MODEL AND USAGE OF A CORE MODULE FOR AXMEDIS/MPEG-21 CONTENT MANIPULATION TOOLS

P. Bellini, P. Nesi, L. Ortimini, D. Rogai, A. Vallotti

DSI-DISIT, Department of Systems and Informatics, University of Florence
Via S. Marta, 3 - 50139 Florence, Italy
nesi@dsi.unifi.it, http://www.disit.dsi.unifi.it

ABSTRACT

Despite the relevance of the MPEG-21 standard, little has been done about the modeling of authoring tools and players for the production/consumption of MPEG-21 digital objects. The design and implementation of MPEG-21 tools present several critical points to be solved at the architectural level, so as to ensure security and provide a suitable support and the requested flexibility for manipulating any kind of digital resource, according to the spirit of MPEG-21 standard. This paper presents the AXMEDIS core model and components for MPEG-21 content authoring tools and players. The proposed architecture provides both data manipulation flexibility and a high level of security when digital resources are used. The same architecture can be used to develop any other MPEG-21 DI tools. The architecture presented has been adopted to build authoring and player tools developed for the AXMEDIS IST FP6 R&D European Commission project (http://www.axmedis.org).

1. INTRODUCTION

The state of the art of multimedia content modeling, packaging, protection and distribution is based on file/stream formats. At present, there is a large number of content formats ranging from basic digital resources (documents, video, images, audio, multimedia, etc.) to integrated content packages such as MPEG-21 [1] and WEDELMUSIC [2], TV-AnyTime, etc. These integrated models wrap any kind of digital resource in a container with the related information (e.g., content metadata), thus making them ready for delivery (streaming or download), even in some protected form. Those solutions are enabling a large range of business and transaction models, supporting them with DRM (Digital Rights Management).

The typical scenario which synthesizes the packaged content lifecycle is shown in Fig.1. The content author has to embed raw digital resources in order to produce a package for the distribution (e.g., WEDELMUSIC, OpenSky package, MPEG-21, photo collection of cameras, etc.). He can decide to protect the produced package and to grant the access to some consumers with a specific license. Once the content has reached the consumer, the player gets access to the license and if authorized, some keys are received to unpack the content and render it according to what is allowed by the license.

The following issues arise in the above scenario:

- modeling and defining the content package, which has to be known from both authoring and player tools, not restricted to any specific content and metadata formats;
- handling the content package, manipulating the structure (i.e., content organization) in an easy way and the digital resources together with their metadata for both rendering/modifying (authoring, editing);
- accessing to content under DRM rules, preventing from unauthorized manipulation of packaged content, while providing transparent means to get the allowed resources;

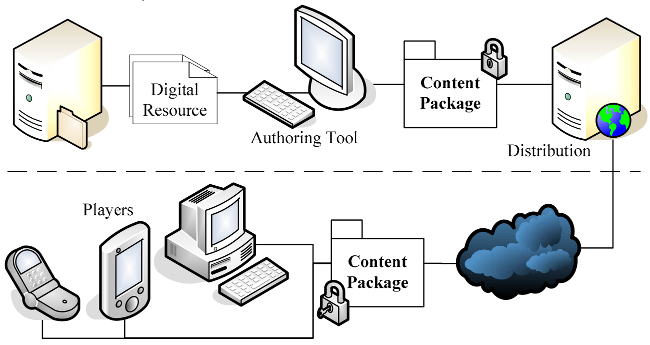

Figure 1 – Scenario on content package and its usage

Other software engineering requirements have to be considered in order to implement reusable components to be used in different tools. The produced tools have to be easily extendable in order to accept quickly new types/models of content and/or metadata. The player can be hosted by different HW/SW platforms, therefore portability of content manipulation software is a key issue. Since a subset of functionalities are needed in both authoring and playing tools, a common core module should be designed addressing package model, manipulation and access control.

MPEG-21 Digital Item [3], DI, has been selected as the underlying packaging model; indeed MPEG-21 provides a unique multimedia framework to address packaging, distribution, adaptation, protection and licensing.

The implementation of a MPEG-21 core module for authoring/playing is a very complex task since it has several critical points. In fact, this module has to assure at the same time (i) easy and fast access at the modeled information and resources; (ii) high security level enforcing the DRM on the digital resources usage. These requirements lead the design towards opposite directions (accessibility vs. security), therefore it is difficult to solve easily both aspects within a unique solution.

This paper reports the architecture of a core module to develop MPEG-21 compliant tools which cover the above mentioned problems. The latter are not solved at MPEG standard level, since such standard only specifies the formats for packages and metadata, but it defines neither tool design nor system architecture for distribution, production and consumption of digital goods. The proposed architecture is currently used for implementing the AXMEDIS tools.

