Modeling a Core Module for MPEG-21 and AXMEDIS Content Manipulation Tools

P. BELLINI, P. NESI, L. ORTIMINI, D. ROGAI, A. VALLOTTI
DISIT-DSI, Distributed Systems and Internet Technology Laboratory
Department of Systems and Informatics, University of Florence/
Via S. Marta, 3 - 50139 Florence, Italy, Tel: +39-055-4796567, Fax: +39-055-4796363
nesi@dsi.unifi.it, http://www.disit.dsi.unifi.it

Cross media and hypermedia content formats and related tools have to have to take into account aspects related to the protection and management of intellectually property right along the value chain from content production to content distribution up to the final user. This trend is provoked by the usage of digital content and rights management in the business to business transactions and by the fact that final users are becoming every day closer to the content producers: posting their content, protecting their content and defining rules for its usage, etc. Therefore, new models and solutions to cope with content packaging and protected content manipulation are needed. On this view, the relevance of MPEG-21 standard is growing, while little has been performed about the modeling of authoring tools and players for the production/consumption of MPEG-21 digital objects. The design of MPEG-21 tools presents several critical points to be solved at architectural and modeling levels to ensure security and flexibility for manipulating any kind of digital resource that may be contained in an MPEG-21 package. This paper presents the evolution of models for content packing with a formalization that allows reasoning and verifying the model validity against extended requirements of the B2B area. The study has produced the AXMEDIS model and metadata (defined on top of MPEG-21 standard) for which a set of innovative software tools have been produced. The AXMEDIS model and components for content authoring/integration tools and final users' players, supports MPEG-21 and extends it according to the spirit of the standard. The proposed solution provides both data manipulation flexibility and a high level of security when digital resources are used for creating more complex objects, in the respect of intellectually property right. It can be used to develop a wide range of tools based on MPEG-21 standard and it is at the basis of the authoring and player tools realized for the AXMEDIS IST FP6 R&D European Commission Integrated Project (http://www.axmedis.org).

Categories and subject descriptors: D.2.6 [**Software Engineering**]: Programming Environments-- Integrated environments; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia—architectures; K.4.4 [**Computer and Society**]: Electronic Commerce--security.
General terms: Design, Languages, Verification
Additional Key Words: authoring tools design, content management, content distribution, content packaging, Digital Rights Management, e-Commerce.

1. INTRODUCTION

The solutions for digital content distribution and e-commerce are mainly grounded on the state of the art of multimedia content modeling, packaging, protection and distribution. Presently, there exists a large number of content formats that ranges from basic digital resources (documents, video, images, audio, multimedia, etc.) to integrated content packages such as: MPEG-21 ISO [MPEG Group], [Burnett et al., 2005], WEDELMUSIC [Bellini et al., 2003], SCORM [Mourad et al., 2005], OMA, TV-AnyTime Forum [Hulsen et al., 2004], etc. These integrated content formats try to wrap different kind of digital resources/files in a container/package with related information (e.g., content metadata and in general descriptors, and relationships among those resources), and making them ready for delivering (streaming and/or downloading), in plain (clear-text) and/or protected forms. In fact, some of the above mentioned solutions are enabling a large range of business and transaction models and provide some integrated DRM (Digital Rights Management) solution to cope with Intellectually Property Right (IPR), such as those based on MPEG-21 REL (MPEG Rights Expression Language) [Wang., 2003], [Wang et al., 2005] and OMA ODRL (Open Digital Rights Language) [Iannella, 2001], [Iannella, 2002], and others [Lin et al., 2005], [Lee et al., 2003].

The definitions and usages of content packages have been mainly due to the needs of satisfying a set of requirements related to the e-commerce/distribution of digital content from distributors to consumers (final users): the so called Business to Consumer (B2C) distribution. The most relevant requirements for B2C are the quality of service (that includes several features to be provided to the user's side) and the security of content delivered to maintain in some measure the control of what the user is consuming in terms of rights. The typical B2C scenario, which synthesizes the lifecycle of the related packaged content, is shown in Figure.1.

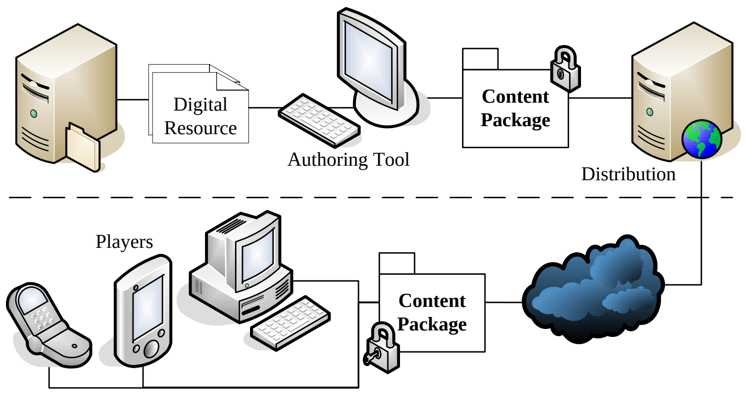

Figure 1 – Scenario on content package and its usage

The content distributor embeds raw content (digital resources) in order to produce a package for the distribution (e.g., a photo collection, a set of videos, an educational training course, a simple digital resource with metadata). The distributor on the basis of the contract with the producer (that, in turn, it is based on its contract with the content owner/author) should protect the produced package to maintain a certain control on the rights exploitation. Thus, the content distributor may make business granting the access to consumers producing specific licenses, that describe a set of rights granted to the consumers (the rights are the actions that can be performed on the content, such as play, print, copy, etc.). Once the package has reached the consumer's player, he/she may perform some action to exploit the acquired rights, for example he/she may decide to make a play. Thus, the player, in order to allow performing only authorized actions, has to verify if it can be authorized on the basis of the license (the license can be on an authorization server and/or cached/hidden into the player). Then, if authorized, specific information (typically called Protection Information or IPMP, Intellectually Property Management and Protection information, such as in MPEG-21 [Prados et al., 2005], [Lin et al, 2005]) have to be obtained to access the package (to unprotect the embedded digital resources) and, thus, to render/execute the multimedia content according to what has been specified in the license. To cope with the B2C cases, the content packages have to satisfy a set of basic requirements of interoperability, flexibility and scalability, and thus with the possibility of working with different:

- types of content (e.g., video, audio, documents, learning objects, images, games);
- formats for the embedded resources (e.g., for audio: wav, AIF, MP3; for documents: DOC, PDF, PS, HTML; and similarly for games, for video, for images, for data, etc.);
- metadata for covering needs of the final users to make queries and indexing and technical descriptions (e.g., Dublin Core, MPEG-7);
- models of distribution (e.g., streaming, downloading, progressive download);
- distribution channels (e.g., broadcasting, internet, mobile, kiosk);
- final user devices for resource rendering/playing (e.g., PC, STB, PVR, PDA, mobiles);
- business models (e.g., pay per play, all you can eat, renting, monthly rate).

The above requirements have to be combined with the needs of content producers and distributors that would maintain under control the security of their content, the exploitation of their rights [Lin et al., 2005], [Koushanfar et al., 2005], [Mourad et al., 2005]. These requirements are leading to provide different modalities of managing the package structure and the organization of digital resources and metadata and digital rights management.

According to the B2C model, when the license is not placed in the package with the content, it is also possible to produce it in a second phase, for example, when the rights are bought. This permits to produce the content in advance according to a large set of different packages (covering different resolutions and formats) for delivering it to the end user according to the needs/capabilities of the final user's player and preferences. This approach can be used, for example, by producing the packages applying a large set of digital adaptation and formatting algorithms to the initial digital resources before their packaging [Vetro and Timmerer, 2005].

