarking systems that are important components of a Digital Rights Management system. The chapter also provides a short introduction to copyright laws and examples of these laws in USA, Australia, and Sweden.

The next chapter introduces Digital Rights Management (DRM), a general description of a basic DRM system, followed by review of some present systems.

Chapter four gives an introduction to MPEG-21, the digital item, and descriptions of the defined parts of MPEG-21. These are the *Digital Item*

[1]SMICL viewer has been developed as part of the project *Collaborative Watermarking* which is supported jointly by Motorola Australian Research Centre and Australian Research Council.

Declaration that gives the basic structure of the digital item, the *Intellectual Property Management and Protection* that protects the items, the *Digital Item Identification and Description* for giving identifications to the digital items, the Rights Data Dictionary and Rights Expression Language describing rights for users of the item. The last part called *Digital Item Adaption* describes how to adapt digital items to network and viewing terminal environments.

Chapter five gives a short description of some *Rights Expression Languages*, used to express complex rights information.

Chapter six provides a description of the SMICL Viewer, the IPMP messages handled by the viewer and the classes of the program.

The final chapter describes the implementation of the Rights enforcing tool and the issues that have arisen during the implementation of this tool. This chapter also has suggestions on further work with this rights enforcing tool.

Chapter 2

Basic Concepts

This section describes a few of the basic concepts related to this thesis. First we give a short introduction to the eXtensible Markup Language used in the Rights Expression Languages described in the following chapters. Then we give a brief introduction to cryptographic systems authentication systems, as these systems are authentication are seen in all of the DRM systems covered. Finally we present a brief example of laws affecting the current DRM technologies.

2.1 XML

The Extensible Markup Language (XML) [17] is a popular language for creating structured documents on the web. It is a text based language similar to HTML (HyperText Markup Language), but is for more general purpose. It provides a tree-based structure for platform-independent data representation. XML is the basis for many different descriptive languages including *Rights Expression Languages* such as XrML [23] and ODRL [25], message passing for distributed systems like SOAP [13] and media formats such as MPEG-21 [4].

2.1.1 Syntax

The language syntax itself is rather simple. It is comprised by matching tags, a tag is an identifier may look like '`<xmltag>`', starting with the symbol '`<`' and ending with the symbol '`>`'. Tags always need an end-tag, which is the same identifier as the start tag but starts with '`</`' instead of '`<`'. The corresponding end-tag to the previous tag would be '`</xmltag>`'. Tags may be nested, and attributes may be associated with tags. Eg. '`<xmltag size = "12">`'. Tags may also be self-ending, in which case they do not need an end-tag. In this case there is a '`/>`' at the end of the tag. Eg. '`<xmltag/>`'.

```
<box>
  <minibox>
    <textbox color="blue">
     My blue textbox
    </textbox>
  </minibox>
</box>
```

Figure 2.1: Small XML example

2.1.2 XML Schema

To use XML as a base for describing data, the tag names must be defined for the data we are describing. To do this XML Schemas are often being used [16]. XML Schemas are XML documents describing correct syntax of a XML-based language. Another syntax description language for XML is Data Type Definition (DTD), which is an XML definition language not based on XML itself. This might be a reason why it is becoming less popular in favour of XML Schemas. Many XML parser use XML Schemas to verify XML documents.

2.1.3 XML Namespaces

To determine what schema is to be used for a particular XML expression and to mix expressions from several XML-languages XML namespaces are available. To denote that a tag belongs to a specific namespace a namespace identifier and a colon, :, is added to the tag. Example `<myNameSpace:xmltag>`. To assign a schema to a namespace the attribute `xmlns= "URI"` is added to the root-tag of an XML document. See Figure 2.2 on the following page.

2.2 Cryptographic Systems

Cryptography is the scheme of protecting data and ensuring that the data is kept secret and is not tampered with. Cryptographic systems are used to ensure *secrecy* and *authenticity* through encryption and authentication systems.

Encryption is a key dependent function $C = E_K(P)$ that transforms a plain text P to a cipher text C. The ciphertext is not readable for an unauthorised user who does not have access to the correct key. *Decryption* is the corresponding function for transforming ciphertext back to the original clear text. We represent the function as $P = D_{K'}(C)$, where P is the original

```
<licenseGroup
xmlns="http://www.xrml.org/schema/2001/11/xrml2core"
xmlns:cx="http://www.xrml.org/schema/2001/11/xrml2cx"
>
<!-- the current namespace is xrml2core -->
...
<cx:locator>
<!-- this expression belongs to the xrml2cx namespace-->
...
</cx:locator>
</licensegroup>
```

Figure 2.2: XML Namespace example

plain text, and C is the cipher text. If a message is to be sent from one user to another, encryption and decryption provides secrecy in the sense that an interception of the message in ciphertext will not result in the correct cleartext.

Digital signatures are use used to ensure authenticy of messages and data, with the help of cryptographic systems. A signature may be created to ensure that a message is *not fabricated* and *not modified*.

Some encryption algorithms depend on a *key* K to transform the plain text to ciphertext. This key is usually a value, a sequence of values, or a password string. The same key, or a corresponding key K' may be used to decrypt the ciphertext.

2.2.1 Symmetric key systems

In a symmetric key encryption system, the *same* key is used for both encryption and encryption. In such systems, we define the encrypting function as $C = E_K(P)$, and the corresponding decryption $P = D_K(C)$.

An example of a simple encryption is the *substitution* encryption. In this system every character in the source text is substituted with a corresponding character from a lookup-table. For example, all 'A' characters in the source text may be substituted with a 'C'. The authorised viewer of this text should have a corresponding lookup-table and translate all 'C's to 'A', etc. to make the text readable.

Another example of a simple cryptosystem is the *vigenére* system. In this system, we use modular addition using a keyword to obtain the ciphertext. The decrypting user has the same keyword and can generate the key sequence that is subtracted from the encrypted text.

Examples of modern symmetric cryptosystems are DES (Data Encryption Standard) and AES (Advanced Encryption Standard).

One common problem with private key encryption is that the key must be shared between both the encrypting party and the decrypting party. If the encrypting party wants to send the key to the decrypting party, there might be a risk of interception if the channel is insecure. The Internet should in general be considered an insecure channel.

An exhaustive search attack is when an attacker tries all possible keys to decrypt a ciphertext. If the computation cost of this attack is high, the system is considered secure.

2.2.2 Asymmetric key systems

Instead of using the same key to encrypt and decrypt the data, asymmetric key systems use a *public* and a *private* key. These systems are usually based on the idea that if user Alice wants to communicate with Bob, Bob sends his public key to Alice, with which she encrypts the information as $y = e_K(x)$. Bob decrypts using his secret decryption key $x = d_K(y)$. For these systems, the cleartext may be derived knowing e_K and the encrypted text. An attack is to encrypt every cleartext x using $y = e_K(x)$ until the matching ciphertext y is found, and if this operation is computationally expensive, the system is considered secure.

Examples of Public-key systems are Rivest-Shamir-Adelman (RSA) [12] and ElGamal [12].

RSA

RSA is the most important public key cryptosystem, its security is based on the complexity in factoring large integers. The system is defined as follows: Let $n = pq$, where p, q are primes. Let ab $\equiv (1 \bmod \varphi(n))$, where $\varphi(n) = (p-1)(q-1)$. Encryption is defined as

$$E(x) = x^b \bmod n$$

and decryption

$$D(y) = y^a \bmod n$$

where x is the cleartext and y is the encrypted text. The values of n and b are public, and the values of p,q and a are private.

The public key K_E is the pair (n, b), and the corresponding private key K_D is the pair (a, n). The values p and q may not be disclosed.

If a user "Alice" wants to send an encrypted message to "Bob", Bob sends his public key K_E to Alice, Alice uses this key to encrypt $C = E(K_E, P)$ and Bob can decrypt using his private key K_D, $P = D(K_D, C)$.

2.2.3 Digital signatures

To authenticate a digital object, such as an image, an e-mail or a program digital signatures can be applied. Digital signatures show that the data received is not fabricated, or has been modified.

Digital signature schemes usually operate on the *hash value* of the data being signed. A *hash function* is a function that takes an arbitrary length input and produces a fixed length output, which will be signed by the signature algorithm.

The RSA algorithm can be used to create a digital signature. The scheme is :

$$sig_K(x) = x^a \bmod n$$

and

$$ver_K(x, y) = \text{true} \leftrightarrow x \equiv y^b \pmod{n}$$

where p, q, a are secret, and b, n are public. x may be the message itself but it is more efficient to use the hash value of the message.

This scheme is similar to the RSA encryption, but in this case, the encryption is done with the secret key instead of the public key, and the decryption is done with the public key.

2.3 User authentication

Most computer systems, online stores, etc. need to know that a user is the one that he/she claims to be. *User authentication* are schemes to establish that a users identity is true.

2.3.1 Password systems

A common authentication system is based on the human memory, a person is authenticated through a (username,password) pair [12]. The username is not always kept secret and is often related to the user. The password is usually a short text that is easy to remember. If a user discloses this password, or if it is intercepted, a malicious user can impersonate him. Password systems may be vulnerable to guessing attacks, since the password is quite often a real word, and not just a random sequence of characters.

2.3.2 Challenge-Response systems

In *challenge-response systems* [12], passwords are one-time only. A challenge-response system is relies on the user having a function, instead of, or in conjunction with a password. The system provides an argument to the function,

and the user responds with the function value. These functions can be kept inside physical devices such as specialised calculators that when given the input, the output is returned. A commonly used device for implementing challenge-response systems is *smart cards*.

2.3.3 Machine vs User Authentication

Authentication may be based on the computer itself, with the user having no part. Encryption keys or challenge-response functions may be tied to the computer, as in Microsoft Media DRM system [29]. A computer can for example be authenticated from the network cards (Media Access Control) MAC-address. This is Machine based authentication. User based authentication relies on that the user has a password, a fingerprint, or that he/she is equipped with a specialised device such as a smart card.

Machine based authentication is tied to a computer, whereas user based authentication relies on the user, a feature of the user or device in possession of the user.

2.4 Digital Watermarking

Watermarking is a way to embed hidden information into a digital object, for different purposes such as authentication and discovery of the source of an object. Physical watermarks are often used to authenticate money. Digital watermarks are concerned with multimedia data, such as images, sound recordings and video.

A digital watermark is hidden within the media, the requirements of the watermark is that it should be invisible and difficult to remove. Furthermore, detection should be fast and cheap [5]. Digital watermarks can be classified into *robust*, meaning that the watermark can withstand modifications, filtering and compressions, or it may be *fragile*, where a small modification destroys the watermark. A watermark *for protecting copyright* should be *robust*, to withstand simple processing of the media such as compression, noise addition, logo insertions, shifts and format conversions. Application areas for fragile watermarks include medical images, where it is important that the image needs to be verified for not being edited, damaged or altered [6].

The probability of *false alarm* should also be extremely small [5].

Robust watermarks is an active research are. A robust watermarking scheme can be used for:

- **Copyright protection**: the origin of an image can be established and credit can be given to the owner.

- **Fingerprinting**: multimedia data can contain information about the receiving machine, and this can help identify a perpetrator.
- **Image annotations**: information about the author, date, etc. can be embedded in the image.

2.5 Digital Rights Management Systems

Digital Rights Management systems are tools and technologies for securely distributing digital assets, and to maintain copyright protection for digital content. DRM systems use cryptography, signatures and authentication schemes to ensure that digital content is received by the right user, and used according to the usage rules specified by the copyright-holder. DRM systems will be covered in chapter 3 on page 14.

2.6 Copyright Laws

There are copyright laws to protect a creation, for example a program, a piece of music, or a movie, from being illegally distributed, but at the same time consumers should have the right to use their purchased products in a reasonable way. The World Intellectual Property Organization (WIPO) [34] is an international organisation working with Intellectual Property, Patent and Copyright related issues. It is part of the United Nations organisation and currently has 179 member states. 35 Countries have signed the WIPO Copyright Treaty (1996), 34 the WIPO Performances and Phonograms Treaty (1996), the former concerning copyright, including software, and the latter more focused on music.

2.6.1 Fair Use

The question of *fair use* is often being raised by critics of DRM systems, *fair use*(US) or *fair trading*(Australia), in the context of copyright protection and consists of a number of exceptions in the copyright laws. In Australia, copyright is not breached if copies are made for :

- research or study
- criticism or review
- reporting of news

- professional advice given by a legal practitioner or patent attorney

In DRM systems, restrictions can be made in a way that generally does not apply to printed mediums such as books. A DRM system can restrict all copying of some content, whereas traditional books and music may be easy to copy for personal use. DRM systems may *disable* the possibility of copying for *fair use*. This has "come to stand in as a general expression for the user-side of the copyright fence" [36].

2.6.2 USA

A famous American law concerning computer related issues is the Digital Millennium Copyright Act (DMCA) [35]. This law claims to implement the WIPO copyright treaty. It adds restrictions such as:

> Section 1201 divides technological measures into two categories: measures that prevent unauthorized access to a copyrighted work and measures that prevent unauthorized copying of a copyrighted work. 2 Making or selling devices or services that are used to circumvent either category of technological measure is prohibited in certain circumstances, ... [35]

This law has been criticised for its restrictions. Some organisations claim that it will halt security research since researchers are not allowed to create and distribute code that removes protection from a system.

> The law provides certain exemptions for research, however they are unclear and must be approved by the Corporation who created the software [37].

2.6.3 Australia

Australia has its own *Copyright Amendment (Digital Agenda) Act 2000 (CADA)* [39], which is similar to the US Digital Millennium Copyright Act, but it is not claimed as an implementation of the WIPO Copyright treaty. This copyright reform defines penalties for the following actions:

- The circumvention of copyright protecting devices
- Removal and alteration of digital rights information
- Unauthorised digitisation

- Manufacture, sales and dealing with unauthorised broadcast decoding devices

The first point relates to both devices and computer programs designed to circumvent copyright protection systems. It is illegal to manufacture, sell, import and advertise such devices, but not to use such a device. The exception to this is interpreted as follows.