In the AXMEDIS project, an additional abstraction has been conceived to simplify for the author content packaging, since MPEG-21 has a very flexible structure providing a high granularity of content organization. AXMEDIS Data Model is also presented. This model is compliant with MPEG-21 and satisfies the requirements gathered by the partners of the AXMEDIS Consortium. AXMEDIS defines a specific structure for the DIs, i.e., how digital resources should be nested, etc. It also defines relevant metadata which have to be included in a DI, without restricting the usage of any additional metadata. An MPEG-21 DI which fulfils the AXMEDIS Data Model is called AXMEDIS Object.

The paper is organized as follows. Section 2 gives a short overview of MPEG-21. An overview of the AXMEDIS Object Manager (AXOM) is reported in section 3. Conclusions are drawn in Section 4.

2. MPEG-21 FEATURES

MPEG-21 is mainly focused on the standardization of the DRM aspects and packaging. In particular, MPEG-21 scope is mainly related with the content and metadata formats leaving completely undefined system architectures, business models, etc. The standardization process of MPEG-21 is still under completion; at present some parts are mature, whereas others are under evolution. Two parts of the standard are the most relevant to this work: “Digital Item Declaration” (DID) and “Intellectual Property Management and Protection” (IPMP). As to the capability of the authorizing access to content, other parts of the standard have to be considered: Rights Expression Language (REL) and Right Data Dictionary (RDD).

The DID defines how DIs have to be represented. A DI is a structured digital object and it is the fundamental unit of distribution and transaction within the MPEG-21 framework. A DI is a package for digital resources and related metadata. A DI 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. In particular, some metadata of general purpose (such as identification, rights description, format description, etc.) have been considered relevant and included in the standard.

IPMP provides the means to include protected content inside a DI. Every DIDL element (i.e., XML element) can be replaced by a protected version: an IPMP element. The latter contains the original DIDL element in a protected form together with the required information to perform un-protection (i.e., applied protection tools, etc.) [4].

Some other parts of the standard define XML schemas to represent important metadata. They are not relevant to the work presented in this paper, and yet they have been taken into account, for the sake of completeness, during core module design in order to guarantee extensibility for a later inclusion of such parts.

3. AXMEDIS OBJECT MANAGER

The core module, which addresses MPEG-21 and other additional requirements, as already stated, has been developed and is called AXMEDIS Object Manager (AXOM) [5]. The AXOM is currently used to develop all the tools of the AXMEDIS architecture which handle DIs and, in particular, with AXMEDIS Objects. Moreover, the AXOM could be used to develop any other application which manages MPEG-21 DIs.

In order to build a reusable technology, the responsibility separation is a mandatory requirement. For such reasons, the AXOM has been decomposed in five parts as shown in Fig.2. They are better analyzed as follows.

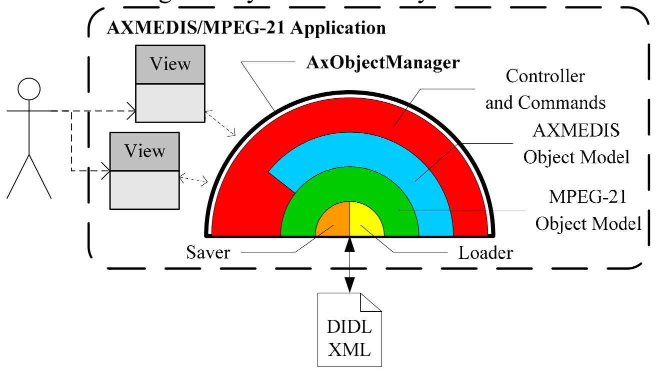

Figure 2 – AXOM layering

One of the most import tasks of the AXOM is to provide the means to create a trusted environment. That is, it has to guarantee that the system is controllable w.r.t. the user actions on any AXMEDIS Object. Different actions on the model should require different grants (i.e., authorizations). Actions can target the content structure, the resources and the metadata. Therefore a unique flow has been conceived in order to handle verification of any action behavior performed on the content.

At the outer level a **controller** class called *AxObjectManager* has been realized and the Command pattern [6] has been applied to its design. It is an intermediate layer between the application views and the object models (i.e., the working document). An application view does not manipulate the object model (AXMEDIS or MPEG-21), while it issues *commands* to the controller class and the latter is in charge of performing the requested actions on the model. An extensible architecture for manipulating AXMEDIS objects or MPEG-21 DIs (from now on simply referred as "content") under DRM enforcement has been realized, allowing the creation of new commands. The controller exposes governed 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.