With the diffusion of content in digital formats along the whole value chain (from production, integration, to the distribution and usage) the usage of the digital packages and protected objects are pervading the business area. Thus producing the needs of associating with the digital content suitable protection models for Business to Business (B2B) transactions. These needs are putting in evidence a large set of new requirements that need to be satisfied by the packaging formats. This evolution is part of the natural migration from traditional to digital media of the whole value chain and it is bringing a large set of benefits to the whole area of digital content business. Among the most evident benefits of the adoption of these models for the B2B we can see the possibility of:

- sharing content as protected packages (with some digital rights management) with other business actors for which the business transaction and agreement is formalized by means of a license per package and/or per resource;
- reducing promotion and business distribution costs since the content can be distributed in protected formats among the business users. The same content package can be reused/nested by others for creating more valuable packages without the possibility of

hiding or reducing the evidence and rights of the initial content ownership according to the license;

- authoring, manipulating and integrating protected and non protected content. This can be performed to (i) produce added value content and packages (e.g., creating a collection of audio files from other packages, using packaged content with audio segments into educational courses), (ii) adapting content packages (which includes protected and/or non protected digital resources) in real-time for the production of content on demand;
- tracking the origin/path of the digital content along the value chain, along the chain of who has created/nested packages. Thus increasing the control about exploited rights for the content owners/authors, producers and integrators with respect to the other actors of the value chain and to the distributors;
- enabling the final users to produce content and considering them as effective producers/authors that may produce and provide licenses to other users and companies (thus, relaxing the difference from producer/distributor and the final users). This is, for instance, what is going to happen in (i) a very simple manner with Video Google (http://video.google.com/) in which the final user may post its home-made video on which he/she may impose some licensing rules about its usage or can give it for free; or (ii) in many web sites in which content is licensed on the basis of a some Creative Commons License (http://creativecommons.org/). In the first case, the package model used by Google allows the management of simple single resource content and the whole solution permits the enforcement of some DRM rule into the corresponding player, while in the case of Creative Commons the license is only textual and thus the enforcement of rights verification on tools is not provided.

In order to provide the above mentioned features, the development of a set of enabling technologies is needed.

The extension of the package model to cope with B2B requirements and in particular for addressing:

- any kind of digital resources, including professional formats and cross media formats integrating different resources: audio, video, document, images, etc.;
- metadata information for B2B and not restricted to any specific content and metadata formats, and maintaining them permanently associated with the content;
- structure of content/package including and integrating together protected and non protected content, organizing nesting levels, and maintaining visibility to the metadata;
- indexing and querying of non protected and protected objects in a simple manner;
- protection information, allowing the applications of multiple protection models to different resources into the same package;

- distribution of content package in a safe and simple manner even when the content contains nesting levels of packaging;
- licensing, allowing the formalization of business licenses that give the possibility of exploiting more complex features such as enhancing, adaptation, extraction, copy, excerpting, and to produce licenses for reselling;
- maintaining the same content package structure and information from the instant or its production, along the value chain, in all cases in which it is reused, to its final usage on the end user player according to the licensed rights for each digital resource;

The support of a secure and efficient authoring/player tool architecture with a core module for supporting the above package and model according to the licensed rights (DRM rules formalized in the license) for digital resources, allowing

- supporting the above mentioned features for the package model, which has to be portable on different devices and thus reusable in different contexts;
- compounding packages in larger packages in an easy manner, considering digital resources together with their metadata for rendering/modifying: layouting content, adapting and processing digital resources and content, etc. The model has to be extensible in order to accept new types/models of content (digital resources) and/or metadata and to be ready to evolve according to the standard updates and extensions;
- protecting simple and nested content packages, leaving the possibility of accessing to visible metadata of nested included levels;
- enforcing security into software applications (authoring tools and/or players), providing of an Application Program Interface to access at the functionalities controlling rights exploitation;
- manipulation of package structure and content (such as nested levels of packaging, hierarchies) during authoring in a safe manner without permitting at the developers to create non secure editors, players and devices;
- supporting the production of specialized commands or custom commands for enforcing the rights corresponding to those imposed by the formal license model of the DRM and rights data dictionary and semantics.

Among the mentioned packaging models, the MPEG-21 results to be the closer to the above mentioned needs and allows being customized/extended. The MPEG-21 standard addresses packaging, adaptation profiles, protection and licensing formats [Burnett et al. 2003] and it is mainly based on the concept of Digital Item, DI, for the packaging [Burnett et al., 2005]. The relevance of the standard is growing while little has been done up to now about the modeling of tools for the production/consumption of MPEG-21 DIs. This is partially due to the fact that, the design of an MPEG-21 core module for authoring/playing is a complex task presenting several critical points. In fact, a core module has to guarantee at the same time (i) easy and fast access at the modeled information and digital resources; (ii) high security level enforcing the DRM on the digital resources usage; (iii) usability of the solution for designing and developing authoring tools and players; (iv) extensibility for accepting new commands and functionalities (enforcement of rights) for any type and new types of content formats. These requirements drive the design in opposite directions (accessibility vs. security) making difficult to cope with these aspects within a unique solution.

Moreover, multimedia and cross-media authoring are complex tasks that involve several aspects such as composition, layouting, adaptation, synchronization, performance [Bulterman and Hardman, 2005]. In addition, in a world in which the DRM is needed the accessibility of digital resources (that may be protected at nesting levels into the data structure), the packaging model and finally the protection models are also relevant. In fact, protecting a cross-media content means to protect the single elements and the resources connected to them (e.g., an HTML/XML page and related digital resources included/connected), this means that the player has to recover these resources not by following the initial (absolute or relative) links but by using specific links towards the protected object package.

This paper presents the evolution of models for content packing with a formalization that allows reasoning and verifying the model validity against functionalities and extended requirements. The study has produced the AXMEDIS package model as a specialization of MPEG-21 model. The work has been developed in AXMEDIS (Automating Production of Cross Media Content for Multi-channel Distribution) which is a large research and development Integrated Project FP6 of the European Commission (http://www.axmedis.org), [Bellini and Nesi, 2005]. The package model produced is capable to cope with the above mentioned requirements/problems and it is supported by an architecture and tools based on a core module called AXMEDIS Object Model, AXOM. It has been used to develop MPEG-21 as well as AXMEDIS compliant tools coping with the mentioned problems and more specifically those about flexibility and security. In AXMEDIS, the MPEG-21 has been selected as the underlying packaging model with some extensions reported in this paper that are improving its usability in the scenarios in which content packages are protected for their usage in the B2B and B2C areas along the whole value chain (from the producer to the consumer passing for the integrators and distributors). Many issues addressed in AXMEDIS are not replicated into the MPEG-21 standard since the latter has a different purpose. In particular, aspects related to the definition of a specific software architecture that could support tool design for consuming digital goods are not in the precise scope of MPEG-21. The proposed model, architecture and solution presented in this paper are currently used by several AXMEDIS tools [Bellini at al, 2005].

2. MPEG-21 Short Overview

MPEG-21 is mainly focused on the standardization of the content packaging and other formats related to DRM aspects. In particular, MPEG-21 scope is mainly related with the content package formats leaving completely undefined system architectures, business models, authoring, etc. The standardization process of MPEG-21 is still under completion [MPEG Group]; presently most of the parts are mature while others are under evolution. The most relevant parts for the work described in this paper are: Digital Item Declaration (DID) [MPEG-21 DID] and the Intellectual Property Management and Protection (IPMP) [MPEG-21 IPMP]. Regarding the DRM capability for granting access to content, other parts to be considered are: Rights Expression Language (REL) and Right Data Dictionary (RDD) [MPEG-21 RDD]. The DID defines how DIs (Digital Items) have to be represented. A DI is a structured digital object and is the fundamental unit of distribution and transaction within the MPEG-21. A DI is a package for digital resources and related metadata, it is represented as an XML document which fulfils the DI Declaration Language (DIDL) schema. DIDL provides placeholders for metadata that can contain any other XML format. IPMP provides the means to include protected content elements inside a DI. Every DIDL element can be replaced by a protected version, its corresponding IPMP element. The latter contains the original DIDL element in a protected form together with some required information to perform un-protection (i.e., applied protection tools, related license services, etc.).