> The permitted purposes specified in the CADA are the reproduction of computer programs to make interoperable products, to correct errors and for security testing, activities covered by libraries (including Parliamentary libraries) and archives exceptions, the use of copyright material for the Crown, and activities covered by the statutory licences for educational institutions and institutions assisting persons with a disability under Part VB of the Copyright Act [39].

As this states, it is permitted to circumvent copyright protections for interoperability purposes, which is not mentioned in the DCMA.

2.6.4 European Union

The European Union (EU) member states have agreed on a copyright related directive [38] that needs to be implemented in the laws of the member states within 18 months of the agreement date.

This directive requires the European states to implement laws restricting *unauthorised copying or reproduction* of works. A number of exceptions *may* be allowed by the member states. Examples of such exceptions are reviews, research, teaching, news reports, uses for disabilited, religious celebrations, caricatures, etc.

This directive also, like the DMCA, restricts tampering with mechanisms for removing copy protections, and rights information protecting works. It also forbids import, export and advertisement of devices, products or components created for these purposes.

2.6.5 Sweden

It is legal to make copies of copyrighted works (except for computer software), to oneself, family and close friends [33].

There was one interesting case concerning distribution of MP3-files on the Internet. A young man maintained a website with links to other servers

containing copyrighted music. The young man was declared innocent, since a clause regarding "performing" recorded music to the public is not a crime [33], though the law states that the economic compensation should be made to the artists.

Sweden is covered by the EU-directives, so the Swedish laws will have to reflect the new copyright EU-directive [38], before or during 2003.

Chapter 3

Digital Rights Management

3.1 Overview

> Digital Rights Management (DRM): can be broadly defined as the concepts, systems, and functions that enable content owners to securely distribute various forms of digital assets, maintain visibility to its creators, and determine the means in which that content can be used, reused, purchased, copied, and distributed by its users [19].

Digital Rights Management (DRM) is a term for technologies and tools protecting the copyright of digital content. These technologies are meant to support creators of digital content to publish on the Internet and to minimise the illegal spreading of their content. Encryption and watermarking are commonly used components of these tools. Encryption protects the content from being consumed by an unauthorised user, and watermarking may be a way to establish the source of some content, or to whom the content has been sold, etc.

Most systems consist of client side software and server software, and communication between these two systems. The usual task of the client side software is to :

- Retrieve content
- Retrieve usage rules and/or access information (encryption keys, etc.)
- Enforce usage rules
- Display content

The main responsibilities of the server systems in DRM are:

- Deliver content
- Communicate usage rules and access information to client
- Handle transactions, payments, etc.

Questions and problems in DRM systems in general are *fair use* problems and the risk of software tampering or reverse engineering of the DRM systems. Many of todays DRM systems have limitations in the transfer of digital content, it may for example only be played on the desktop computer.

3.2 DRM Systems

There are a number of existing DRM systems, some reviewed briefly in this chapter. Some key features are found in all of these systems. They are all three-part systems, a *client program*, a *content server* and a *license server* or *clearinghouse*. In these systems the content server delivers encrypted content, the clearinghouse delivers a license to the client and the client decrypts the content and enforces the usage rules.

3.2.1 Microsoft DRM

Overview

The Microsoft DRM utilises a form of *machine based authentication*, content is supposed to be viewed on one computer only. The client viewer is *individualised* on installation, creating encryption keys for the client viewer. As a user wants to view protected content, these steps are performed:

- The client sends a request to the clearinghouse for a license to the content.
- The clearinghouse requests identification of the client to verify that it is a known authorised viewer. (And not a forbidden third-party viewer or a viewer that has been compromised)
- The clearinghouse creates an individual license for the viewer, tied to the user's computer.

A license may also be delivered along with the media, so that no negotiation is needed.

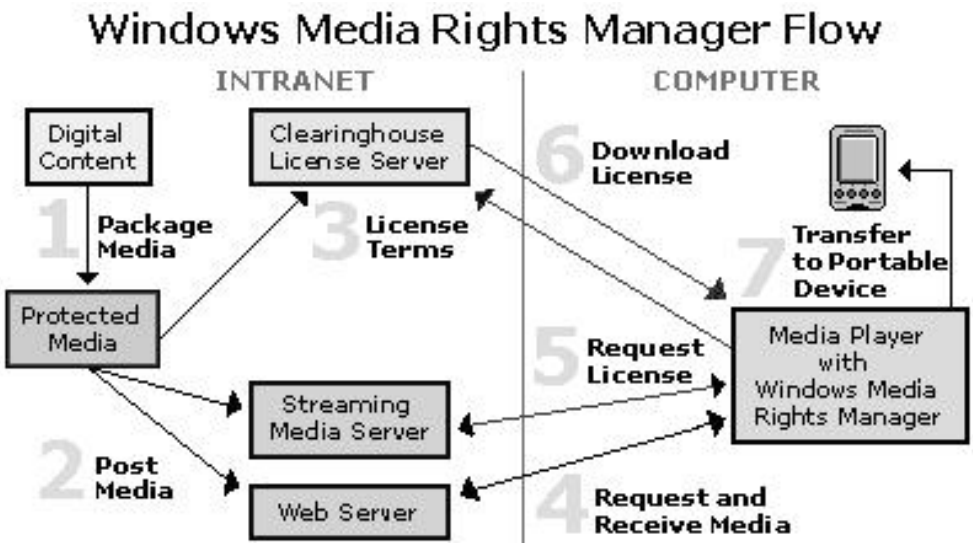

Figure 3.1: Microsoft DRM overview

A schematic view of the Microsoft DRM system is seen in figure 3.1.

Some sound files may require a *secure audio path*, that is an encrypted delivery of the sound to the audio card, so that no interception of the sound signal within the computer is made. This feature is implemented in some sound cards and the Microsoft Windows XP operating system.

According to [28], Microsoft confirms that a version of their DRM system has been reverse-engineered, and that the source code for a program that removes the protection from an encrypted media file is being spread on the Internet. This code was posted in the Usenet group sci.crypt, by a user with alias Beale Screamer. This person or organisation claims that Microsoft DRM does not allow fair use of acquired media files.

Availability

The Microsoft Media DRM system is compatible with any media player that supports use of the Microsoft DRM manager.

Microsoft supplies a Software Development Kit (SDK) for their DRM system. It is available from the Microsoft website [29].

3.2.2 IBM Cryptolope

Overview

Cryptolope is a *cryptographic envelope* protecting various digital content. The cryptolope system created by IBM features a Java client tied to the user's web-browser (Netscape Navigator 4.02, Windows 95/NT). It is only intended for media that can be viewed using this web-browser. The content is delivered as a cryptolope; a digital container containing the protected document, the authentication information and the user's usage rights to the document.

The DRM system for the Cryptolope consists of a clearinghouse, a certificate authority and a user client. The cryptolope itself may be transported on the Internet, or it might be copied on CD-ROMs etc. It is not dependent on a secure transport.

The cryptolope itself is a JAR file (Java Archive) that carries the content and related information. It contains:

- The content itself
- Usage rules, permissions
- Digital signature
- Encrypted keys for decryption of the content

The certificate authority encrypts, signs and packages the content.

The keys to decrypt the content are stored encrypted **within** the cryptolope. In this way the clearinghouse does not need to know the decryption keys of all content. Instead, the clearinghouse decrypts the keys from the cryptolope, and supplies them to the decrypting system (the client).

The opener of a cryptolope executes code in the cryptolope itself to gain access to the content.

To ensure that the cryptolope has not been tampered with, it is signed by the certificate authority.

To prevent the illegal redistribution of the viewed media, the player does not permit saving of the viewed content, if not otherwise stated in the permission part of the cryptolope. One further action to prevent screen captures and similar copying is watermarking. This watermarking is carried out by the client software through encrypted executable java-code included in the cryptolope. The watermark can be made to identify the user of the content.

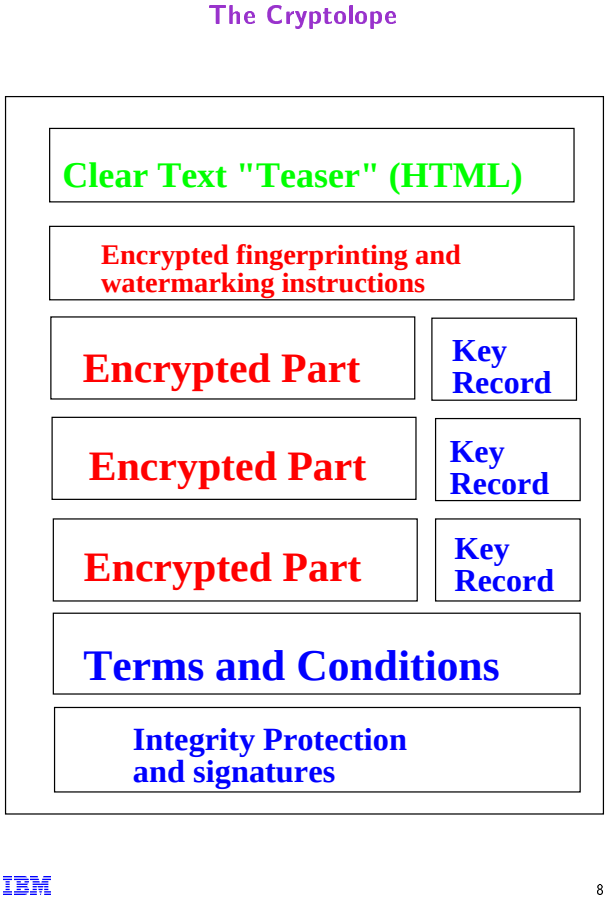

Figure 3.2: Example of a cryptolope, from [2]

Availability

The IBM Cryptolope player is supplied by IBM. It is free for download from their web page and it runs on Windows platforms, with the Netscape browser.

Conclusions

The system itself is well designed, but it lacks interoperability. The only supported platform is Microsoft Windows with Netscape. Furthermore it only supports pictures and text.

3.2.3 IBM Electronic Media Management System

A newer system than the Cryptolope from IBM is the Electronic Media Management System (EMMS). This system consists of seven parts shown in figure 3.3. The system is intended to protect all types of digital content, and a Client Software Development Kit (SDK) is provided for enabling handling of new content types other than those already supported.

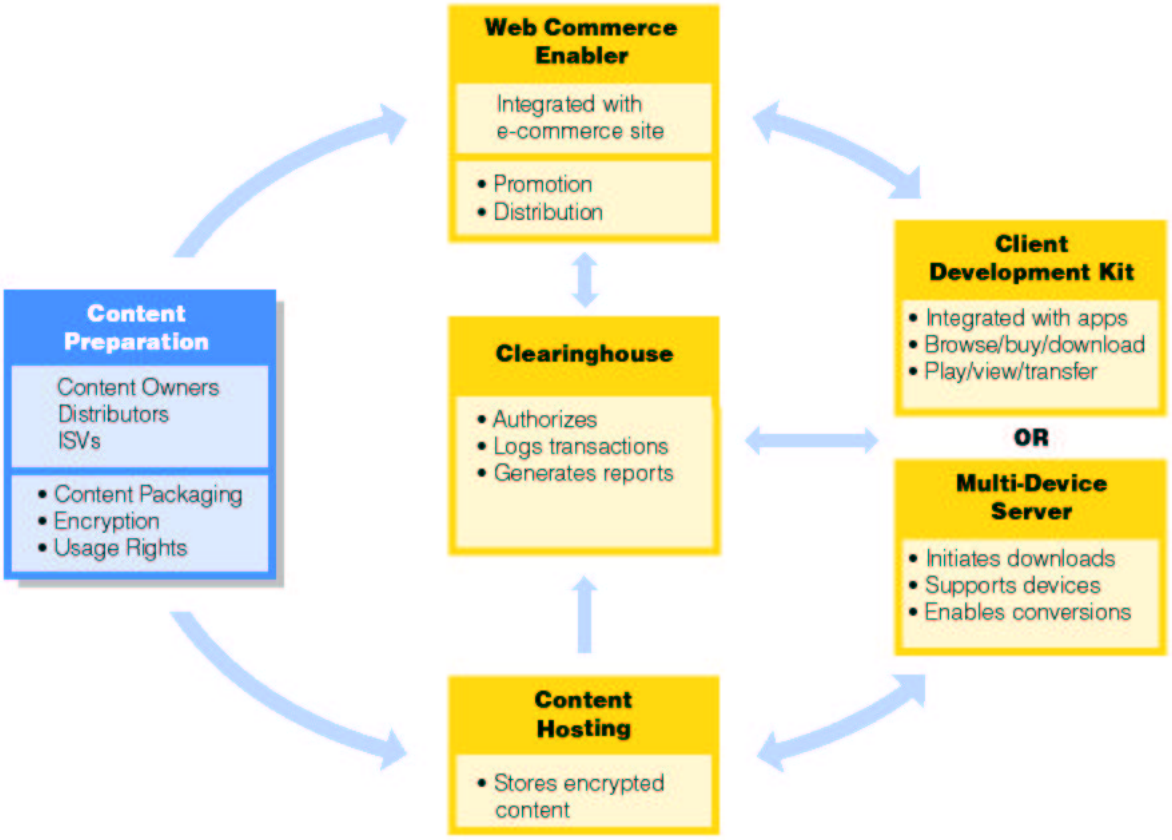

Figure 3.3: IBM EMMS components

The system provides functionality for packaging media, delivering and enforcement of usage rules. These rules can be pay-per-view, time restraints, usage restraints or permissions to transfer the content to other devices such as CD-R or trusted mobile devices.