Each conceivable command has been realized as a class which models execution behavior and required grants to allow such execution. In that way, the controller is able to handle generically the request received by the user respecting governance. Please note that the controller is not in charge of verifying the grants, since it has been designed to provide hooks, thus enforcing easily control mechanisms.

By using model encapsulation, avoiding direct manipulation from the view, a good level of security has been achieved in terms of robustness against developer's malicious content handling. This feature cannot be guaranteed if the view obtains at any time references to content elements in the model. For such reasons, views are not allowed to target content elements by using pointers. Command execution may return content elements information, providing a clone. This avoid accessing via pointer at the memory of the whole object model. When digital resources have to be accessed (e.g., for their rendering) the chain of unprotection tool is activated, thus allowing to establish a direct stream from the encapsulated resource and the application view. Command targets have to be indicated using logical references. In that way, common cracking activities like memory dump are not useful to access content data without a proper authorization.

Since the AXOM manipulates MPEG-21 DIs, the **MPEG-21 Object Model** has been developed (see Fig.3). This model consists of a set of class hierarchies, which represent the standardized XML elements [3]. The design has been focused on the DIDL hierarchy, as the latter is the infrastructure the other hierarchies lean upon. Nevertheless, the model has been designed to be expandable and flexible, thus allowing to cope with standard metadata.

The main class is *MPEG21Element* and provides model expandability. It exposes the basic functions to browse the model and to manipulate it at a structural level. Moreover, *MPEG21Element* declares virtual functions which allow identifying classes on the basis of the namespace and the name of the corresponding XML element. Some sub-hierarchies of *MPEG21Element* have been already produced. For example the one which origins from *DIDLElement*, to represent DIDL XML elements and the one which origins from *IPMPElement* to represent protected elements standardized in IPMP.

MPEG21ElementCollection has been designed to provide a common mechanism to manage child elements. Since each XML element has structural constraints (as specified in the related XML schema, e.g., of ordering), the *MPEG21ElementCollection* provides functions to manage children, while respecting the given constraints. This class is used by MPEG21Element to keep references to its children.

The MPEG-21 Object Model provides some listener interface, i.e., interfaces which have to be implemented by those classes requesting to be warned every time something has changed 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)

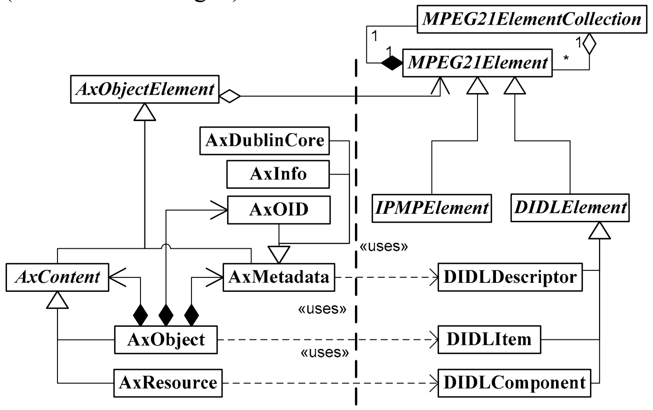

Figure 3 – part of AXMEDIS/MPEG-21 object models

Since AXMEDIS Objects have specific features w.r.t. a generic DIs, **AXMEDIS Object Model** has been created. AXMEDIS Data Model main features are as follows: it has a recursive structure, i.e., an AXMEDIS Object could contain others AXMEDIS Objects; it could embed or refer to one or more digital resources; each object has a unique identifier called AXOID; each object contains Dublin Core metadata; each object contains business-level metadata called AXMEDIS Info; any MPEG-7 metadata; etc.

The AXMEDIS Object Model is based 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 "simplified view" of the MPEG-21 Object Model, while it remains structurally a full MPEG-21 implementation. The AXMEDIS Object Model is composed by the following classes:

- *AxMetadata* represents generic metadata. It mainly provides methods to get access to the related XML. *AxInfo*, *AxDublinCore* and *AXOID* are subclasses of it and they represent those metadata which are mandatory in AXMEDIS. Each class provides specific methods to manage the related metadata;
- *AxResource* represents a digital resource;
- *AxObject* class represents an AXMEDIS Object therefore it could contain other *AxObject*, *AxResource*

and *AxMetadata*, it has to contain an *AxInfo*, an *AxDublinCore* and an *AXOID*.

All these classes are in charge of synchronizing the underlying MPEG-21 Object Model every time any action is made on the AXMEDIS Object Model.