3. AXMEDIS Package Model and the MPEG-21 standard

The MPEG-21 standard has been defined to satisfy different application domains, ranging from B2B and B2C; and remains enough abstract to avoid putting restrictions that are needed for the effective coverage of specific cases as those described above. In this section, the AXMEDIS model is presented. It is an extension of MPEG-21 DIDL by following the spirit of the standard that leaves flexibility to expand the model. Thus, the expansion is in some how a natural operative method of applying/profiling the MPEG-21 formats. In AXMEDIS, extensions to MPEG-21 model have been realized in order to obtain a specific kind of MPEG-21 Digital Item that can satisfy the above mentioned requirements in term of package structuring, managing and mandatory metadata. The authors of this paper, while trying to exploits MPEG-21 technology, have contributed to extend the standard to allow covering some of the above mentioned B2B requirements and in particular to remove a limitation that were present in the previous versions of MPEG-21, to allow the management of metadata for protected objects as described in this paper. Considering the above mentioned requirements, most of them are covered by the MPEG-21 package model and elements. In facts: DIDL allows including any kind of digital resources, including professional formats typically non delivered to the final users and not only audiovisual; IPMP allows the applications of multiple protection models to different resources into the same package; REL/RDD allows the formalization of licenses that give the possibility of exploiting complex features such as adaptation, extraction, copy, excerpting, and to produce licenses for reselling. Other requirements mentioned above can be satisfied only if the package model is extended for addressing:

- metadata information for B2B and not restricted to any specific content and metadata formats. MPEG-21 is not defining specific metadata for B2B but supports their insertion in the model;
- structure of content/package including and integrating together protected and non protected content, organizing nesting levels, while maintaining visibility to the metadata. MPEG-21 DI and IPMP do not describe how to use the protection models and mechanisms to preserve the visibility of metadata;
- indexing and querying of non-protected and protected objects in a simple manner. This requirement can be supported only if both protected (with nested levels of protection) and non protected objects present a uniform model for indexing the content into the database by using suitable B2B metadata structure for nested levels. The satisfaction of this requirement is possible only by the definition of a specific uniform model of metadata organization for protected and not protected objects, even in the presence of nested levels;
- distribution of content package in a safe manner. This requirement mainly depends on the distribution tools, and the package manipulation and the visibility of metadata; it is very relevant for indexing any kind of objects;
- maintaining the same package model from the production in the business area to its final usage in the end-user player (or authoring tool in the case of business user) according to the licensed rights for each digital resource. To satisfy this requirement is possible only by the definition of specific methodology/semantics to cope with metadata when the object is protected and when is not protected;

These features are concretized in the model with the MPEG-21 extensions described in the next subsections. Please note that any AXMEDIS object is an MPEG-21 compliant object, while the opposite is not always true (not all the MPEG-21 objects are AXMEDIS objects).

3.1. B2B Needs and AXMEDIS Information

The B2B content distribution and trading requires the mandatory presence of certain information in order to have an immediate knowledge of the origin of the newly discovered content package and facilitate the trading of content without the needs to receive additional information. The description of the content with B2B metadata allows a better exploitation of content in a B2B community where new digital products are shared and promoted by Business parties even via P2P tools and mediations. To this end, in the AXMEDIS model, a set of metadata have been included such as: Dublin Core for the classification, a number of identifications IDs (among them the AXOID is mandatory and unique, AXMEDIS object ID, while others can be added such as ISRC, ISBN, ISMN, etc.), descriptors in MPEG-7 format and general XML, etc. In addition, other specific descriptors/metadata to cope with the object trading have been added and grouped in the so called AXInfo (AXMEDIS Information):

- Identification of the Content Creator: who has produced the digital content, general information and a specific ID;
- Identification of the Content Owner: who has the original right of the artistic work from which the content has been created, general information and a specific ID;
- Version code, object status and object type, and other typical information for managing the evolution of the object state;
- List of potentially available rights that can be acquired for that content object. They are the formalization (in terms of a simplified MPEG-21 REL) of the rights that potentially can be bought from the owner/creator for that content;

This information is related to each digital item contained into the object. Each digital object may be organized according to linear or hierarchical model in which each object may contains other objects creating in this way several nesting levels. Thus, in general, the model is not linear and the metadata may be present at each level and the objects can be protected or not creating nesting levels of protection. This is what typically happen when a complex cross media content is produced for example to build a training course, a DVD, a multimedia collection, etc.

3.2. Basic Model for Composition of protected and unprotected objects

The package models are typically structured as an aggregation of descriptors (including metadata and identification codes), and components (including digital resources and eventually nested packaged objects) according to a recursive structure. The package model has to allow protection of the included content and formally can be defined as *Package Model PM1:*

```
Type Pack := clear-obj[list Desc, list Comp] | prot-obj[Prot-info, Enc]
Type Desc := …any descriptor…
Type Comp := Res | Pack
Type Res := …any digital resource…
Type Prot-info := …a set of protection information…
Type Enc := …any valid encoding for protection…
```

The protection is a transformation of the clear object in the corresponding protected form, where the whole package is encoded (e.g., encrypted) according to the chosen protection technology [Lin et al., 2005]. This model is typically used for content distribution of single digital resources with conditional access and DRM even in its simpler non hierarchical version in which only single components are include into the package -- i.e., `Comp := Res`

In MPEG-21, protecting content means to replace the readable DIDL element in "clear-text" containing the content and digital resources with an IPMP element which is encoded in some non readable manner.

In some specific structures of content, when nested levels of content/item composition are present, non protected and protected objects may appear at different nesting levels (and the latter provoke the substitution of specific IPMP elements). This means that content element can be protected and included/collected in other items which in turn may be also protected as a whole, etc. Thus, according to the model defined above, the protection of a package *p* is performed by transforming it into protected object `prot-obj` according to the Protection Information `pi` which produces the encoded chunk by using (in this case, for example) the function `encrypt()`. The opposite operation of `unprotect()` allows us to get back the package `p` in the original format.

```
        protect(p:Pack, pi:Prot-info) = prot-obj[pi, encrypt(p)]
        unprotect(prot-obj[pi, encrypt(p)]) = p
```

A fundamental feature for the B2B area of the content package is the accessibility of the metadata in the hierarchy of the content structure. The application of operations `getmetadata()` for extracting metadata and for taking the content and content element produce the following results:

```
getmetadata(clear-obj[d:list Desc, c:list Comp]) = d
getmetadata(prot-obj[pi:Prot-info, e:Enc]) = Ø
getcontent(clear-obj[d, c], pos:Position) = getelementat(c, pos)
getcontent(prot-obj[pi, e], pos) = Ø
```

Thus, for package model PM1, content access is prevented by the protection, and the metadata are not accessible without a previous un-protection of the content package.

Two examples are analyzed in the following for PM1: (i) a simple content package which embeds a single digital resource with its metadata, and (ii) a more complex package with a multi-level hierarchy. Both are defined as clear-text objects and then protection is applied.

Let us now analyzing the first case for PM1 in which we start from producing a package: `clear-pack1`:

```
Var clear-pack1:Pack := clear-obj[<d1:Desc, d2:Desc>, <res1:Res>]
Var prot-pack1:Pack := protect(clear-pack1, pi1:Prot-info) =
        prot-obj[pi1, e1:Enc]
```

Where: `Var e1 := encrypt(clear-pack1);`

Thus: `getmetadata(clear-pack1) = <d1, d2>`, and `getmetadata(prot-pack1) = Ø`

As a result, `prot-pack1` does not present any accessible information about internal descriptors contained into the original object. Please note that the only way to retrieve content description is to unprotect the package.

```
getmetadata(unprotect(prot-pack1)) = getmetadata(clear-pack1) = <d1,d2>
```

Let us now analyzing a second case for PM1 in which we start from producing a complex package with some nesting levels:

`Var o2:Pack := clear-obj[<d3:Desc>, <o3:Pack, res3:Res>]`

Where: *`Var o3:Pack := clear-obj[<d4:Desc>, <clear-obj[<d5:Desc>, <res5:Res>],`*
`clear-obj[<d6:Desc>, <res6:Res>]>]`

Thus: *`getmetadata(o2) = <d3>`*

`getmetadata(getcontent(o2, 0)) = getmetadata(o3) = <d4>`

`getmetadata(getcontent(getcontent(o2, 0)), 1)) = <d6>`

Then, *`o2-bis`* object is created by starting from *`o2`* protecting the nested object *`o3`*:

`Var o2-bis:Pack := clear-obj[<d3>, <protect(o3, pi3:Prot-Info), res3>]`

Obtaining: *`getmetadata(o2-bis) = <d3>`*

`getmetadata(getcontent(o2-bis, 0)) = getmetadata(prot-obj[pi3, e3]) = Ø`

Please note when protecting in a content sub-tree the metadata regarding the leaf of the composed content or those placed in the lower levels of the hierarchy are obviously included in the protected content (e.g., IPMP). This makes hard to make accessible the metadata information and could be avoided only by additional processing in copying those metadata while protecting, thus changing the object structure.