Availability

The content management and server software part all run on Windows 2000/NT, while the client SDK runs on all desktop Windows platforms, except Windows 95 and earlier versions. The content mastering system supports a number of different encodings such as Realnetworks G2 and other.

There are a number of websites, mainly in Japan [26], selling music using the EMMS system.

The fact that this system only supports Windows platforms may be a drawback for both content-providers and consumers.

3.2.4 Intertrust DRM

Intertrust is a USA based company working with Digital Rights Management systems and security. Their main product is the *Rights|System*, which is a DRM system consisting of server software, clearinghouse and client side software. Intertrust is also the owner of Star-Lab [30] and the company holds 24 US patent covering content protection and DRM-related technologies. The company has recently gone through a down-sizing, removing their products from the market. A new licensing agreement has been made with Sony so the company may survive on revenues from their patents. Intertrust also holds a number of lawsuits against Microsoft regarding Intertrust patents, claiming that many Microsoft products are based on work from Intertrust.

The Intertrust Rights|System is the main product, it consists of a number of content packagers, content servers, client programs and toolkits.

Usage rules are being defined using XML, and the packagers encrypt the content and create a *Rights|packs* for the content, a Rights|pack is a package of usage rules and decryption keys to the content.

After the content has been packaged, the encrypted content may be sent to a media delivery server. The Rights|pack is sent to a *Content Rights Server*. Content Rights Server software is provided by Intertrust.

An example walkthough for the user in the Intertrust Rights|System:

- The consumer purchases the content through an existing (non-Intertrust) vendor system;
- The vendor system sends an "authorisation" to the user;
- The user client uses this "authorisation" to receive a Rights|pack for the content;
- The user downloads the content; and
- The user client consumes the content according to the rules specified in the Rights|pack.

Supported systems

The Intertrust system supports packagers and clients for some known file-types. New file-types can be supported through packager and client SDKs and toolkits supplied by Intertrust.

For non-DRM supporting file types the content itself is wrapped in a secure container file format, file types that have support for DRM extensions are not wrapped in this containers. Examples of DRM-enables file formats that are supported by the Rights|System are MPEG-4 and PDF.

Rights|System supports the Rights Expression Language XMCL.

Viewers mentioned to work with the Intertrust DRM system are *Adobe Acrobat Reader* for *PDF-files*, *Musicmatch* for *MP3* and *"MPEG-4 players"* for video. There are clients for Desktop computers, mobile phones, PDAs and digital television decoders.

Availability

The different parts of the Intertrust DRM solution are supported for different platforms. The packager is supported for Windows NT 4.0 and Windows 2000, the Rights-server software is supported for Solaris.

The desktop client supports the Microsoft Windows family of operating systems, the PDA, mobile and digtial television clients support a variety of processors.

3.2.5 Summary

The systems are all well-designed and most of them are built to provide extensions and support for new file types. Most of the systems have clients for the Microsoft Windows operating systems only, and expect the consumer to use this operating system. We have not seen any comparisons in usage of the DRM systems, but it seems that there is no system used very widely. Perhaps the Microsoft Media DRM system will become dominant since it can be shipped with the Microsoft Media player, and the widely used Microsoft operating systems.

Chapter 4

MPEG-21

MPEG-21 is a multimedia framework providing the "big picture" in describing multimedia content on the Internet. It is currently being developed by the MPEG committee, and is still in the process of development. MPEG-21 is based on XML and has the ability to describe complex "Digital Items", stand-alone, compound or grouped together with other items. Examples of items are pictures, movies, sound or just text. MPEG-21 consists of seven parts, some nearly standardised and some unfinished [27]:

- *Digital Item Declaration* (a uniform and flexible abstraction and interoperable schema for declaring Digital Items);
- *Digital Item Identification and Description* (a framework for identification and description of any entity regardless of its nature, type or granularity);
- *Content Handling and Usage* (provide interfaces and protocols that enable creation, manipulation, search, access, storage, delivery, and (re)use of content across the content distribution and consumption value chain);
- *Intellectual Property Management and Protection* (the means to enable content to be persistently and reliably managed and protected across a wide range of networks and devices);
- *Terminals and Networks* (the ability to provide interoperable and transparent access to content across networks and terminals);
- *Content Representation* (how the media resources are represented); and
- *Event Reporting* (the metrics and interfaces that enable users to understand precisely the performance of all reportable events within the framework).

4.1 Digital Item Declaration

The Digital Item Declaration is the foundation that the digital item is based on and provides the syntax that defines the digital item. A digital item is declared using XML, the definition of the declarations can be found in [8].

There are numerous concepts described in [8]. Here we only describe the basics. All these parts are *xml-tags*. The *container* is declared as standard XML tags. For example:

```
<container> ... </container>
```

4.1.1 Container

A container is a grouping of *items* or *containers*. The container may have *descriptors*, giving information about the container. This information may be meta-data, usage rights, etc.

4.1.2 Item

An *item* may contain sub-items and/or *components*, there may be *descriptors* associated with the item.

An item without sub-items is considered to be the lowest level of granularity, allowing a user to access the item, but not *components*. If an item contains sub-items, these may be accessed.

4.1.3 Descriptor

The *descriptor* associates descriptive information with an *item*, a *component* or a *container*. This mechanism allows for adding meta-data such as MPEG-7 descriptors describing the author, year, genre, etc. and it also allows for other descriptive information such as comments, rights expressions, etc.

4.1.4 Component

The *component* is the basic building block of a digital item. It contains a *resource* along with a set of *descriptors*. It may also contain *anchors* describing points of interest in the resource, for example specific time intervals in a movie.

4.1.5 Resource

A resource is a locator pointing to where the actual data is located, or how it can be identified using Uniform Resource Identifier (URI), which may be a traditional Uniform Resource Locator (URL), or it may be a Uniform Resource Name (URN). The URN is a location-independent way of identifying a resource. For example:

```
<Resource ref="http://www.pics.com/pic1.jpg" type="image/jpg" />
```

4.2 Intellectual Property Management and Protection

MPEG-21 Intellectual Property Management and Protection (IPMP) is the framework for DRM in MPEG-21. IPMP is a framework to allow for MPEG-21-content to be protected by IPMP-tools that may be supplied by different vendors. The content contains references to which *tools* are required to view the content.

The design goals are to create platform independence and vendor interoperability among IPMP-tools. Any content protected by a vendors IPMP tool should be viewable on any MPEG-21 terminal. An MPEG-21 terminal may be a stationary computer, a hand-held device, or a mobile phone.

4.2.1 Terminal

The terminal is used by the user to consume digital content according to the usage rules. Since the content may be encrypted, the terminal communicates with IPMP Tools and requests decryption of the content. The MPEG-21 infrastructure allows for many different terminals and tools, but the goal is interoperability, so one given terminal should be able to use the same IPMP tool as another.

4.2.2 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." [10]

The information on what tools to be used to decode or process some content is supplied with the content. The terminal receives the Tools ID and determines whether or not it needs to download the Tool. If so, it downloads the tool and installs it before the content is consumed.

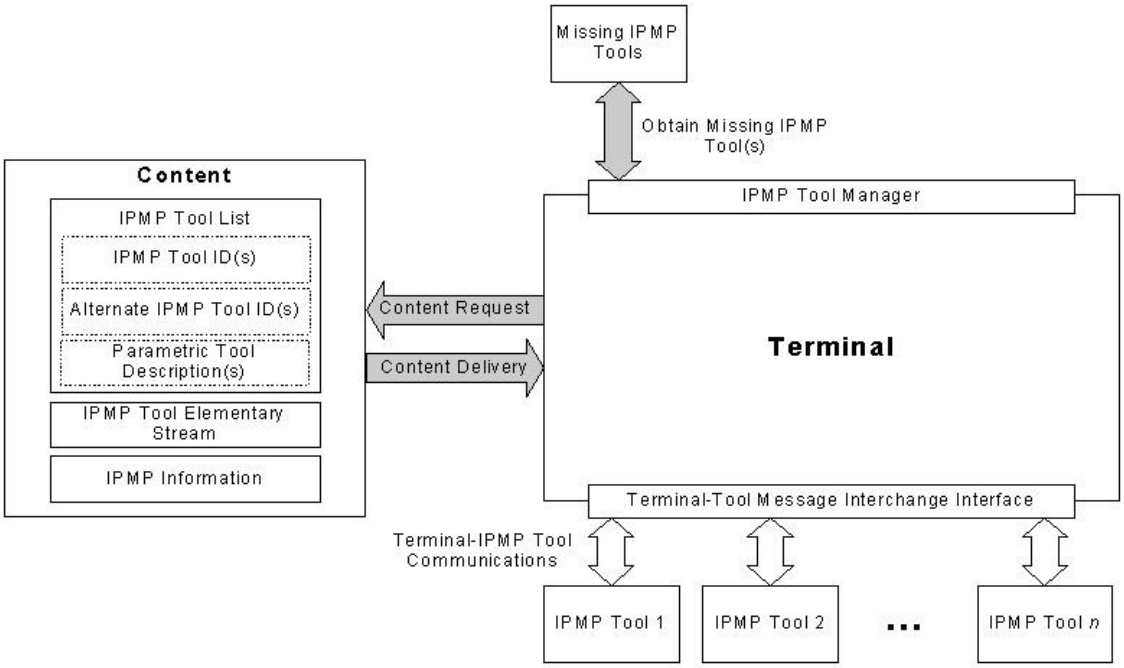

Figure 4.1: MPEG-21 IPMP Architecture [10]

4.2.3 Tool manager

The tool manager is the part of the IPMP system that keeps track of the IPMP tools. It downloads and installs them as necessary. An IPMP tool may be delivered along with the content but the general case should be that the tool manager retrieves the tool as an Internet resource.

4.2.4 Messaging Interface

The terminal communicates with the IPMP-tools through a messaging system (partially) defined by MPEG. The terminal sends messages to the tools to instantiate them and to ask for permissions to view content or use the tools to decode content.

The messaging system is partly defined by MPEG. Some messages are defined down to bit-level, whereas some just have high-level descriptions.

4.3 Digital Item Identification and Description

Any digital item must have a mechanism for identifying the item, independent of its location. The Digital Item Identification and Description scheme (DIID) [9] working document defines a system for accomplishing this.

4.3.1 Goals

The working document from MPEG [9] aims to define the following;

- How to identify uniquely and describe Digital Items (and parts thereof) ; other Entities.
- The relationship between Digital Items (and parts thereof) and existing identification systems; and
- The relationship between Digital Items (and parts thereof) and relevant description schemes.

4.3.2 Syntax

Similar to the rest of MPEG-21, The DIID is based on XML. It utilises *Uniform Resource Names (URN)* [18], which is a way to identify digital object independently of its location. The URN mechanism also allows for different identification schemes, such as ISBN or other existing or future schemes. The system for identification is determined by the *namespace* identifier following the `urn:` expression. The namespace definition and the URN keyword are case-insensitive, but the specific identifier string may be case-sensitive depending on the identifying system. An example of a URN:

```
URN:foo:a123,456
```

In this example, the URN states that this is a URN, *foo* is the namespace of the system for identifying this resource, and *a123,456* is the identification for this specific resource.

As defined in [9], the only tag defined for the DIID scheme is the *diid:identifier* tag. An example from [9]:

```
<diid:identification>
        urn:mpeg:mpeg21:diid:cid:1702.F109/0000011
</diid:identification>
```

By definition of URN, the identification subsystem has to define whether or not a URN identifier is equivalent to another identifier.

4.4 Rights Data Dictionary and Rights Expression Language

These documents describe the language to use when describing users rights to items, and how to add new terms to this existing language. At least XrML and ODRL have been submitted for this purpose, and XrML has been chosen as the basis for defining the MPEG-21 Rights Expression Language.

4.5 Digital Item Adaption

Digital Item Adaption focuses on the possibility of adapting the digital items for different usage scenarios. An environment may be described by the users terminal, and the delivered digital items may be adapted for this users terminal. This allows for many kinds of adaptions such as content filtering (violence, adult content, coarse language, etc.), language choices, content quality, content encoding (jpeg, gif, mp3, ogg, etc.), and many other adaptions.

4.6 MPEG-21 Example

An example of an MPEG-21 Digital Item can be a digital photo-album with matching music that may be played for free a fixed number of times, and for usage above the fixed number a specified fee must be paid. Another example can be a video clip sent to a hand-held device, where a number of options for example for media quality may be chosen before the media is delivered.

A small example of MPEG-21 can be seen in figure 4.6 on the following page, this type of digital item is the reference for the work with the implementation.

The example shows a `Container`, with one item consisting of two sub-items. The `<resource>` is a URI link to the actual content. The `<component>` tag links a number of `<descriptor>` tags to a `<resource>`. A number of `<descriptor>`'s can be added to hold information concerning for example rights and origin descriptions for the resource.

In this example, the DIID [9] scheme is added to identify the first picture. The following descriptor is a reference to the license for the use of the picture called *myFirstPicture.jpg*.