While the AXMEDIS Object Model is based on the MPEG-21 model, the latter is independent from the former and it could be reused in other applications. However, this choice brings forth a hard problem which is the synchronization of the AXMEDIS Object Model w.r.t. modification made on the MPEG-21 Object Model. This problem has been solved according to an event-driven approach. That is, by exploiting the listener interfaces provided by the MPEG-21 Object Model, an AXMEDIS Object is able to modify coherently its structure w.r.t. 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.

The **MPEG-21 Loader** is in charge of reading the XML document representing a DI and building the corresponding object model. Since the MPEG-21 Object Model is extendable, the MPEG-21 Loader has to be extendable as well. Moreover, since the same object model should be used on several types of device (from a personal computer to a pocket pc, etc.), the SAX2 interface as defined by the W3C has been selected for loading. In fact, this approach allows exploiting a larger set of existing XML parsers (also for resource constrained devices), and it ensures a less time and memory consuming process than the DOM approach. SAX2 parsing procedure produces a straightforward sequence of events, that notifies the occurrence of XML entities (i.e. elements, characters, etc.), which has to be managed by a unique handler. When loading a DI containing an extendable and heterogeneous set of elements, a general handler cannot cope with all the situations that can arise. In fact, the insertion of a single MPEG-21 element in the object model could end up handling more than one event. To solve this problem, the object loading has been realized thanks to the collaboration of different loader classes: a loader, that finds an MPEG-21 element having a loading model which is not directly manageable, can delegate a suitable loader to handle it. An abstract base class for loading, which realizes the underlying collaboration mechanisms, has been developed. Concrete classes have been implemented, achieving different loading behaviors. The loading process leans on a factory mechanism. Abstract Factory pattern [6] has been used in conjunction with Factory Method pattern [6] in order to create elements given a namespace and an element name. In particular elements of MPEG-21 hierarchies, such as DIDL, IPMP DIDL, can be created.

The **MPEG-21 Saver** has the responsibility of writing a correct XML document on the basis of the current object model. As to the MPEG-21 Loader, it has to be extendable. Hence, a hierarchical approach, like that of factories, has been adopted. Specific classes (writers) are in charge of knowing how all the elements of the related namespace should be represented in XML. The stream concept has been exploited to model the destination of the writing process, allowing producing DIs to files, strings, network messages, etc. The writing process is generically managed by orchestrating all the namespace-dependant writers. It is worthwhile pointing out that the writers do not have to write elements directly in XML syntax. In order to prevent them from such triviality, *XMLWriter* class has been developed. This class exposes high level methods which allow representing elements in XML syntax. In that way, the real XML writing is hidden and transparent for the element writers.

4. CONCLUSIONS

A flexible and extendable model to manage MPEG-21 DIs, while satisfying all the requirements of AXMEDIS tools have been designed and implemented in C++ for providing several OS portability (also considering differences among compilers). It includes classes and infrastructure to load and save DIs. This solution can be extended according to the model flexibility and at the same time it assures the needed protection level. An additional view of the DIs has been provided considering relevant business metadata and a simplified content structure. This has allowed a simpler interaction with the object model. Finally, the AXMEDIS Object Manager architecture has been realized allowing embedding control mechanisms for DRM enforcement. To retrieve additional details on the presented work, please refer to public reports and deliverables of the AXMEDIS project [7]. The validation process has consisted in exploiting the produced core module in AXMEDIS Editor (authoring tool) and in the AXMEDIS Players.

5. ACKNOWLEDGMENTS

The authors would like to thank all the AXMEDIS project partners for their contribution and collaboration. A specific acknowledgment to the EC IST FP6 for partially funding the AXMEDIS project.

6. REFERENCES

[1] L. Chiariglione, MPEG Group, "The MPEG Home Page", www.chiariglione.org/mpeg.

[2] P. Bellini, J. Barthelemy, I. Bruno, P. Nesi, M. Spinu, "Multimedia Music Sharing among Mediateques, Archives and Distribution to their attendees", *Journal on Applied Artificial Intelligence*,Vol.17, N.8-9, pp.773-796, 2003.

[3] MPEG Group, "Introducing MPEG-21 DID", www.chiariglione.org/mpeg/technologies/mp21-did/

[4] MPEG Group, "Introducing MPEG-21 IPMP Components", www.chiariglione.org/mpeg/technologies/mp21-ipmp/

[5] AXMEDIS DE2.1.1A "User Requirements", www.axmedis.org/documenti/view_documenti.php?doc_id=1062

[6] E. Gamma, R. Helm, R. Johnson, J.Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[7] AXMEDIS DE3.1.2A "Framework and Tools Specifications", www.axmedis.org/documenti/view_documenti.php?doc_id=1379