For example, by using the MPEG-21 notation, there are no possibilities to access metadata of the content parts or of the components of protected objects. Only if IPMP elements are substituted with the corresponding DIDL elements (by means of IPMP tools to unprotect the object) the metadata may become accessible again. In the following, an MPEG-21 based example is reported for an object which contains two leafs resources.

```
DIDL
   Item
         Descriptors
            Metadata of the composed content
         Item
            Descriptors
               Metadata of content part 1
            Component
               ...
         Item
            Descriptors
               Metadata of content part 2
            Component
```

That becomes in its protected form:

```
DIDL
   IPMPDIDL:Item
         IPMPDIDL:Info
         IPMPDIDL:Content
            Encrypted composed content
```

3.3. Extended Model for Composition of protected and unprotected objects

In order to facilitate the B2B negotiation and transaction, content must provide always accessible metadata even when the content is protected. Any content producer, interested in producing, selling and/or reselling simple of complex content for distribution, should have the possibility of deciding which metadata have to be accessible (public) to other B2B customers/users and which of them have to be left protected without making complex their management and without leaving to other users of its content the possibility of hiding its public metadata. This is a requirement that cannot be satisfied by using the PM1 model presented above. In order to satisfy these needs, a more powerful model is needed.

Therefore, two kinds of descriptors modeling respectively public and private content descriptions have to be present in the structure of the package. Public descriptors can contain descriptions associated with a whole sub-tree as protected content. Thus, when the object is protected a "public descriptor tree" structure has been defined in order to describe the content after its protection respecting the hierarchical tree of the internal nested levels. On this aim, the following extended package model has been defined and referred to *Package Model PM2:*

```
    Type Pack := clear-obj[list Desc, list Comp] |
                      prot-obj[Prot-info, Pub-desc-tree,  Enc]
    Type Desc := Pub-desc  | Priv-desc
    Type Pub-desc := …any public descriptor…
    Type Priv-desc := …any non public (private) descriptor…
    Type Comp := Res | Pack
    Type Res := …any digital resource…
    Type Prot-info := …a set of protection information…
    Type Pub-desc-tree := desc-tree[list Pub-desc, list Pub-desc-tree]
    Type Enc := …any valid encoding for protection…
```

Where: Type *Pub-desc-tree* represents a tree structure of public descriptors according to the nested levels. According to this model, a set of functionalities is needed for extracting from content parts the entire set of related public descriptors:

```
getpublicdesc(res:Res) = Ø
getpublicdesc(clear-obj[d:list Desc, c:list Comp]) =
    desc-tree[<∀ds:Desc in d. ds is Pub-desc>,
               <∀cp:Pack in c. getpublicdesc(cp)>]
```

Please note that *getpublicdesc()* produces a tree of public descriptors by recursively browsing inner components of the content into the package:

```
getpublicdesc(prot-obj[pi:Prot-info, pdtree:Pub-desc-tree, e:Enc]) = pdtree
```

In order to grant accessibility of content description the function for package protection has to use the descriptor extraction (*getpublicdesc())* in order to add the hierarchical description in the protected element. This allows retrieving metadata and identification from any protected object.

```
protect(p:Pack, pi:Prot-info) = prot-obj[pi, getpublicdesc(p), encrypt(p)]
```

Thus: *unprotect(prot-obj[pi, pdtree:Pub-desc-tree, encrypt(p)] = p*

Function `getpublicdesc()` can now rely on the suitable element of the protected package to obtain the description that is related to what is protected.

```
getmetadata(clear-obj[d, c]) = d
getmetadata(prot-obj[pi, pdtree, e]) = pdtree
getcontent(clear-obj[d, c], pos:Position) = getelementat(c, pos)
getcontent(prot-obj[pi, pdtree, e], pos) = Ø
```

In the following, two similar examples to what has been presented for the PM1 model are examined for the proposed PM2 model. In both of them, all the descriptors, even those related to the inner elements are accessible in a protected package without the need of using the unprotect function.

Let us now analyzing the first case for PM2 mode in which we start from producing a package: `clear-pack1`. From this example, the metadata can be easily accessed on the basis of the model proposed.

```
Var clear-pack1:Pack := clear-obj[<d1:Pub-desc,d2:Priv-desc>, <res1:Res>]
Var prot-pack1:Pack := protect(clear-pack1, pi1:Prot-info) =
        prot-obj[pi1,desc-tree[<d1>, Ø], e1]
```

Where: `Var e1:Enc := encrypt(clear-pack1);`

Thus: `getmetadata(clear-pack1) = <d1, d2>`

`getmetadata(prot-pack1) = desc-tree[<d1>, Ø]`

Let us now analyze a second case for PM2 in which we start from producing a complex package with some nesting levels and it is shown that the model allow accessing to the public metadata even nesting objects into protection nesting layers:

`Var o2:Pack := clear-obj[<d3:Pub-desc>, <o3:Pub-desc, res3:Res>]`

Where: `Var o3:Pack := clear-obj[ <d4:Pub-desc>, < clear-obj[<d5:Pub-Desc>, <res5:Res>], clear-obj[<d6:Pub-Desc, d7:Priv-desc, d8:Pub-desc>,<res6:Res>] >]`

Thus: `getmetadata(o2) = <d3>`

```
getpublicdesc(o3) = desc-tree[<d4>, < desc-tree[<d5>, < >],
                              desc-tree[<d6, d8>, < >] >]
```

According to the PM2, the protection of object `o3` into `o2` is performed obtaining `o2-bis`. This action is performed by transforming the object into `prot-obj` with the Protection Information model including the “public” descriptor and the coding of the whole `o3`:

```
Var o2-bis:Pack := clear-obj[<d3>, <protect(o3, pi3:Prot-info), res3>] =
        clear-obj[<d3>, <prot-obj[pi3, pdtree3, e3], res3>]
```

Where: `Var pdtree3:Pub-desc-tree := desc-tree[<d4>, < desc-tree[<d5>, < >],`

```
                desc-tree[<d6, d8>, < >] >]
        Var e3:Enc := encrypt(o3)
```

As a result, `o2-bis` continues to present accessible descriptors of all the elements contained into the package and the public descriptor tree of the protected version of `o3` is structurally identical to *o3* considering all its subcomponents.

```
getmetadata(o2-bis) = <d3>
getmetadata(getcontent(o2-bis, 0)) = getmetadata(prot-obj[pi3, pdtree3, e3]) =
        desc-tree[<d4>, < desc-tree[<d5>, < >], desc-tree[<d6, d8>, < >] >]
```

3.4. Considerations on PM2 model

Content needs to be protected in order to be safely distributed and exploited under DRM governance. AXMEDIS exploits MPEG-21 IPMP technology in order to produce protected content on the basis of PM2 model. Since MPEG-21 IPMP has standardized how to protect MPEG-21 DI, AXMEDIS has selected which DI tent parts have to be protected. In such a way, protected AXMEDIS Objects remain valid MPEG-21 DIs. Furthermore, AXMEDIS model imposes the IPMP component to contain information which is specific for B2B application domain, as described above. The protected version of AXMEDIS object elements may expose additional protection information structured. As the DIDL model, the IPMP has been designed by considering flexibility to deal with any protection mechanism. Also in this case, AXMEDIS has refined the MPEG-21 standard in order to address B2B aspects. The above presented PM2 model of AXMEDIS has been partially included enhancing the MPEG-21 IPMP standard by introducing the capability of adding a description to the protected content.