To identify the IPMP tool needed to consume the content, an IPMP tool-list is referenced in the next descriptor. This tool-list contains information on what IPMP tools that are needed.

```
<didl xmlns="urn:mpeg:mpeg21:2002/01-DIDL-NS">
  <Container>
    <Item>
      <Item>
        <Component>
          <!-- Identification of the picture -->
          <Descriptor id="Identification">
            <Statement type="text/xml">
              <diid:Identification>
                urn:mpeg:mpeg21:diid:cid:1702.F109/0000011
              </diid:Identification>
            </Statement>
          </Descriptor>
          <!-- License for this resource -->
          <Descriptor id="License">
            <component>
              <resource ref="license.xml" type="text/xml"/>
            </component>
          </Descriptor>
          <!-- IPMP tools used for viewing this picture -->
          <Descriptor id="License">
            <component>
              <resource ref="myToolList.ipmp" type="application/ipmp-toollist"/>
            </component>
           </Descriptor>
          <!-- The resource itself -->
          <Resource ref="myFirstPicture.jpg" type="image/jpg" />
        </Component>
      </Item>
      <Item>
        <!-- Another resource -->
        <Component>
          <Resource ref="mySecondPic.bmp" type="image/bmp" />
        </Component>
      </Item>
    </Item>
  </Container>
</didl>
```

Figure 4.2: MPEG-21 example, including license entry

The second resource in the example is not affected by the constraints and the tools protecting the first resource.

Chapter 5

Rights Expression Languages

A rights expression language is a language used to describe different users' rights to some content. It may be used to describe simple rules such as that user Alice is allowed to see picture1 but not picture2 while Bob can view both pictures for a specified time period. It may also be used to describe more complex scenarios such as that Alice can view picture1 if she pays a specified amount of money to Bob and half of that amount to Charles.

Examples of rights expression languages are ODRL [25], XrML [23], and XMCL [24]. All these languages are XML-based, this is probably because that XML is very extensible and it has been developed and supported in more and more areas recently. MPEG made a call for proposals on Rights expression languages for MPEG-21. At least XrML and ODRL were submitted, the chosen language was XrML.

5.1 XrML

The *eXtensible rights Markup Language*(XrML) is developed and maintained by Contentguard [21]. The language is defined by XML-schema and contains a core schema and standard extensions schemas. The basic structure of XrML is *the license*. The license contains grants for principals to exercise rights on certain resources. *A right* may be to view, store, modify, transfer or perform an action on the actual resource. A resource is some form of digital entity which the right is being exercised on, it may be identified by a URL reference or in any other way. *A principal* is the person or entity that exercises the right on the resource, a principal can be a user, an organisation, a trusted device, etc.

The keywords *right*, *resource* and *principal* are the basic entities of XrML, they may be used as defined in the *XrML standard* and *standard extensions* or they may be substituted with user-specified expressions. For example, the resource `<cx:digitalwork>` expression in XrML may be substituted with `<mpeg21rel:digitalItem>` if the language is to be be used with MPEG-21.

```
<license>
    <grant>
        <!-- PRINCIPAL -->
        <keyHolder>
            <info>
                <dsig:KeyValue>
                    <dsig:RSAKeyValue>
                        <dsig:Modulus> ... </dsig:Modulus>
                        <dsig:Exponent> ... </dsig:Exponent>
                    </dsig:RSAKeyValue>
                </dsig:KeyValue>
            </info>
        </keyHolder>

        <!-- RIGHT -->
        <cx:play/>

        <!-- RESOURCE -->
        <cx:digitalWork>
            <cx:locator>
                <nonSecureIndirect
                URI="http://www.books.com/thriller.pdf" />
            </cx:locator>
        </cx:digitalWork>
    </grant>
</license>
```

Figure 5.1: Example XrML license

The structure of an XrML license can be seen in figure 5.1. In the example, the principal is identified through the public key (dsig:RSAKeyValue). Note that the modulus and exponent need to contain actual values. The `<cx:play />` expression gives the principal the right to view (play) the resource. The resource itself is a URL contained in the digitalWork expression. The *keyHolder* expression is derived from the abstract *principal* concept, *cx:play* from the *right* concept and *cx:digitalWork* is a *resource*.

Contentguard supplies a *Software Development Kit* (SDK) for XrML, it is based on the *Microsoft XML parser* and the *Microsoft Component Object Model* (COM). The only supported platforms are the Microsoft Windows family of operating systems. The XrML SDK will be described in further

detail in chapter 7 on page 40.

5.2 XMCL

The *eXtensible Media Commerce Language* (XMCL) is a language developed by Realnetworks but was not submitted for standardisation to the MPEG for MPEG-21. Similar to XrML, the language is based on a "license" model. A license is given to the user. This license contains rights, description of the resource and conditions of use. Instead of having a number of grants in a license, each XMCL document may contain a number of licenses to different resources.

The main difference with XrML is that XMCL is smaller and more specified for systems such as IBM *EMMS* and RealNetworks *Media Commerce Suite*. According to the XMCL initiative [24], the language will provide better interoperability because it is smaller and narrower than XrML. This ensures that all systems support the functionalities of XMCL

5.3 ODRL

IPR systems has developed the Open Digital Rights Language (ODRL), based on Nokias MRV and merged with XMCL for the MPEG-21 submission. It differs from XrML in the sense that the model is "offer-agreement" instead of just "license". The content provider sends *an offer* to the consumer, when the consumer agrees it becomes an *agreement*. ODRL is released in the Open Source spirit, and IPR systems do not have any license requirements. Similar to XrML, ODRL is based on XML schemas and has core expressions that can be extended.

An example of the structure of ODRL can be seen in figure 5.3 on the next page. The *context* keyword identifies the object that the license refers to, and it may also include details on the date of the license and other information describing the transaction. The *permission* tag defines what actions are allowed on the content, this can be subject to *constraints*, such as that the content may only be viewed by some user, some group, or some device, etc. *Requirements* may be set in such way that the user needs to pay a certain amount, or that some usage tracking system must be used. The *party* tag identifies the parties involved in the agreement. The offer may have a unique identifier, so that an agreement may be made with a reference to the corresponding offer.

```
<rights>
    <context>.
        <uid> ... </uid>
    </context>
    <offer>
        <asset> ... </asset>
        <permission>
            <permission-type>
                <requirement> ... </requirement>
                <constraint> ... </constraint>
            </permission-type>
            <condition> ... </condition>
        </permission>
        <party>
            <context> ... </context>
            <rightsholder> ... </rightsholder>
        </party>
    </offer>
    <agreement>
        <context> ... </context>
        <party> ... </party>
        <permission> ... </permission>
        <asset> ... </asset>
    </agreement>
</rights>
```

Figure 5.2: ODRL simplified example

Chapter 6

SMICL MPEG-21 Viewer

6.1 Introduction

As part of the research of the *Secure Multimedia Information Communication Laboratories* (SMICL) [14] at the University of Wollongong, Nicholas Sheppard has developed an MPEG-21 viewer.

The viewer supports a subset of the MPEG-21 digital item declaration [8]. It is written in C++ using the wxWindows toolkit [31] for portability. It uses the C XML library for Gnome [32] to parse the MPEG-21 DID. The graphical user interface lets the user browse the digital item as a tree structure.

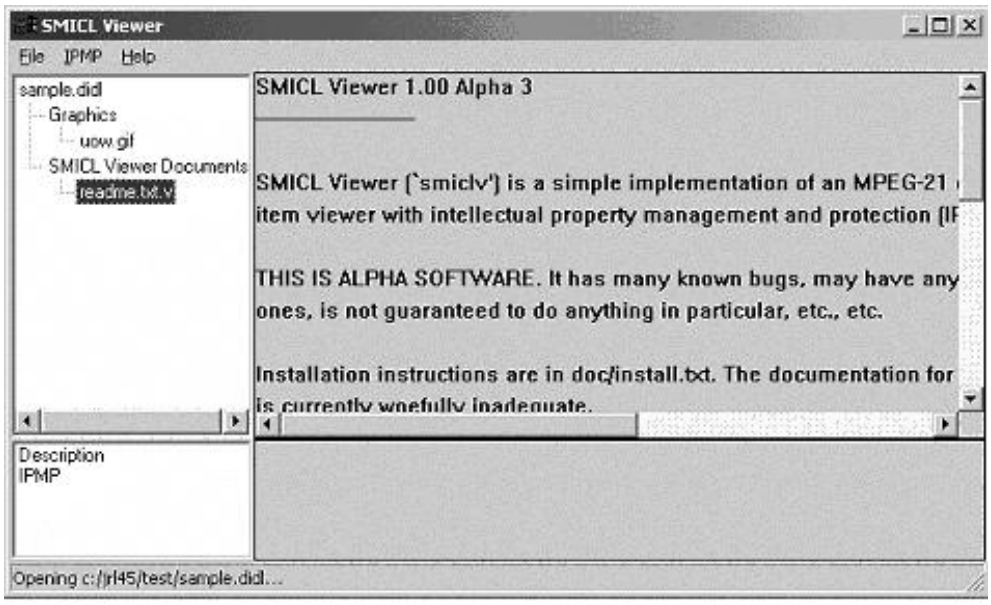

Figure 6.1: SMICLV User interface

The SMICL Viewer is a research platform for MPEG-21 related technologies. It has been compiled and runs on Solaris, Linux and MS Windows, but

can easily be ported to other platforms.

IPMP is partly implemented, the viewer supports some IPMP messages (mentioned later). The IPMP tools are not acquired on demand but are supposed to reside in dynamic link libraries in a specific directory. Most of the IPMP-messages are mentioned in [10], but most of them are not fully specified, therefore the current implementation may have to be changed if the document is updated.

6.2 Functionality

The viewer is able to read a Digital Item Declaration (DIDL), and display it as a tree structure. As the user clicks on the components in the tree-structure, the viewer displays these resources.

The viewer can render images and text, and support for video will probably be incorporated in the future.

DIDL files are loaded from URL:s, which can be transported from web-sites, ftp-sites or local files. The viewer supports caching.

The GUI provides a few menu options, the user can *open* a Digital Item Declaration (DIDL) file, change *configuration*, *list* IPMP tools, *clear cache* and view information *about* the viewer.

6.3 Architecture

The diagram in figure 6.2 shows the architecture of the SMICL viewer, where *DIDL* is a Digital Item Declaration in XML format and *wxWindows, libXML* are toolkits.

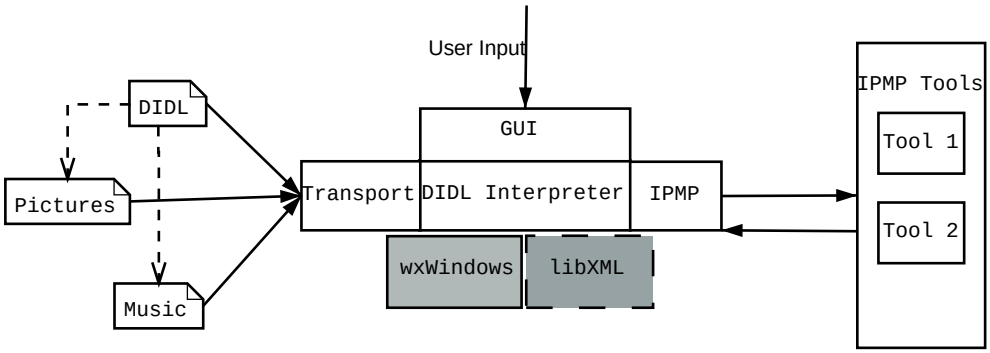

Figure 6.2: SMICL Viewer overview

The viewer has been compiled using Microsoft Visual C++ in Microsoft Windows and with g++ on Linux/gtk and Solaris/motif.

6.3.1 wxWindows

The wxWindows toolkit [31] is used for portability, this toolkit is an open-source, free, C++-based collection of classes for a number of applications. The toolkit provides classes for graphical user interface, files, streams, multi-threading, storage data-types such as hash tables and many other features. The wxWindows toolkit is available for a range of systems, including Windows, Solaris, Linux, OpenVMS, and Macintosh.

6.3.2 libXML

Libxml is the XML C library developed for the Gnome project. This library is released as open-source, and it is free. The library provides a Simple API for XML (SAX)-like interface [32] for access to the parsed XML document.

The XML C library for Gnome is ANSI-C/POSIX compliant and available for Linux, Windows and Solaris, and it is ported to other systems.

6.3.3 DIDL Interpreter

The DIDL interpreter interprets the parsed Digital Item Declaration Language (DIDL)-XML document and creates objects for the expressions in the DIDL. For example, the <item> tag will be parsed, and a *diItem* object (see 6.4.9 on page 39) will be created, containing references to the child-tags, associated descriptors, etc.

6.3.4 IPMP

The Intellectual Property Management and Protection (IPMP) parts of the viewer handle the instantiation and communication with the IPMP tools used to consume some digital content. This part consists of a message router and an IPMP terminal containing references to available tools.

6.3.5 IPMP Tools

A few IPMP tools are provided for the SMICL viewer, one is a simple encryption and decryption tool, one is a "confirmation" tool, which displays a confirmation dialog, and one is a dummy tool which does nothing.

6.3.6 GUI

The graphical user interface, as seen in figure 6.1 on page 34, is created using wxWindows [31] classes, and is adapted to the environment on which the SMICL viewer is compiled. The example shown in 6.1 is a snapshot from the Windows version.

6.4 Classes

The SMICL Viewer consists of more than 60 classes, a simplified overview of these can be seen in figure 6.3.