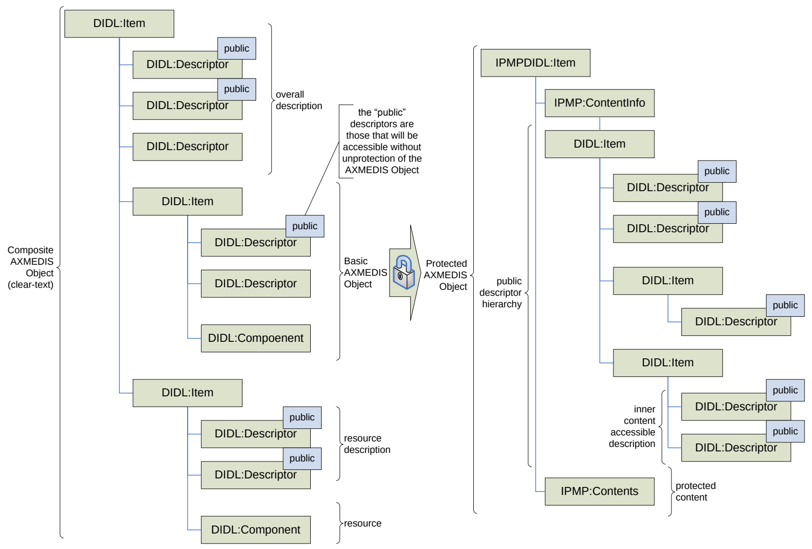

Figure 2 – AXMEDIS Object after protection in terms of MPEG-21 elements

In Figure 2, the content transformation after a protection process in terms of MPEG-21 elements has been sketched. To make the figure simpler and immediate, only the relevant MPEG-21 elements have been considered. The Descriptor element, as an example, is not a leaf since it can contains a Statement as its child element. The diagram puts in evidence which part of the content structure remain accessible even after the protection. Please note that some

Descriptor elements have been tagged with “public” attribute. The “public” attribute has been introduced by AXMEDIS to be sure that some content information will be accessible without needing an un-protection after any protection process. The attribute “public” allows defining the `Pub-desc` of PM2 model. The concept is that even the root level of the AXMEDIS Object hierarchy is protected, all the “public” metadata will be copied on the appropriate *ContentInfo* element, which has been designed to contain description about the protected content. Since the content structure its hierarchical, it has been decided to mimic the content structure and use MPEG-21 DIDL hierarchical element Item. AXMEDIS authoring tool is capable to apply multiple protection algorithms on each content element [Nesi et al., 2006].

4. AXMEDIS Object Manager

The AXMEDIS Object Manager (AXOM) is a module, which addresses MPEG-21 general modeling and other additional features which have been outlined in the previous sections. The AXOM can be used to create software tools which handle DIs in the respect of the DRM rules. The AXOM is used in the AXMEDIS tools that have to cope with content objects and it is compliant with MPEG-21, extending it with additional features and data structures. Thus, it can be used as a basis to design and develop a large range of e-commerce applications which are able of manipulating both MPEG-21 DIs and AXMEDIS Objects, for both editing/authoring and playing content objects. One of the most important features of the AXOM is to provide the means to create a trusted environment. Moreover, it is capable to support: MPEG-21 packaging model; packaging model PM2 of AXMEDIS; manipulation/access of package structure and content (such as nested levels of packaging, hierarchies) in the respect of DRM licenses; production of specialized commands or custom commands for enforcing the rights corresponding to those imposed by the formal license model of the DRM.

The AXOM is capable to guarantee that a tool built on its accessible commands is controllable with respect to the user actions on content object. Thus, different actions on the content model should require different grants (i.e., authorizations). Actions can target the content structure, the resources and the metadata. Thus a unique flow to handle verification of any action behavior performed on the content has been conceived.

In order to build a reusable and flexible infrastructure, the AXOM has been decomposed in five parts as shown in Figure 3, separating responsibilities of the several elements: **MPEG-21 Object Model** responsible of managing the MPEG-21 DI allowing content access and manipulation; **AXMEDIS Object Model** a refinement and an extension of MPEG21 DI, targeting the mentioned requirements and realizing on top of the MPEG-21 the PM2 model and semantics. It also allows to access and manipulate high-level features in the underlying MPEG-21 structure; **AXMEDIS/MPEG-21 XML Loader** to load MPEG-21 as well as AXMEDIS Objects from a given input source as a byte stream. The source can be textual XML or in a corresponding binary form. The Loader has the responsibility of validating the correct structure of the input source; **AXMEDIS/MPEG-21 XML Saver** to write to an output destination an MPEG-21 DI, thus also a valid AXMEDIS Object. This module is optional since it is not needed to build a player; **AXMEDIS Command Manager** to provide an Application Program Interface for accessing to the content package elements. The interface is used for both playing and authoring AXMEDIS, and MPEG-21 content packages. The Command Manager allows transparent manipulation of content in respect to DRM rule and directly accessing to the Servers for the acquisition of the License and of Protection Information. It allows to access or to manipulate content with a built-in authorization check on the basis of MPEG-21 REL.

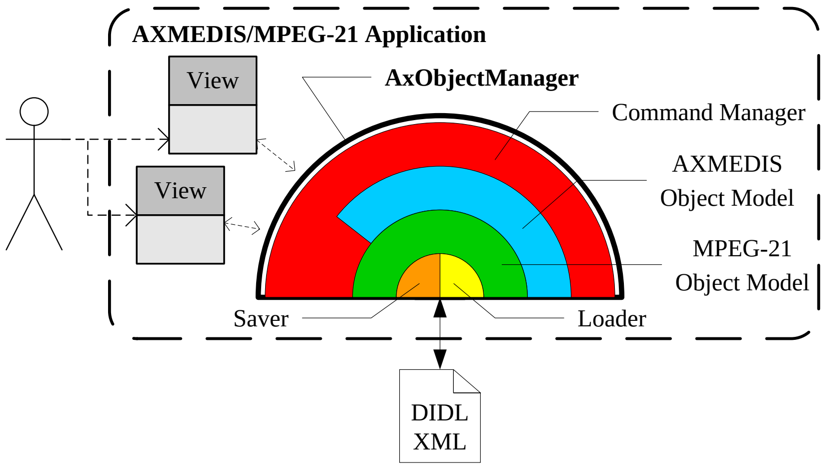

Figure 3 – AXMEDIS Object Manager (AXOM) layers

Defining features on object model that manage MPEG-21 content packages grants to the system the ability to deal with all kinds of MPEG-21 objects and not only with AXMEDIS content packages. Provided AXOM API is capable to cope with MPEG-21 object model directly (as shown in the diagram) to allow manipulation of MPEG-21 DIs.

The AXOM architecture supports extendibility, new types of objects based on MPEG-21 model can be easily managed in MPEG-21 Object Model without changes in present structure. Moreover, only minor changes are needed to process content models that extend MPEG-21 or AXMEDIS elements definitions. Hooks to perform operations on content packages are provided by the upper level layer by means of *AxObjectManager* and *AxCommand* set. These classes offer: (i) a range of executable commands targeting the manipulation of both AXMEDIS and MPEG-21 object models, (ii) coordination of functionalities of lower levels modules to organize a reliable and protected execution flow.

The set of commands is easily extendable to manage future needs of upper level applications as shown in the sequel. Execution of defined commands is performed using class *AxObjectManager* as shown in the sequel.

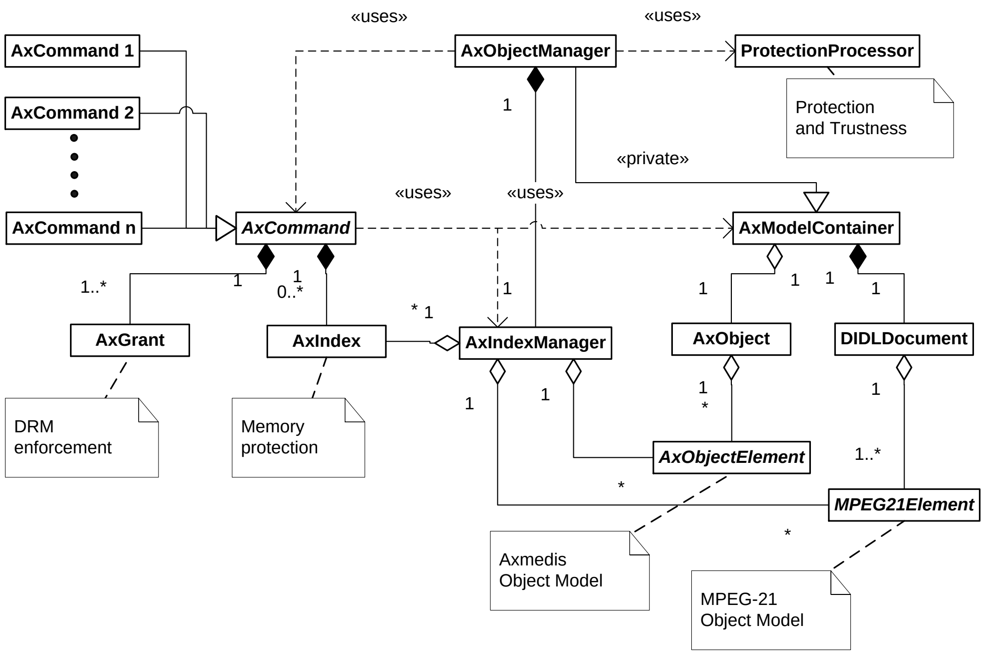

Figure 4 – AXOM class hierarchy

As depicted in Figure 4, the Command pattern has been applied [Gamma et al., 1995]. The **controller** is the *AxObjectManager* which exposes the AXOM API. The *AxObjectManager* is an intermediate layer between the application views and the object models (i.e., the object/package model with resources and descriptors). A view cannot directly manipulate the object model, while it has to issue **commands** to the *AxObjectManager* class. The latter is in charge of performing the requested actions on the model. In particular, each conceivable command has been realized as a class which implements the interface *AxCommand*. It exposes two main methods: *execute* and *getRequiredGrants*. The *execute* method of *AxCommand* has to be implemented by each specialized class to actually perform the action on the object model. The command execution method is able to directly access to the data model without any restriction, since the command is executed only if the authorization is obtained. The *getRequiredGrants* method of *AxCommand* allows the verification of the requested grants. Thus, the *AxObjectManager* class is able to handle the request received by the user and the set of conceivable commands can be augmented without any impact on the current architecture.

The *AxObjectManager* has few control points, the code to verify the commands can be easily inserted in. In fact, the *AxObjectManager* class is not in charge of verifying the grants, while it has been designed in order to provide hooks to easily create control mechanisms. The DRM enforcement is modeled by the declaration of the requested rights which has to be stated by each *AxCommand* inherited class. In fact, to each command is associated a list of *AxGrant*, which are the basic arguments for issuing the request to the authorization service. In that way, the *AxObjectManager* is able to generically handle any request issued by the user, respecting governance. Please note that, the *AxObjectManager* is not in charge of verifying the grants, since it has been designed to provide hooks, by means of which it is possible to enforce control mechanisms. In fact, it delegates to an authorization service the task of determining if the user has been authorized or not on the basis of the license. The AxObjectManager exposes functionalities for: creating new content; opening existing content by indicating a URI; browsing content structure; accessing metadata and resources embedded or referred to by the content; manipulating content structure, metadata and resources; saving the content; adding/modifying protection information associated to any content element.

By using model encapsulation, a good level of security has been achieved in terms of robustness against developer's malicious content handling preventing direct manipulation from the views. Thus, views are not allowed to target content elements by using pointers. Command targets have to be indicated using logical references that prevent the access to the physical addresses referring to the digital resources. Thus, the view interacts with the model using instances of the class *AxIndex*. These are used like pointers, while they can be actually resolved only by the *AxObjectManager* which generated them.

All the security aspects are demanded to Protection Processor class. This class has two basic roles: (i) performing verification on the component which is using *AxObjectManager* (i.e., the content manipulation application) to allow detecting typical tampering activities, thus to be confident on the software integrity; (ii) protect/unprotect content elements and to query for authorization of requested manipulations in a transparent manner with respect to the connected services. When digital resources have to be accessed (e.g., for their rendering on a rendering component/module view) the chain of un-protection tools is activated, thus allowing to establish a direct stream from the encapsulated resource and the rendering component/module view. Command execution returns content elements information, providing a reference to a clone. When an action expects object elements as results, *AxObjectManager* provides clones of them to the view. Since the view does not obtain the direct reference to the content element contained in the package, the action targets, like the source and the destination of a move command, have to be indicated using indexes: i.e., logical references to the real content elements.

4.1. MPEG-21 and AXMEDIS Data Models

The MPEG-21 Object Model (reported in Figure 5) consists of a set of class hierarchies representing the standardized XML model of MPEG-21. The design has been modeled on the basis of the DIDL hierarchy, as it is the infrastructure on which the other hierarchies lean on (see for example IPMP and DII). The model has been designed to be expandable and flexible thus allowing to cope with standard metadata and to accept the introduction of new MPEG-21 standard parts. In particular, *MPEG21Element* class provides model expandability, since it exposes the basic functions to browse the model and for its manipulation at structural level. Moreover, *MPEG21Element* class presents virtual functions to allow identifying classes on the basis of the namespace and the name of the corresponding XML element. Sub-hierarchies of *MPEG21Element* have been already produced. For example, *DIDLElement* sub-hierarchy to represent DIDL XML elements and *IPMPElement* sub-hierarchy to represent protected elements standardized in IPMP DIDL.

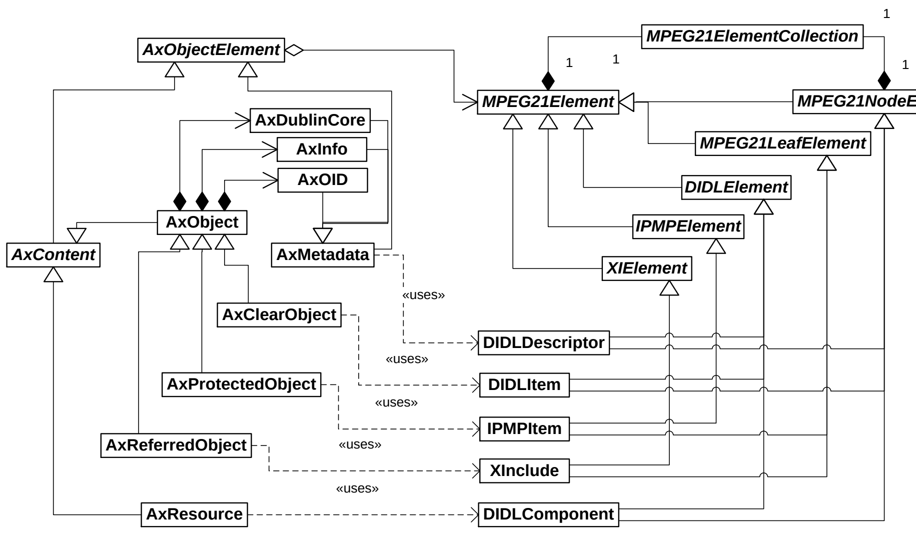

Figure 5 – AXMEDIS and MPEG-21 Object Models relationships

MPEG21ElementCollection has been designed to provide a common mechanism to manage child elements. Since each XML element has structural constraints, the *MPEG21ElementCollection* provides functions to manage children respecting the given constraints. This class is used by *MPEG21Element* to maintain references to its children. The MPEG-21 Object Model provides some listener interface -- i.e., interfaces which has to be implemented by those classes which want to be warned every time something change in the DI. In particular, the following events have been provided: structure event (remove, add, etc.); property event (attribute value changed); and content event (text content changed).

Since AXMEDIS Objects have specific features with respect to a generic DIs, **AXMEDIS Object Model** has been created. The AXMEDIS Data Model design represents only an instance of what can be performed by using the AXOM to realize a customized model to cope with a specific MPEG-21 profile.

AXMEDIS Data Model presents a set of features: (i) realize the PM2 package model; (ii) present a recursive structure, i.e., an AXMEDIS Object could contain others AXMEDIS Object; (iii) may refer to one or more digital resources; (iv) each object has a unique identifier AXOID; (v) each object contains Dublin Core metadata; (vi) each object contains business-level metadata, AXInfo; (vii) each object may contain any MPEG-7 metadata and other descriptors in the spirit of PM2 package model. The AXMEDIS Object Model leads on the MPEG-21 Object Model providing simpler and specific interfaces to manage AXMEDIS Objects. In fact, the AXMEDIS Object Model can also be seen as a “specific view” of a corresponding MPEG-21 Object Model, while structurally remain a full MPEG-21 implementation. While the AXMEDIS Object Model leads on the MPEG-21 one, the latter is independent from the former and could be reused in other applications. However, this choice brings a hard problem which is the synchronization of the AXMEDIS Object Model with respect to the modifications made on the MPEG-21 Object Model. This problem has been solved using an event-driven approach. That is, exploiting the listener interfaces provided by the MPEG-21 Object Model, an AXMEDIS Object is able to coherently modify its structure with respect to the underlying DI. If the DI (e.g., after a modification) does not match the AXMEDIS Data Model, the AXMEDIS Object will try to fit the DI as much as possible discarding those parts which are not complaint to the model.

5. Using AXOM, AXMEDIS Object Manager

As stated, the AXOM can be used to build a large set of content manipulation applications/systems obtaining trusted behavior encapsulated in a clear use of AXMEDIS/MPEG-21 Objects. The developer can use the API exposed by the AXOM for object loading, browsing and navigation in the object structure, obtaining streams for embedded digital resources according to the rights defined in the licenses. Every action performed on the content is related to a set of rights, which have to be owned at consumption time to enable the action. The result is a transparent DRM-including manipulation of the underlying content package. The API is organized in two main subsets: **Object Model Access** – functionalities for browsing and reading content elements; i.e. getting metadata, components, reading content element attributes such as resource descriptors, technical information: MIME-type; **Object Model Manipulation** – functionalities for modifying content package according to DRM rules by using a set of executable commands to add, remove, copy, move content elements such as metadata or resources, to change content elements attribute.