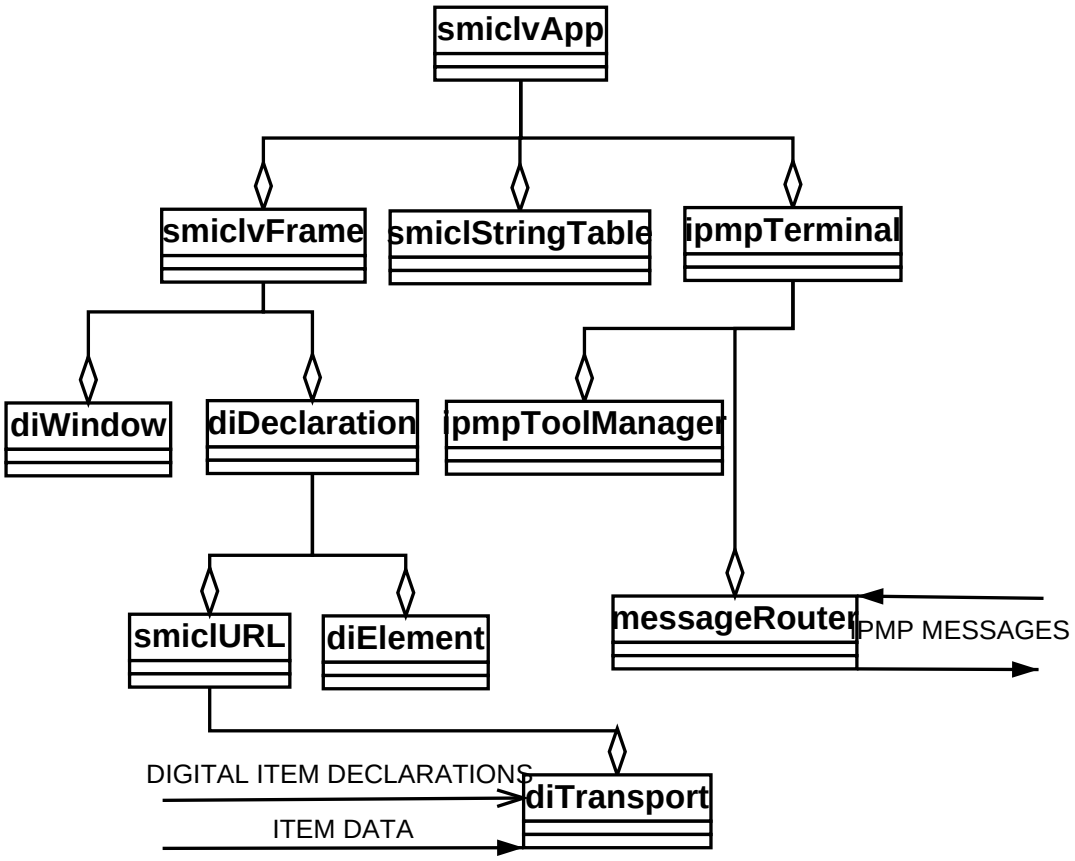

Figure 6.3: Simplified class overview of the SMICL viewer

6.4.1 smiclvApp

This class is the main application, it contains references to the parts of the viewer.

6.4.2 smiclvStringTable

The viewer requires a language resource file containing string resources to be used for the GUI. The string table class contains these strings that are loaded from the language resource file.

6.4.3 smiclvFrame

The frame class contains the parts of the GUI, this class contains menu declarations and initialisation and a reference to the render window.

6.4.4 ipmpTerminal

The IPMP terminal is the messaging centre for the IPMP tools. The terminal is created on startup of the viewer, the viewer only contains one instance of this class. The IPMP terminal contains methods for carrying out the various requests and handling responses from the IPMP tools.

6.4.5 ipmpToolManager

The tool manager handles the loading of the IPMP tools. This class loads all the dynamic link files contained in the plugins path, in the viewers resource path.

6.4.6 messageRouter

The messages sent from and to the IPMP terminal are routed in the message router. IPMP Messages may be synchronous or asynchronous.

6.4.7 diWindow

The diWindow class is the main GUI component. This class creates and contains references to the navigator window, the main display window and the descriptor window. The navigator window is the leftmost window used to navigate the digital item declaration as a tree structure. The main display window displays the resource, and the descriptor window at the bottom is used to display descriptors.

6.4.8 diDeclaration

The diDeclaration is the document containing the digital items, this class holds references to the digital item declaration, as parsed by the XML parser and interpreted as diElements.

6.4.9 diElement

All digital items, descriptors, statements, etc. are interpreted as derived classes of the diElement class. Examples of derived classes are *diItem*, *diComponent* and *diDescriptor*, these classes correspond to the tags <item>, <component> and <descriptor>.

6.4.10 smiclURL

This is the class handling parsing of a URL, and determining how to transport the requested data.

6.4.11 diTransport

The diTransport is the base class for classes handling transport of data, the derived classes being the *smiclHTTP*, *smiclFile* and *smiclFTP*.

Chapter 7

Extensions to the SMICL Viewer

7.1 Introduction

The implementation part of this thesis is the creation of a Rights Expression Language (REL) handling IPMP-tool for the SMICL Viewer. The tool is implemented as a plugin for the viewer. The REL plugin has basic functionality for inclusion of the eXtensible rights Markup Language (XrML) in MPEG-21. The implementation is based on references to licenses *inside* the Digital Item Descriptions. We will discuss refinements of the IPMP messages, and the implementation in the SMICL Viewer. We will give remarks on the effects of including licenses in MPEG-21 DIDL, and propose behaviour for a rights enforcing tool.

7.2 XrML Software Development Kit

The *XrML Software Development Kit* (SDK), from Contentguard [21], is a software for integrating XrML functionality into applications. It is created as a modular framework with good options for extensions. It is based on the *Component Object Model* (COM). It is only available for the Microsoft Windows platform.

The XrML SDK is built as seen in figure 7.1 on the facing page. License handling is separated into *license interpretation* and *condition validation*. License interpretation is the process of finding *grants*, for a *principal* to exercise a *right* to a *resource*, subjected to a number of *conditions*. Condition validation refers to the monitoring of conditions related to a grant. The normal sequence of events for an application using the XrML SDK is license interpretation followed by condition validation.

The XrML SDK allows for the creation of a number of *license interpreter plugins* and *condition validator plugins*. This is to support XrML extensibility

through inheritance of the basic XrML expressions.

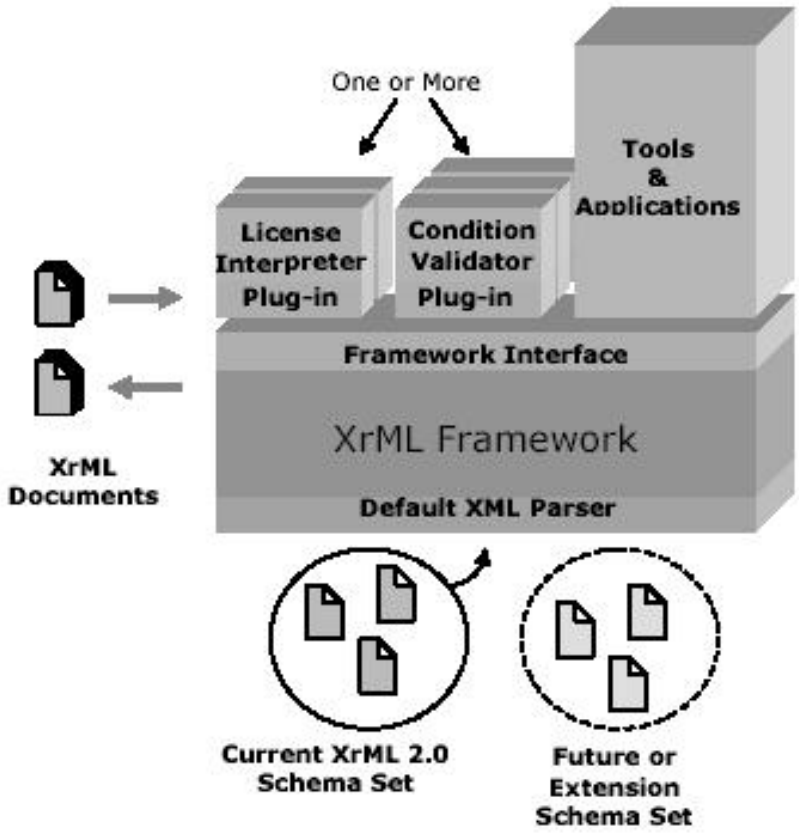

Figure 7.1: The XrML SDK architecture

The XrML standard concepts right, resource and principal are all abstract concepts, and can be inherited. The right may be `<cx:play>`, `<cx:print>` (as already defined in XrML) or it may be an application-specific such as `<myNamespace:myRight>`.

An example of this is seen in figure 7.2 on the next page, where the XrML resource is replaced by the `<mns:movie>` construct. This new type of resource must be validated by a license interpreter plugin that is able to handle the mns-namespace.

Whenever the main license interpreter finds an expression that it doesn't know how to interpret, it requests the license interpreter plugins to validate the expression.

The XrML SDK is free for use in personal software, but a license is required for use in commercial software.

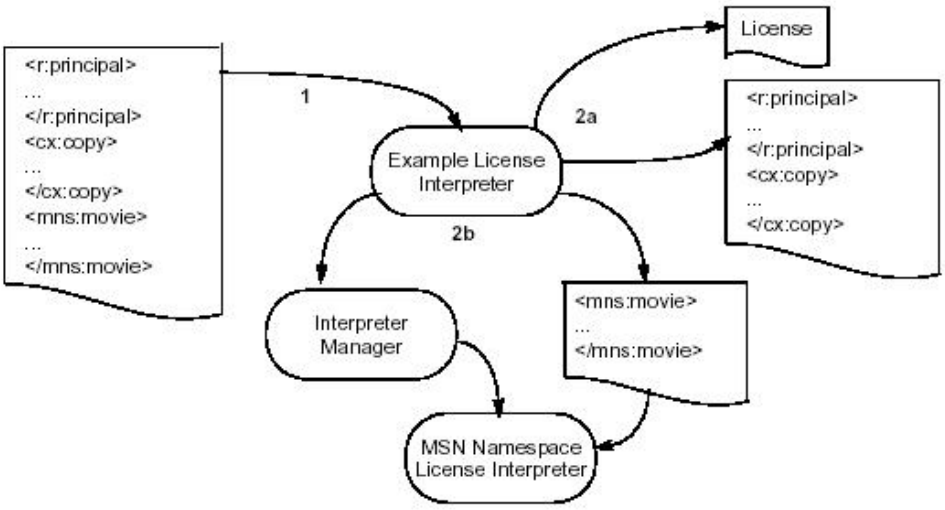

Figure 7.2: XrML Namespaces

7.3 System Architecture

The REL plugin is compiled as a Windows *dynamic link library* (dll) file. On startup, the SMICL Viewer loads the REL plugin dll, and the plugin registers itself with the SMICL Viewer through a function called smiclv_ipmp. This function registers the plugins tool-ID and a small API with the SMICL viewer.

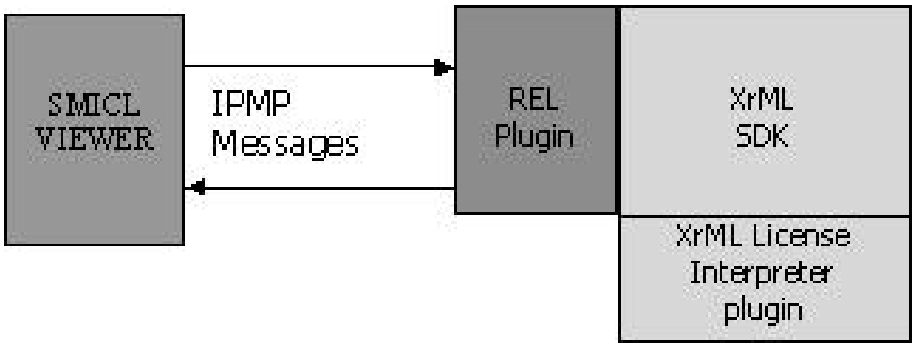

Figure 7.3: REL plugin overview

7.4 License Interpreter plugin

An example of a license interpreter plugin is available with the XrML SDK distribution. From this we have implemented a plugin that handles the

"MPEG Paramas"-namespace, an example where the resource is inherited to enable referencing to MPEG-21 Digital Item Identification, that is being developed by MPEG. The XML statements <digitalItem>, <digitalContainer>, etc. are introduced.

This namespace is temporary and for experimental purposes, it is not enclosed with this thesis since it is part of MPEG core experiments and not released by the MPEG. We believe the MPEG will define and release another namespace for these definitions at a later stage.

7.5 Condition Validation

At the moment condition validation supports *validityInterval*, as implemented in the default condition validator plugin. Whenever the validityInterval becomes invalid, the condition validator plugin send an event to the REL plugin.

The plugin needs to send an IPMP-revoke message and the SMICL Viewer needs to be updated to support that message.

7.6 Class Overview

The REL plugin consists of a few classes and some C-functions for IPMP messaging, these classes are described below.

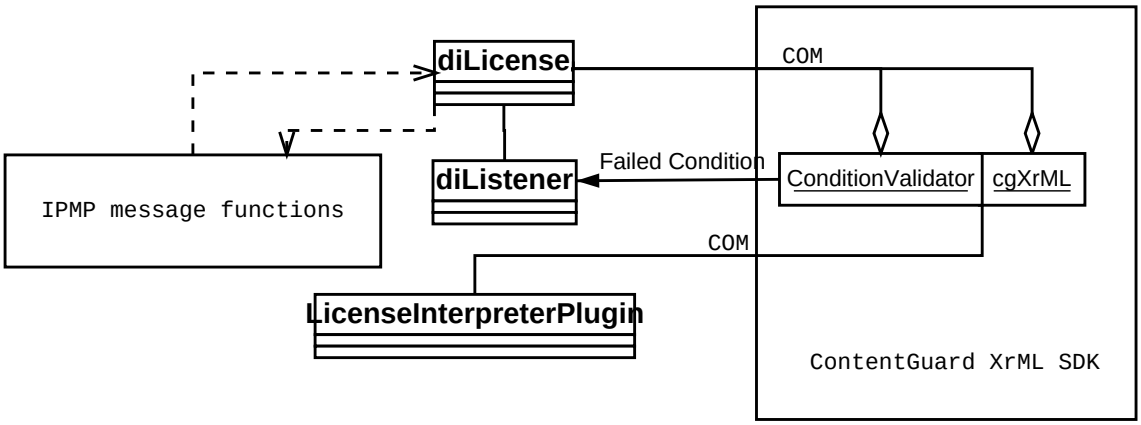

Figure 7.4: REL plugin classes

7.6.1 License handler class

The license handler class called *diLicense* only has a few public functions, these are described here.

Constructor

The constructor creates the XrML framework object.

```
diLicense(
  const MPEG21_Octet *license, // Null terminated string
                               // containing license
  MPEG21_Short Sender,         // Sender ID of the intent-request
  MPEG21_Short Recipient       // This Tools own ID
)
```

validate

Validates a license and gets the conditions for use of a the resource identified by the supplied MPEG-21 Identification string.

```
int validate(
  const MPEG21_Octet *id
  // null terminated string containing identification of a resource
  // expected to be like
  // <diid:identification>INSERT_URN_HERE</diid:identification>
)
returns : 1 on success, 0 on failure
```

CheckConditions

The CheckConditions method starts the condition checking and creates a listener (see 7.6.3 on the facing page) for the condition validator.

```
int CheckConditions()
returns         : 1 on success, 0 on failure
```

revoke

This method stops the condition validation and sends a revoke message. This method is called by the condition listener.

```
void revoke(
  DWORD dwCookie,   // cookie of the condition manager
                    // that fired a condition
  BSTR bstrURI,     // Namespace of the condition that failed
  BSTR bstrBaseName // basename of the failing condition
)
```

7.6.2 License interpreter plugin

We have created a license interpreter plugin to handle the "mpegParamas" example namespace.

The license interpreter plugin is based on the example license interpreter plugin supplied with the XrML SDK. The only functionality of the license interpreter plugin is to validate that the data of the <diid:identification> expression matches the data of the <diid:identification> of an intent-request.

The main function of the license interpreter plugin is the *ValidateExtendedGoal* function, it performs a string comparison for the request identity-string with the identity-string in a grant.

An example is the string

```
<diid:identification>urn:a:b</diid:identification>
```

, where *urn:a:b* is the string being compared.

7.6.3 Condition Listener

From the examples of the XrML, we have implemented a listener for the condition validator. This listener listens for events fired by the condition validator.

When an event is fired by the condition validator, the condition listener calls the *revoke* method in the diLicense class, and passes the appropriate information.

The condition listener is implemented as a COM object. There are two public methods.

setLicense

Sets the parent license to this listener. Must be called before condition validation starts.

```
void setLicense(
  diLicense *lic   // pointer to the parent diLicense
)
```

Event handler

Handles the events fired by the condition validator. Calls the revoke method in the parent diLicense.

```
STDMETHOD(NotifyEvent)(
  DWORD dwCookie,     // Cookie of the condition validator
  BSTR bstrURI,       // Namespace of the failing condition
  BSTR bstrBaseName,  // Basename of the failing condition
  VARIANT_BOOL bValue // VARIANT_FALSE on a failed condition, otherwise true
)
```

7.7 Extending The Viewer

In extending the viewer a number of design decisions need to be made. In the following sections we describe these issues and the proposed solutions.

7.7.1 XrML license transfer

The license must be transferred with the digital item that is delivered. There have not been any clear directions from the MPEG on how this should be done. We have identified two main ways to deliver the license. One is within the scope of the digital item and the other is outside the scope of the digital item.

7.7.2 Effects of including license in DID

From the description of the `<descriptor>` element in MPEG-21, the effective scope of the `<descriptor>` is its parent element. Therefore, the scope of the XrML license must be the corresponding `<descriptor>`'s parent element.

The parent element may be a compound item, in that case the XrML license scope must apply to the descendant elements.

An XrML license is a file describing the restrictions of usage for a certain resource. It includes "grants" allowing the use for a "principal" to a specified "resource". If user Alice is allowed to view the resource picture1.jpg, there will be a grant that allows her to do that. If there is no grant allowing user Bob to view picture1.jpg, he should not be granted access to that resource. This has one important implication, described below.

> The use of a resource or an item within a license's scope not mentioned in a license shall be disallowed, except for resources that are sub-items of an allowed item.

7.7.3 IPMP communication

The communication between IPMP tool and terminal must be designed to allow interoperability. Still it has to be intelligent so that no unnecessary work is being done by terminal or tool. The REL tool will have to acquire the license and the identification of the item being processed to be able to carry out the validation process. This can be accomplished in a number of ways. Three ways are presented as follows.

Requesting mime-type

On instantiation step, the tool registers itself to the terminal. It sends a message containing a flag telling that it needs descriptors from the DID. In that request is also a string containing the mime-type of the requested descriptors. For the REL tool, this would be "text/xml". When the terminal encounters protected content, it sends an intent-request, which states that it intends to view a specified content. This intent-request would hold the descriptors requested, or pointers to the data of those descriptors. The positive effects of this solution would be that it would be quite generic, a tool could request all descriptors of specified mime-type, and the terminal would not need to know much about the behaviour of the tool. The negative is that the tool will have to parse the descriptors received for identifying what they are before handling of the descriptors. This method would be more effective if all data contained in descriptors would have a mime-type identifying more of what it contains, for example "application/xrml" and "application/diid" or similar instead of just "text/xml" for both options.

Requesting descriptor ID

Instead of requesting the mime-type of the descriptors, the tool could request the ID of the descriptor. In this case the descriptors could be requested in

a particular order, and sent back to the tool in the same order. With this solution the tool can assume that the descriptors are in the same order as requested, and it would not have to have a specific identifying process before processing the data.

The drawback of this solution would be that the descriptors would have to have set IDs. Eg. the descriptor containing the XrML license could have the ID "XrML-License" or "REL", and the descriptor containing IPMP-toollist would have "IPMP". An MPEG-21 file not following these guidelines would be incorrectly handled by the terminal.

Requesting DOM node

The Document Object Model (DOM) supported by most XML parsers could be used for eliminating double-parsing of the XML. If the viewer and the REL plugin use the same XML parser, a reference (pointer) to the DOM-node could be sent. This could be an efficient option, but this requires that the viewer and the REL plugin share parser or for the MPEG-21 to have a DOM interface in the IPMP messaging.

Choice in the SMICL Viewer

At the moment the IPMP tool-lists are being identified by the mime-type of a descriptor set to "application/ipmp-toollist". The continuing work will be based on this approach, since identifying descriptors by the ID would have to be defined by the MPEG, and it is not at the moment. No indications have been seen that it will either. The REL plugin uses the Contentguard XrML SDK, which is based on the Microsoft XML parser. Using this parser in the SMICL viewer would render the viewer platform-specific, so sending a DOM-node reference is not a good alternative.

7.8 IPMP Messages

The IPMP messages are described in [10], but not many of them are fully specified. Most of the descriptions are very brief. In this section we describe the implementation of the IPMP messages in SMICLV. The details of these messages can be found in the header file *MPEG21-IPMP.h*, in the appendix section.

7.8.1 Messages of the SMICLV IPMP Terminal

IPMP_CreateNewToolInstance

Create a new tool instance. This message is sent to a tool when the tool is needed. The terminal attaches any needed parametric descriptions supplied in the IPMP tool-list.

IPMP_RequestInstTools

Request the terminal to send a *ToolInstNotification* message containing all the instantiated tools on the terminal.

IPMP_AddToolInstNotificationListener

This message allows for a tool to receive notifications on all new instantiated tools on the current terminal.

IPMP_RemoveToolInstNotificationListener

This message is sent when a tool desires to end receiving notification messages for new instantiations.

IPMP_ToolInstNotification

A tool instantiation notification message. This is sent to the terminal when an IPMP tool has be instantiated. This message contains the tools ID and information on whether or not the tool needs descriptors, and if the tool needs the data of the resource being viewed.

IPMP_ProcessData

This message is sent to a tool when the terminal wants the tool to process some data. The data can be processed "in place", in this case a pointer is passed with the message, otherwise the actual data in sent with the message.

The terminal expects an *IPMP_ProcessDataReturn* message in response to this request.

IPMP_ProcessDataReturn

When a tool has processed some data, for example, decrypted encrypted data, it sends a *IPMP_ProcessDataReturn* message with the processed data or a pointer to the processed data.

IPMP_IntentRequest

When the terminal intends to use a protected resource, it must send a message containing a number of intent requests to the tools protecting the resource. This message will contain appropriate information for the tool to verify the intent. In the current SMICLV implementation of IPMP, the intent request message contains information about

- *what* the intent is, copy, view, authenticate or broadcast
- if the resource is required for processing, a pointer to the resource, or the data of the resource sent with the message
- if the tool requires some descriptors associated with the resource, these descriptors are sent with the message

IPMP_IntentResponse

This message is the response to the intent request(s), it contains

- A response to the intent request(s), granted or denied
- The processed data, if it has been processed

IPMP_IntentRevoke

If an intent, previously granted, needs to be revoked, an *IPMP_IntentRevoke* message is sent to the terminal. This message contains information on what intent has been revoked.

In the SMICL viewer, this message is sent by the REL plugin whenever a required usage interval expires.

7.8.2 REL Plugin handled messages

IPMP_CreateNewToolInstance

When the plugin receives this message, it sends an *IPMP_ToolInstNotification* message to the sender to tell that it needs all the "text/xml" descriptors associated with the current object.

IPMP_Intent_Request

The intent request should be sent to the plugin when the user wants to view content protected by the REL plugin. The REL plugin supports the *view* intent. The message should consist of all associated "text/xml" descriptors for that resource. The REL plugin expects one descriptor to contain an XrML license and another to contain an MPEG-21 Digital Item Identification expression.

The intent request handler sends an *IPMP_Intent_Response* to the sender, telling if viewing of the content is permitted.

7.9 Algorithms

The following algorithms describe the sequence of events in the REL plugin, the first is the *Intent-request handler*, which receieves an intent request from the viewing terminal. The second algorithm describes the steps taken as the license is being interpreted. The third is the steps taken for handling *conditions* associated with the *grant* found in the condition validation.

7.9.1 Intent-request handler

The handler for the "Intent-request" message is the central part in the REL plugin. It receives a license, and an identification of the resource that is being viewed. After validation of the license it sends a response, whether or not viewing is allowed. The main steps are

1 Receive message

2 If the message is supported

  2.1 Extract the text data fields from the message

  2.2 For every text data field

    2.2.1 Quick-Parse the text data

    2.2.2 If appears to be a license

      2.2.2.1 Set license to text data

    2.2.3 If it appears to be a <diid:identification>

      2.2.3.1 Set identification to text data

3 Create a license object with the license and identification supplied

7.9.2 License Interpretation

1 Create an XrML object

2 Save the license to a temporary file

3 Let the XrML object load the license from the file

4 Create XrML objects from template files for principal and right

5 Create XrML resource object from a template file and substitute the token in the file with the identification of the resource being viewed

6 Create XrML object lists for resource, principal and right

7 Add the principal to a principal list, add the right to a rights list, add the resource to a resource list.

8 Call the XrML license interpreter with the rights list, the principal list and the resource list

9 If interpretation succeeded

  9.1 Retrieve the list of conditions for the resource

  9.2 If one ore more conditions are present

    9.2.1 Set up condition validation

  9.3 Return a non-negative value

10 Else

  10.1 Return zero, indicating failure

7.9.3 Condition validation

When there are conditions present, condition validation must be made. The current implementation only supports the validityInterval expression. This is used to set a restricted time period for usage. The condition validation works as follows.

Given a list of conditions from the license interpretation

1 Create a condition validator object

2 Add the list of conditions to the validator object

3 Start the condition validator

4 Wait for the condition validator to fire events

  4.1 When an event is fired

    4.1.1 Send a IPMP-revoke message to the viewer, telling that viewing is no longer permitted

Chapter 8

Conclusion

Digital Rights Management is a challenging task. A number of systems have been proposed over the past few years, each with their own gains and drawbacks. MPEG-21 promises to provide a general framework for multimedia content delivery including digital management of rights.

We have reviewed some existing systems for Digital Rights Management and the main Rights Expression Languages. Further we have given a description of MPEG-21, and the MPEG-21 Intellectual Property Managament and Protection (IPMP) system. A description of the SMICL MPEG-21 Viewer, and its architecture has been given. We have investigated how to include rights information into digitial items, and recommended a behaviour for a rights enforing tool for MPEG-21.

The main part of our work has been the implementation of a Rights Expressison Language tool implemented as a plugin for the SMICL MPEG-21 Viewer, and the required extensions to the viewer.

8.1 Functionality

The REL plugin provides a basic functionality for validating licenses. It matches a request to *a grant*, and responds to the requested *intent request*. Furthermore the REL tool provides *condition validation*, it will monitor the *validityInterval* condition and revoke an expired intent.

8.2 Security

There are a number of extensions that are required to enhance security.

1 No encryption, everything is in clear text. This means that a malicious user can inspect the Digital Item Declaration for the resources, download them and use them without restrictions.

2 There is no mutual authentication of either viewer or plugin. This means that the viewer can not be establish the authenticy of the plugin, it may be a fabricated, untrusted plugin. The plugin cannot trust that the terminal can be trusted.

3 There is no license authentication, the license is may be modified by a malicious user to enable unrestricted access to the content.

4 Code needs further testing to ensure correct performance in different environments.

5 An initial condition validation should be done before sending the *Intent-Reponse* to the viewer to prevent "free" viewing.

Mutual authentication is an important issue for the establishment of a trust relationship between terminal and tools, this topic may be a point of future work.

Since all MPEG-21 files are in clear text and no authentication is made, a simple attack to override license rules would be to erase the "ipmp-toollist" entry in the MPEG-21 DID. Another simple attack would be to change the license reference in the DID to point to another license that permits everything.