The **Object Model Access** API covers the basic functionalities presented in Section 3.3 to explore and retrieve information from the content package; some of them are listed in the following:

- *AxObjectElement getAxObjectElement(AxIndex index)* – to obtain information about a content element, it provides a clone of the element included in the package;
- *vector AxIndex getAxChildIndexes(AxIndex index)* – to realize *getcontent()* of the PM2 model. It gets the list of indexes of the package component when applied to the root level or to an inner sub-package;

- *vector AxIndex getAxMetadataIndexes(AxIndex index)* – to realize *getmetadata()* of the PM2 model. It gets the list of indexes of the package descriptors;
- *DataSource getResourceAsset(AxIndex index, string right, string details)* – to realize the digital resource extraction, providing a byte stream that can be used for different purposes such as rendering, printing, content processing. This method can used for different purposes. The second and third parameters allow to impose of the action to be performed on the resource in terms of rights and related details —e.g., play for 15 minutes;
- *AxPublicMetdataTree getPublicMetadataTree(AxIndex index)* – to realize *extract-pub-desc()* of PM2 model from a given package (root or sub-tree).

The **Object Model Manipulation** API has been kept minimal; while an extensible set of elementary command classes have been provided. Thus the content manipulation looks like a sequence of commands targeting the AXMEDIS Object.

5.1. AXOM Commands and usage

In the following, a subset of the commands which are at disposal in the AXOM is presented. Each command models a specific manipulation on the content package. The important aspects of each class, inherited by *AxCommand*, are the constructors, which are needed for setting the operation parameters, and additional methods defined in order to get the execution results. The results typically may contain additional information not directly accessible in the modified package.

- *AxCommandAdd*– this command class has been defined to add AxObject elements. It presents constructors to impose the entry point of the addition: *AxCommandAdd(AxObjectElement newElement, AxIndex parentIndex)* to add an element at the end of the list of descriptors or components; *AxCommandAdd(AxObjectElement newElement, AxIndex parentIndex, AxIndex referenceIndex, bool insertBefore)* to insert a new content element before or after a given element in the list *AxIndex getIndexOfAddedElement()* to obtain after the execution the logical reference of an added new element
- *AxCommandDelete* – command class defined to reduce the content package by deleting descriptions, digital resources or inner packages. A constructor is needed in order to select the target element to be removed: *AxCommandDelete(AxIndex deleteIndex)*
- *AxCommandEdit* – command class defined to edit the attributes included in the content elements of the package:. *AxCommandEdit(AxObjectElement dataElement, AxIndex editIndex)* change the attributes of the target element by copying them from *dataElement*;

The usage of AXOM functionalities is quite simple. For the manipulation of content package the creation of a new command and its execution are needed. The AXOM is responsible of providing indexes for the root level or for any sub-tree of the package hierarchy.

```
// creating a manager to manipulate a new AXMEDIS object
AxObjectManager myEmptyObject = new AxObjectManager();
// creating a resource element targeting to a digital resource URL (a
jpeg image)
AxResource myDigitalRes = new AxResource();
myDigitalRes.load("bar.jpg");
// performing the addition of the created resource in the empty object
// step1: creation of the suitable command object
AxCommandAdd addCmd = new AxCommandAdd(myDigitalRes,
myEmptyObject.getRootIndex());
// step2: execution of the command
myEmptyObject.executeCommand(addCmd);
// step3: (optional) gathering of the results: the index of the added
element
AxIndex addedResIndex = addcmd.getIndexOfAddedElement();
```

The usage of the AXOM hides the DRM verification of grants which can be needed to authorize the manipulations. The AXOM is capable of controlling the execution of the above mentioned commands, delegating to them the specification of the grants needed and the accesses to the content elements. On the other hand, only the AXOM can (i) request the grant authorization from the Authorization Service, and can (ii) unprotect the content package.

5.2. Defining custom commands for using AXOM

The proposed solution has been designed to speed-up the creation of AXMEDIS-compliant tools, which can easily exploit the provided functionalities and can also extend the command set to their specific need. A custom new command can be defined by taking into account its fundamental aspects: authorization, un-protection and behavior implementation. By specializing from *AxCommand* class the custom command class presents *getRequiredGrants*, *getAccessedIndexes* and *execute* that have to be implemented according to the desired semantics and manipulation logic. The personalizations have to define one or more constructors to give arguments to the manipulation (e.g., target elements) and to introduce specific methods in order to obtain back information after a performed execution.

An example of an extension set of manipulation commands could be one for resource processing. Let us consider a command for processing an image performing a mirror transformation (left to right) and replace the resource with the result of the processing. Command *AxCommandImageMirror* defines a constructor which accepts the target image as an argument. The constructor can be defined as *AxCommandImageMirror(AxIndex imageIndex)*. The following pseudo-code provides and example about the definition of the command, including the declaration of the required rights and what has to be unprotected. The method *execute*() sketching the command implementation is also reported. Each proposed custom command has to be certified and approved to be compliant with the semantics of the rights. This means that the enforcement of the rights into the authoring and player tools has to be performed in according to a rights data dictionary – e.g., MPEG-21 RDD [MPEG-21 RDD] and that defined by Mi3P [MI3P DICT].

```
class AxCommandImageMirror : public AxCommand
{  private AxIndex targetIndex;
   //constructor
   AxCommandImageMirror(AxIndex imageIndex)
   {     targetIndex = imageIndex;
   }
   //declaration of required granted rights the operation "modify" has to
   // be authorized with details "apply mirror filter"
   vector AxGrant getRequiredGrants()
   {     return new vector { new AxGrant(targetIndex, "modify",
                              "apply mirror filter") };
   }
   //declaration of content elements to be unprotected
   //only the target resource has to be unprotected
   vector AxIndex getAccessedIndexes()
   {     return new vector { targetIndex };
   }
   void execute(AxModelContainer model,
                AxIndexManager indexManager, AxStatusManager s)
   {     //since the model has been already unprotected it is possible to
         //have direct access to the target resource (located by targetIndex)
         AxResource res = (AxResource) indexManager.resolveIndex(targetIndex);
         string mime res.getMIMEType();
         //processing
         inputstream bytes, res.getAsset().getInputStream();
         inputstream mirrorbytes = applyMirrorFilter(mime, bytes);
         res.getAsset().setInputStream(mirrorbytes);
   }
};
```

Please note that *targetIndex* is used to store the location of the target resource, which is set only at construction time. This index is returned as the unique "accessed index" of the grant authorization. The index is used at command execution time to retrieve the resource in the object model.

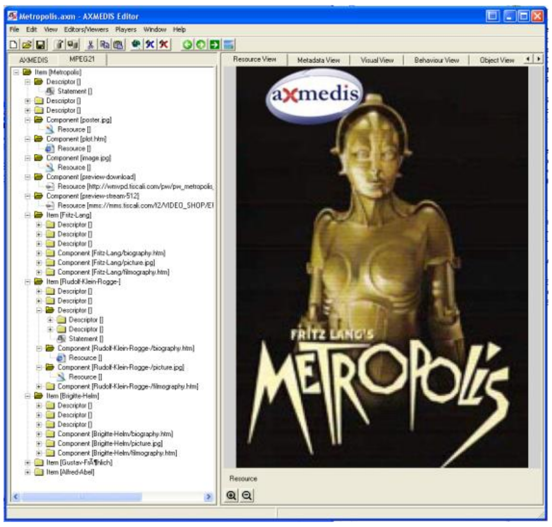
Metropolis.axm - AXMEDIS Editor
File Edit View Editors/Viewers Players Window Help
AXMEDIS
MPEG21
Item [Metropolis]
Descriptor []
Statement []
Descriptor []
Descriptor []
Component [poster.jpg]
Resource []
Component [plot.htm]
Resource []
Component [image.jpg]
Resource []
Component [preview-download]
Resource [http://wmvpd.tiscali.com/pw/pw_metropolis_
Component [preview-stream-512]
Resource [mms://mms.tiscali.com/t2/VIDEO_SHOP/E
Item [Fritz-Lang]
Descriptor []
Descriptor []
Descriptor []
Component [Fritz-Lang/biography.htm]
Component [Fritz-Lang/picture.jpg]
Component [Fritz-Lang/filmography.htm]
Item [Rudolf-Klein-Rogge-]
Descriptor []
Descriptor []
Descriptor []
Descriptor []
Descriptor []
Statement []
Component [Rudolf-Klein-Rogge-/biography.htm]
Resource []
Component [Rudolf-Klein-Rogge-/picture.jpg]
Resource []
Component [Rudolf-Klein-Rogge-/filmography.htm]
Item [Brigitte-Helm]
Descriptor []
Descriptor []
Descriptor []
Component [Brigitte-Helm/biography.htm]
Component [Brigitte-Helm/picture.jpg]
Component [Brigitte-Helm/filmography.htm]
Item [Gustav-FrÃ¶hlich]
Item [Alfred-Abel]
Resource View
Metadata View
Visual View
Behaviour View
Object View
axmedis
FRITZ LANG'S
METROPOLIS
Resource