One way to ensure that a license or a digital item declaration has not been tampered with is to add a digital signature to the DIDL or to the license.

These security issues are proposals for future work.

8.3 Future work

- Port the plugin to other platforms. This requires the XrML SDK to be ported to other platforms, or an implementation without the help of the XrML SDK. MPEG-21 aims to be interoperable, so restricting the implementation to Microsoft Windows does not conform with the MPEG-21 specifications.

- Adding encryption and authentication of DID and license. To increase the level of security, encryption must be added. This will prevent attacks such as modifications of the DID and the licenses, communication interception. If authentication of licenses is added, the REL tool can establish that the license has not been fabricated or modified.

- Define semantics for and implement support for nested licenses. Schemes should be defined to resolve issues such as conflicting rights and different rights for items and sub-items.

- Create a condition validator plugin to handle more conditions. Conditions for a specific system may be implemented, for example communication with an online usage tracking service such as the Universal Description, Discovery, and Integration (UDDI).
- Add decryption keys to the license, so REL-tool or another tool may decrypt encrypted content. If the content should be protected, it needs to be encrypted to prevent non-authorised viewers to view the content.
- Verify the current implementation with the updated MPEG working documents. The MPEG-21 specifications are under development so the current implementation may need to be changed to conform with the updated documents.
- Update the license interpreter plugin when the correct namespace has been defined by MPEG. The "MPEGParamas" example that has been used is experimental, the final version of the XrML extensions will include more expressions.

Chapter 9

Glossary

Cryptolope - Cryptographic Envelope, a DRM system from IBM

DI - Digital Item, the main concept of MPEG-21, a single or compound digital object associated with MPEG-21 descriptors.

DIA - Digital Item Adaption, part of MPEG-21 that enables adaption of Digital Items for different users and viewers.

DIDL - Digital Item Declaration Language, XML based language to describe Digital Items in MPEG-21

DMCA - Digital Millenium Copyright Act, U.S. law concerning copyright for digital information.

DRM - Digital Rights Management, systems and tools to maintain and enforce usage rights

EMMS - IBM Electronic Media Management System, a DRM system from IBM.

IPMP - Intellectual Property Management and Protection, a part of MPEG-21 and MPEG-4 designed to enable DRM systems with MPEG streams.

IPMP-Tool - A tool for decryption, authentication or other purposes in an MPEG IPMP system.

MPEG - Moving Pictures Expert Group, creators and maintainers of the popular MPEG standards for video and multimedia

ODRL - Open Digital Rights Language, an "open-source" XML based language for describing rights in DRM systems.

REL - Rights Expression Language, language to describe user rights to some digital object

SMICL - Secure Multimedia Information Communication Laboratories, research group at the University of Wollongong

SMICLV - SMICL Viewer, an MPEG-21 viewer developed at the University of Wollongong

URN - Uniform Resource Name, Non-location dependent identification system for digital resources

WIPO - World Intellectual Property Organization, an organisation in the UN working with patent and copyright questions.

XrML - The eXtensible Rights Management Language, a Rights Expression Language from Contentguard Inc.

Bibliography

[1] IBM. *Cryptolopes: more information.* `http://www-3.ibm.com/-software/security/cryptolope/about.html`, May 2002

[2] IBM. *Superdistribution of cryptolope.* `http://www.research.ibm.com/people/k/kaplan/-cryptolope-docs/cryfoils-full.ps`, May 2002

[3] Neil McAllister. *Freedom of Expression - Emerging standards in rights management.* `http://www.newarchitectmag.com/documents/-s=2453/new1011651985727/index.html`, May 2002

[4] ISO/IEC MPEG (Moving Picture Experts Group). `http://mpeg.telecomitalialab.com`, March 2002

[5] Maes, M.; Kalker, T.; Linnartz, J.-P.M.G.; Talstra, J.; Depovere, F.G.; Haitsma, J: *Digital watermarking for DVD video copyright protection.* IEEE Signal Processing Magazine, September 2000

[6] Lim, Y., Xu, C., Feng, D.D. *Web Image Authentication Using Invisible Fragile Watermark* `http://www.cs.usyd.edu.au/~vip2001/-proceedings/yslim.pdf`, June 2002

[7] ISO/IEC MPEG (Moving Picture Experts Group). *Study on MPEG-21 (Digital Audiovisual Framework) Part 1 v2.0* `http://mpeg.telecomitalialab.com/working_documents/-mpeg-21/tr/pdtr.zip`, June 2002

[8] ISO/IEC MPEG (Moving Picture Experts Group). *Digital Item Declaration FCD*, `http://mpeg.telecomitalialab.com/public/-mpeg-20_did_fcd.zip`, March 2002

[9] ISO/IEC MPEG (Moving Picture Experts Group). *Digital Item Identification and Description FCD*, `http://mpeg.telecomitalialab.com/-public/mpeg-21_diii_fcd.zip`, March 2002

[10] ISO/IEC MPEG (Moving Picture Experts Group). *MPEG-21 Intellectual property management and protection.*

http://mpeg.telecomitalialab.com/public/mpeg-21_ipmp.zip, March 2002

[11] Stinson, D.R. *Cryptography, Theory and Practice.* CRC Press, 1995

[12] Pfleeger, C.P. *Security in Computing.* Prentice Hall, 1989

[13] W3C - World Wide Web Consortium. http://www.w3.org, June 2002

[14] Secure Multimedia Information Communications Laboratories. http://www.itacs.uow.edu.au/research/smicl/index.html, June 2002

[15] Brisbane, Gareth *Image Watermarking* http://www.itacs.uow.edu.au/research/smicl/projects/IWM/-index.html, May 2002

[16] W3C - World Wide Web Consortium. *XML Schema.* http://www.w3.org/XML/Schema, May 2002

[17] W3C - World Wide Web Consortium. *eXtensible Markup Language* http://www.w3.org/XML/, June 2002

[18] IETF RFC 2141. *Uniform Resource Name (URN) Syntax.* ,1998

[19] CMS Watch. *Topic: Digital Rights Management (DRM).* http://www.cmswatch.com/ContentManagement/Topics/-FeaturedTopic/?topic_id=10, June 2002

[20] The Intertrust website. http://www.intertrust.com, June 2002

[21] Contentguard http://www.contentguard.com, June 2002

[22] XrML 2.0 Specification & Schema http://www.xrml.org/get_XrML.asp, March 2002

[23] ContentGuard *eXtensible rights Markup Language* http://www.xrml.org, June 2002

[24] eXtensible Media Commerce Language http://www.xmcl.org, June 2002

[25] *The Open Digital Rights Languge Initiative* http://www.odrl.net, June 2002

[26] IBM. *Electronic Media Management System (EMMS)* http://www-3.ibm.com/software/data/emms/, June 2002

[27] ISO/IEC MPEG (Moving Picture Experts Group). *Requirements for Digital Item Adaption* http://mpeg.telecomitalialab.com/cfp/-dia_cfp_criteria.zip, June 2002

[28] Costello, Sam. *Microsoft's digital rights management hacked.* http://www.cnn.com/2001/TECH/internet/10/25/ms.hacked.idg/, June 2002

[29] Microsoft *Microsft Media DRM* http://www.microsoft.com/-windows/windowsmedia/drm.asp, June 2002

[30] Star Lab Website http://www.star-lab.com, May 2002

[31] *wxWindows cross-platform GUI library* http://www.wxwindows.org, June 2002

[32] *The XML C library for Gnome* http://xmlsoft.org/, June 2002

[33] *Upphovsrätt - Sajten om upphovsrättsliga spelregler (Swedish Copyright laws)* http://www.upphovsratt.nu, June 2002

[34] World Intellectual Property Organization http://www.wipo.int, June 2002

[35] *THE DIGITAL MILLENNIUM COPYRIGHT ACT OF 1998; U.S. Copyright Office Summary* http://www.loc.gov/copyright/-legislation/dmca.pdf, December 1998

[36] Caslon Analytics Intellectual Property guide *fair use: incentives, innovation, users* http://www.caslon.com.au/ipguide11.htm, June 2002

[37] The Anti-DMCA website http://www.anti-dmca.org, June 2002

[38] The European Parliament *DIRECTIVE 2001/29/EC; The harmonisation of certain aspects of copyright and related rights in the information society.* http://europa.eu.int/comm/internal_market/en/-intprop/news/com29en.pdf, , June 2002

[39] Commomwealth of Australia: Attorney-Generals Department *Copyright Reform : Copyright Amendment (Digital Agenda) act 2000* http://www.law.gov.au/publications/copyfactsheet/-copyfactsheet.html, June 2002

Appendix A

SMICLV MPEG-21 IPMP

This is the header file for the SMICL viewer part of the MPEG-21 messaging system.

A.1 MPEG21-IPMP.h

```
/*
 * MPEG21-IPMP.h
 *
 * Definitions for MPEG-21 IPMP Extensions
 * described in "MPEG-21 Intellectual
 * Property Management and Protection",
 * http://mpeg.telecomitalialab.com/working_documents.htm
 *
 * All smiclv IPMP plug-ins should #include this file.
 *
 * Author: Nicholas Paul Sheppard (nps@uow.edu.au)
 *         & Rickard Lonneborg (jrl45@uow.edu.au)
 *         School of Information Technology and Computer Science
 *         The University of Wollongong NSW 2522
 *         Australia
 *
 *      Created: 13th August 2001
 * Version 1.00: 27th February 2002
 */

#ifndef SMICL_MPEG21_IPMP_H
#define SMICL_MPEG21_IPMP_H

/* constants */
#define IPMP_TOOLIDLEN      16
#define IPMP_MAXCONTEXTID   0xffff
#define IPMP_TERMINAL       0
```

```
// basic types
typedef unsigned char   MPEG21_Octet;
typedef unsigned short  MPEG21_Short;
typedef unsigned int    MPEG21_Long;
typedef unsigned char   IPMP_ToolID[IPMP_TOOLIDLEN];
typedef unsigned short  IPMP_ContextID;

typedef MPEG21_Octet * MPEG21_OctetPtr;


/* message status */
typedef enum {
    IPMP_MSG_STATUS_MSG_POSTED = 0,
    IPMP_MSG_STATUS_INVALID_SENDER_ID,
    IPMP_MSG_STATUS_INVALID_RECIPIENT_ID,
    IPMP_MSG_STATUS_MSG_MODE_NOT_SUPPORTED,
    IPMP_MSG_STATUS_GENERIC_ERROR
} IPMP_ToolMsgStatus;


/* message mode */
typedef enum {
    IPMP_MSG_MODE_SYNC = 0,
    IPMP_MSG_MODE_ASYNC
} IPMP_MsgMode;


/* message interchange interface */
typedef IPMP_ToolMsgStatus  IPMP_ReceiveMessage_t(
                    MPEG21_Short    Sender,
                    MPEG21_Short    Recipient,
                    MPEG21_Long MsgSize,
                    MPEG21_Octet    Msg[],
                    IPMP_MsgMode    MsgMode,
                    MPEG21_Long MsgID
                );

/*************************************
 *
 * IPMP_ToolListDescriptor
 */
```

```
/* descriptor tags -- non-normative */
typedef enum {
    IPMP_ToolListDescriptor_tag = 1,
    IPMP_ToolParametricDescriptor_tag
} IPMP_Descriptor_t;

typedef struct {
    MPEG21_Long SizeOfArray;
    /*
     * MPEG21_Octet Data[SizeOfArray + 1];
     */
} ByteArray;

typedef struct {
    MPEG21_Octet    Tag;        /* descriptor tag */
    ByteArray   parametricDesc; /* parametric description */
} IPMP_ToolParametricDescriptor_t;

typedef struct {
    MPEG21_Octet    Flags;      /* flags */
        /*
         * isAltGroup;
         * isParametric;
         * reserved = 0b111111;
         */
    IPMP_ToolID ToolID;     /* tool ID */
    /*
     * if (isAltGroup) {
     *  MPEG21_Octet    numAlternates;
     *  IPMP_ToolID Alt_IPMP_ToolIDs[numAlternates];
     * }
     * if (isParametric)
     *  IPMP_ToolParametricDescriptor_t toolParamDescr;
     */
} IPMP_Tool_t;

typedef struct {
    MPEG21_Octet    Tag;        /* descriptor tag */
    MPEG21_Octet    numTools;   /* number of tools in list */
    /*
     * IPMP_Tool_t  ipmpTool[numTools]
     */
} IPMP_ToolListDescriptor_t;
```

```
/*************************************************************************
 *                                                                       *
 * Non-normative message syntax definitions. These may need to be altered if *
 * MPEG releases normative syntax for these messages.                    *
 *                                                                       *
 * Variable-length fields are given in comments within the structure     *
 * definition.                                                           *
 *                                                                       *
 *************************************************************************/

/* message purposes */
typedef enum {
    IPMP_CreateNewToolInstance_tag = 1,
    IPMP_RequestInstTools_tag,
    IPMP_AddToolInstNotificationListener_tag,
    IPMP_RemoveToolInstNotificationListener_tag,
    IPMP_ToolInstNotification_tag,
    IPMP_RequestToolContextID_tag,
    IPMP_SupplyToolContextID_tag,
    IPMP_DecConfigFromBitstream_tag,
    IPMP_MessageFromBitstream_tag,
    IPMP_DescriptorFromBitstream_tag,
    IPMP_DataTerminate_tag,
    IPMP_ProcessData_tag,
    IPMP_ProcessDataReturn_tag,
    IPMP_IntentRequest_tag,
    IPMP_IntentResponse_tag,
    IPMP_IntentTerminate_tag,
    IPMP_IntentRevoke_tag,
    IPMP_InitAuthentication_tag,
    ToolToUserMessage_tag,
    UserToToolMessage_tag
} IPMP_MessagePurpose;


/*************************************************************************
 *
 * IPMP_CreateNewToolInstance
 *
 */
```

```
/*
* Flags for the identifying a tools capabilities
*/

#define IPMP_CREATEINSTANCE_FL_ISPARAMETRIC 0x80
#define IPMP_CREATEINSTANCE_FL_RESERVED     0x7f

typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    MPEG21_Octet        Flags;          /* flags */
    IPMP_ToolID     ToolID;         /* tool ID from Tool List */
    /*
     * if (isParametric)
     *  ByteArray   ParametricDesc;
     */
} IPMP_CreateNewToolInstance_t;



/*****************************************************************************
 *
 * IPMP_RequestInstTools
 *
 */
typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
} IPMP_RequestInstTools_t;



/*****************************************************************************
 *
 * IPMP_AddToolInstNotificationListener
 *
 */
typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
} IPMP_AddToolInstNotificationListener_t;


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