Figure.6 – AXMEDIS Editor. Please note that the editor is capable of showing the object as an AXMEDIS hierarchy, or an MPEG-21 hierarchy. In this case an MPEG-21 hierarchy is shown.

6. CONCLUSIONS

The multimedia and cross media content formats and related tools have support protecting and manipulating in several manners, in order to allow distribution in which some digital rights management and/or other action control solution-technology are used. These new needs come from the usage of the DRM technologies into the B2B area and also for the fact that the final users are becoming content producers, shortening de-facto the value chain and bringing the needs of having package models capable of supporting nesting levels of content and protection, reuse of non-protected and protected objects for the production/distribution of other content products. This paper presented the evolution of models for content packing with a formalization that allows verifying the model validity against operations performed on the content models. The study has produced the AXMEDIS model and metadata, which have been defined extending MPEG-21. On the basis of the AXMEDIS model presented a software core called AXMEDIS Object Manager, AXOM, has been designed and realized. The AXOM supports high flexibility and permits the production of software application without allowing the violation of rights. The AXOM is currently used in all AXMEDIS content authoring and players tools of AXMEDIS to cope with AXMEDIS and MPEG-21 objects (see Figure 6). The AXOM solution can be used to develop a wide range of tools based on MPEG-21 standard (for PC on Windows and Linux, for PDA and Set Top Boxes). To retrieve additional details on the presented work, please refer to public reports and deliverables of the AXMEDIS.

7. ACKNOWLEDGMENTS

The authors would like to thank all AXMEDIS partners such as TISCALI spa, HP Italy, EUTELSAT (France), University Pompeo Fabra (Spain), SEJER (France), GIUNTI (Italy), FHG-IGD (Germany), University of Leeds (UK), EPFL (Ch), University of Reading (UK), EXITECH, etc., for their contributions, funding and collaborations, and a warm thanks to the AXMEDIS User Group that followed and supported the project evolution with several suggestion and validation. A specific acknowledgment to EC IST FP6 for the partial funding of AXMEDIS project.

8. REFERENCES

- AXMEDIS, “Framework and Tools Specifications”, http://www.axmedis.org
- Bellini, P., Barthelemy, J., Bruno, I., Nesi, P., Spinu, M., “Multimedia Music Sharing among Mediatheques, Archives and Distribution to their attendees”, Journal on Applied Artificial Intelligence, Vol.17, N.8-9, pp.773-796, 2003.
- Bellini, P., Nesi, P., “An Architecture of Automating Production of Cross Media Content for Multi-channel Distribution”, in Proc. of first International Conference on automated production of cross media content for multichannel distribution, AXMEDIS 2005, IEEE Computer Soc. Press., Florence, Italy, November 2005.
- Bellini, P.; Nesi, P.; Rogai, D.; Vallotti, A., “AXMEDIS tool core for MPEG-21 authoring/playing”, Proc. Of the first International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution, IEEE Computer Soc. Press, Nov. 2005, Florence, Italy, AXMEDIS 2005.
- Bulterman D. C. A., Lynda Hardman, “Structured multimedia authoring”, ACM Transactions on Multimedia Computing, Communications, and Applications, TOMCCAP, Vol.1, Issue 1, February 2005.
- Burnett, I.; Van de Walle, R.; Hill, K.; Bormans, J.; Pereira, F., “MPEG-21: goals and achievements”, IEEE Multimedia, Vol.10, Issue 4, pp.60-70, Oct-Dec 2003.

- Burnett, I.S., Davis, S.J., Drury, G. M., "MPEG-21 digital item declaration and Identification-principles and compression", IEEE Transactions on Multimedia, Vol.7, N.3, pp.400-407, 2005.
- Chiariglione, L., MPEG Group, “The MPEG Home Page”, www.chiariglione.org/mpeg.
- Gamma, E., R. Helm, R. Johnson, J.Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
- Hulsen, P.; Kim, J.-G.; Lee, H.-K.; Kang, K.-O., “Delivering T-learning with TV-anytime through packaging”, Proc. of the IEEE International Symposium on Consumer Electronics, pp.614-619, Sept. 1-3, 2004.
- Iannella, R., “Open Digital Rights Language (ODRL)”, Version 1.1 W3C Note, 19 September 2002, http://www.w3.org/TR/odrl .
- Iannella, R., “Standards for Digital Rights Languages”, PlanetEBook, August 24, 2001.
- Koushanfar, F., Inki Hong, Miodrag Potkonjak, “Behavioral synthesis techniques for intellectual property protection”, ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol.10, Issue 3, ACM Press, July 2005.
- Lee, J., Hwang, S.O., Jeong, S.-W., Yoon, K.S., Park, C.S., Ryou, J.-C., “A DRM Framework for Distributing Digital Contents Through the Internet”, ETRI Journal, Vol.25, N.6, pp.423-435, December 2003.
- Lin, E.T., Eskicioglu, A.M., Lagendijk, R.L., Delp, E.J., “Advances in Digital Video Content Protection”, Proceedings of the IEEE, Vol.93, N.1, pp.171-183, January 2005,
- Mi3P, Music Industry Integrated Identifier Project, http://www.mi3p-standard.org/, MI3P-DICT-10-FDS - The MI3P Data Dictionary Standard - Final Draft Standard http://www.mi3p-standard.org/specification/MI3P-DICT-10-FDS.pdf RDD
- Mourad, M., Hnaley, G.L., Sperling, B.B., Gunther, J., “Toward an Electronic Marketplace for Higher Education”, Computer of IEEE, pp.58-67, June 2005.
- MPEG Group MPEG-21 DID, “Introducing MPEG-21 DID”, www.chiariglione.org/mpeg/technologies/mp21-did/
- MPEG Group MPEG-21 IPMP, “Introducing MPEG-21 IPMP Components”, www.chiariglione.org/mpeg/technologies/mp21-ipmp/
- MPEG Group MPEG-21 RDD, “Introducing MPEG-21 RDD”, www.chiariglione.org/mpeg/technologies/mp21-rdd/
- Nesi, P., Rogai, D., Vallotti, A., “A Protection Processor for MPEG-21 Players”, Proc. of the IEEE International Conference on Multimedia and Expo, Toronto, Canada, 9-12 July 2006.
- OMA http://www.openmobilealliance.org/
- Prados, J., Rodriguez, E., Delgado, J., "Interoperability between different rights expression languages and protection mechanisms", in Proc. of the 1st International Conference on Automated Production of Cross Media Content for Multi-Channel Distribution, AXMEDIS 2005, Florence, Italy, 30 Nov.-2 Dec. 2005.
- SCORM: http://www.adlnet.org/
- TvAnyTime: http://www.tv-anytime.org/
- Vetro, A., Timmerer, C., "Digital item adaptation: overview of standardization and research activities", IEEE Transactions on Multimedia, Vol.7, N.3, pp.418-426, 2005.
- Wang, X., “MPEG-21 Rights Expression Language: enabling interoperable digital rights management”, IEEE Multimedia, Vol.10, Issue 4, pp.60-70, Oct-Dec 2003.
- Wang, X., De Martini, T., Wragg, B., Paramasivam M., Barlas C., "The MPEG-21 rights expression language and rights data dictionary", IEEE Transactions on Multimedia, Vol.7, N.3, pp.408-417, 2005.
- WEDELMUSIC: http://www.wedelmusic.org