```
 * IPMP_RemoveToolInstNotificationListener
 *
 */
typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
} IPMP_RemoveToolInstNotificationListener_t;


/*****************************************************************************
 *
 * IPMP_ToolInstNotification
 *
 */

#define IPMP_INSTANCE_INTENT_FL_VIEW        0x80
#define IPMP_INSTANCE_INTENT_FL_COPY        0x40
#define IPMP_INSTANCE_INTENT_FL_BROADCAST   0x20
#define IPMP_INSTANCE_INTENT_FL_AUTHENTICATE    0x10
#define IPMP_INSTANCE_INTENT_FL_RESERVED    0x0f


#define IPMP_INSTANCE_FL_NEED_RESOURCE   0x80
#define IPMP_INSTANCE_FL_NEED_DESCRIPTOR 0x40
#define IPMP_INSTANCE_FL_RESERVED        0x3f

typedef struct {
    IPMP_ToolID ToolID;             /* tool ID */
    IPMP_ContextID  ContextID;          /* context ID */
    MPEG21_Octet    DataFlags;          /* Data required by tool*/
    MPEG21_Octet    IntentFlags;            /* Tool capabilities */
    /*
     * if (needDescriptor)
     *   ByteArray  DescriptorMimeType
     */
} IPMP_Instance_t;

typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    MPEG21_Short        numTools;       /* number of tools in this message */
    /*
     * IPMP_Instance_t  Instance[numTools];
     */
```

```
} IPMP_ToolInstNotification_t;


/*****************************************************************************
 *
 * IPMP_RequestToolContextID
 *
 */
typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    IPMP_ToolID     ID;         /* tool ID to request */
} IPMP_RequestToolContextID_t;


/*****************************************************************************
 *
 * IPMP_IPMP_SupplyToolContextID
 *
 */
typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    IPMP_ContextID      ID;         /* context ID */
} IPMP_SupplyToolContextID_t;


/*****************************************************************************
 *
 * IPMP_ProcessData
 *
 */

#define IPMP_PROCESSDATA_FL_INPLACE 0x80
#define IPMP_PROCESSDATA_FL_RESERVED    0x7f

typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    MPEG21_Octet        Flags;          /* flags */
    MPEG21_Long     DataSize;       /* data length */
    /*
     * if (inPlace)
     *  MPEG21_Octet *  Data;
     * else
     *  MPEG21_Octet    Data[];
```

```
     */
} IPMP_ProcessData_t;


/*****************************************************************************
 *
 * IPMP_ProcessDataReturn
 *
 */
typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    MPEG21_Octet        Flags;
    MPEG21_Long     DataSize;       /* data length */
    /*
     * if (inPlace)
     *  MPEG21_Octet *  Data;
     * else
     *  MPEG21_Octet    Data[];
     */
} IPMP_ProcessDataReturn_t;


/*****************************************************************************
 *
 * IPMP_IntentRequest
 *
 */
enum {
    IPMP_INTENT_VIEW = 0x01,
    IPMP_INTENT_COPY = 0x02,
    IPMP_INTENT_BROADCAST = 0x03,
    IPMP_INTENT_AUTHENTICATE = 0x04
};

#define IPMP_INTENT_FL_DOMAIN       0x80
#define IPMP_INTENT_FL_PERMGRANTED  0x40
#define IPMP_INTENT_FL_RESERVED     0x3f

typedef struct {
    MPEG21_Octet        ID;         /* intent identifier */
    MPEG21_Octet        Flags;          /* flags */
} IPMP_Intent_t;
```

```
#define IPMP_INTENTREQUEST_FL_INPLACE   0x80
#define IPMP_INTENTREQUEST_FL_RESERVED  0x7f

typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    MPEG21_Octet        Flags;          /* flags */
    MPEG21_Long     DataSize;       /* data length */
    MPEG21_Short        numIntents;     /* number of intents */
    /*
     * if (needDescriptor)
     *     MPEG21_Long  numDescriptors
     *     ByteArray    Desc[numDescriptors];
     *
     * if (needResource && inPlace)
     *     MPEG21_Octet *   Data;
     * else if (needResource)
     *     MPEG21_Octet Data[];
     *
     * IPMP_Intent_t    Intent[numIntents];
    */
} IPMP_IntentRequest_t;


/*****************************************************************************
 *
 * IPMP_IntentResponse
 *
 */
#define IPMP_INTENTRESPONSE_FL_PROCESSED    0x80
#define IPMP_INTENTRESPONSE_FL_INPLACE      0x40
#define IPMP_INTENTRESPONSE_FL_RESERVED     0x3f

typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    MPEG21_Octet        Flags;          /* flags */
    MPEG21_Long     DataSize;       /* data length */
    MPEG21_Short        numIntents;     /* number of intents */
    /*
     * IPMP_Intent_t    Intent[numIntents];
     */
} IPMP_IntentResponse_t;
```

```
/*****************************************************************************
 *
 * IPMP_IntentTerminate
 *
 */
typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    MPEG21_Short        numIntents;     /* number of intents */
    /*
     * IPMP_Intent_t    Intent[numIntents];
     */
} IPMP_IntentTerminate_t;


/*****************************************************************************
 *
 * IPMP_IntentRevoke
 *
 */
typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    IPMP_Intent_t       Intent;         /* intent information */
} IPMP_IntentRevoke_t;


/*****************************************************************************
 *
 * ToolToUserMessage -- NORMATIVE
 *
 */

typedef struct {
    MPEG21_Short    ID;
    ByteArray   displayText;
} DTArray;

typedef struct {
    MPEG21_Short    ID;
    MPEG21_Short    SubID;
    ByteArray   promptText;
} RTArray;

typedef struct {
```

```
    MPEG21_Octet    Flags;
        /*
         * isExclusive;
         * reserved = 0b1111111;
         */
    MPEG21_Short    ID;
    MPEG21_Short    SubID;
    ByteArray   PromptText;
} OptionArray;

typedef struct {
    MPEG21_Octet        languageCode[3];    /* language code */
    /*
     * if (inclDisplayText) {
     *  ByteArray   titleText;
     * }
     * if (inclDisplayText) {
     *  MPEG21_Short    numDisplayText;
     *  DTArray     displayText[numDisplayText];
     * }
     * if (needReplyText) {
     *  MPEG21_Short    numPromptText;
     *  RTArray     promptText[numPromptText];
     * }
     * if (inclOptionSelect) {
     *  MPEG21_Short    numOptions;
     *  OptionArray option[numOptions];
     * }
     * if (inclSMIL) {
     *  MPEG21_Octet    Flags;
     *              isReferenced;
     *              reserved = 0b1111111;
     *  if (isReferenced)
     *      ByteArray   SMIL_URL;
     *  else
     *      ByteArray   SMIL;
     * }
     */
} IPMP_Language_t;

#define IPMP_TOOLMESSAGE_FL_DISPLAYTITLE    0x80
#define IPMP_TOOLMESSAGE_FL_DISPLAYTEXT     0x40
#define IPMP_TOOLMESSAGE_FL_REPLYTEXT       0x20
```

```
#define IPMP_TOOLMESSAGE_FL_OPTIONSELECT    0x10
#define IPMP_TOOLMESSAGE_FL_SMIL        0x08
#define IPMP_TOOLMESSAGE_FL_RESERVED        0x07


typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    MPEG21_Octet        Flags;          /* flags */
    MPEG21_Octet        numLanguages[3];    /* number of languages */
    /*
     * IPMP_Language_t  Language[numLanguages];
     */
} ToolToUserMessage_t;

#define IPMP_TOOLMESSAGE_OPT_FL_EXCLUSIVE   0x80
#define IPMP_TOOLMESSAGE_OPT_FL_RESERVED    0x7f;


/*****************************************************************************
 *
 * UserToToolMessage -- NORMATIVE
 *
 */

#define IPMP_USERMESSAGE_FL_REPLYTEXT       0x80
#define IPMP_USERMESSAGE_FL_OPTIONSELECT    0x40
#define IPMP_USERMESSAGE_FL_RESERVED        0x3f

typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    MPEG21_Octet        Flags;          /* flags */
    MPEG21_Octet        languageCode[3];    /* language code */
    /*
     * if (inclReplyText) {
     *  MPEG21_Short    numReplyText;
     *  RTArray     replyText[numReplyText];
     * }
     * if (inclOptionSelect) {
     *  MPEG21_Short    numOptions;
     *  MPEG21_Octet    optionResult[numOptions / 8];
     * }
     */
} UserToToolMessage_t;
```

```
/*****************************************************************************
 *
 * IPMP_InitAuthentication -- NORMATIVE
 *
 */
enum {
    IPMP_NO_AUTH_REQUIRED = 0x01,
    IPMP_NO_ID_DO_SECURE_CHANNEL = 0x02,
    IPMP_DO_ID_NO_SECURE_CHANNEL = 0x03,
    IPMP_DO_ID_DO_SECURE_CHANNEL = 0x04
};

typedef struct {
    MPEG21_Octet        Purpose;        /* message purpose */
    IPMP_ContextID      ContextID;      /* context ID */
    MPEG21_Octet        AuthType;       /* authorisation type */
} IPMP_InitAuthentication_t;


/*****************************************************************************
 *                                                                           *
 * IPMP API.                                                                 *
 *                                                                           *
 *****************************************************************************/

typedef struct {
    char *  text;
    int exclusive;
} option_t;

typedef IPMP_ToolMsgStatus  IPMP_DefaultCreateNewToolInstance_t(
                    MPEG21_Short    Sender,
                    MPEG21_Short    Recipient,
                    MPEG21_Long MsgSize,
                    MPEG21_Octet    Msg[],
                    IPMP_MsgMode    MsgMode,
                    MPEG21_Long MsgID,
                    IPMP_ToolID ToolID
                );

typedef MPEG21_Octet *      DefaultToolToUserMessage_t(
```

```
                    const char *display_title,
                    const char *display_text[],
                    const char *reply_text[],
                    const option_t *option_select[],
                    const char *SMIL,
                    MPEG21_Long *size
                );

typedef void            free_t(void *);

typedef void *          GetContext_t(IPMP_ContextID id);
typedef void *          SetContext_t(IPMP_ContextID id, void *p, free_t *pfn);
typedef int         mtoi_t(const void *p, int n);
typedef void *          itom_t(void *p, int c, int n);

typedef void            void_t(void);

typedef struct {
    IPMP_ReceiveMessage_t *         SendMessage;
    IPMP_DefaultCreateNewToolInstance_t *   DefaultCreateNewToolInstance;
    IPMP_ReceiveMessage_t *         DefaultProcessData;
    DefaultToolToUserMessage_t *        DefaultToolToUserMessage;
    GetContext_t *              GetContext;
    SetContext_t *              SetContext;
    mtoi_t *                mtoi;
    itom_t *                itom;
} IPMP_API_t;

#endif
```

Appendix B

REL Plugin

B.1 Rights.h

This file contains the declaration of the diLicense class, which handles XrML licenses.

```
#ifndef SMICL_DIRIGHTS_H
#define SMICL_DIRIGHTS_H


#include "StdAfx.h"

//
// diLicense class
//
class diLicense {
public:
    //
    // Constructor
    // Writes the license to a temporary file
    // Creates the XrML object
    // Loads the license
    //
    // Arguments  : license - XrML data as a null-terminated string
    //              Sender  - ID of the ipmp-terminal
    //              Recipient - ID of this tool
    //
    //

    diLicense(const MPEG21_Octet *license,
            MPEG21_Short Sender, MPEG21_Short Recipient);
    //
    // Validates if there is a grant for user
```

```
    // identified by the
    // 1 .identity key in the
    // xml\PrincipalTemplate.xml file
    // 2. Resource identified by the ID expression
    // 3. View right defined in the xml\RightsTemplate.xml file
    //
    // Arguments : ID - identifier string
    // Returns   : 1 if a grant found, otherwise 0
    //

    int validate(const MPEG21_Octet *id);
    //
    // Start the condition checking
    // Returns  : 1 if succeeded or no conditions to be checked, else 0
    //
    int CheckConditions();
    // Send a revoke message to the ipmp terminal
    // Arguments   : dwCookie  - cookie of the condition manager
    //               bstrURI   - failing condition namespace
    //               bstrBaseName - falining condition name
    void revoke(DWORD dwCookie, BSTR bstrURI, BSTR bstrBaseName);

private :
    IcgXrML *xrml;
    IcgElementList *cond;
    IcgElementList *princL, *rightL, *resL;
    static int LicId;
    long CondMgrCookie;
    MPEG21_Short    MyID, IPMPTermID;   // MPEG21_message information
    CComBSTR extractID(const MPEG21_Octet *ID);
    CComPtr<IUnknown>    condL;
    // Condition listener,
    // Set to IUnknown to resolve
    // cross-dependancy compile errors
};

#endif
```