Automating Production of Cross Media Content for Multi-channel Distribution
www.AXMEDIS.org

DE4.1.1.3 Content Modeling and Managing, Update of DE4.1.1.2

Version: 1.1
Date: 03/10/2007
Responsible: DSI (revised and closed by coordinator)

Project Number: IST-2-511299
Project Title: AXMEDIS
Deliverable Type: Report and Prototype
Visible to User Groups: Yes
Visible to Affiliated: Yes
Visible to Public: Yes

Deliverable Number: DE4.1.1.3
Contractual Date of Delivery: M36
Actual Date of Delivery: 03/10/2007
Work-Package contributing to the Deliverable: WP4
Task contributing to the Deliverable: T4.1.1, T4.1.2, T4.1.3, T4.1.4
Nature of the Deliverable: Report and Prototype
Author(s): EPFL, DSI, UNIVLEEDS, EXITECH, SEJER

Abstract
This Deliverable reports the AXMEDIS approach to content management in terms of technology to be adopted for the modeling, editing, viewing and authoring of objects content. The state of the art of existent technology is reported and the existent problems are highlighted. The work performed to extend the existent toolsets to meet the AXMEDIS requirements is described and a plan of foreseen enhancements is provided.

Keyword List:
Data Model, Editors, Authoring Tools, Viewers, MPEG-21 Terminal.

Table of Contents

1 EXECUTIVE SUMMARY AND REPORT SCOPE 6

2 INTRODUCTION 7

3 AXMEDIS OBJECT MANAGER AND PROTECTION PROCESSOR (AXOM) (DSI) 10

3.1 AXMEDIS OBJECT MANAGER: WORK PERFORMED 15
3.1.1 AXMEDIS Package Model and the MPEG-21 standard 15
3.1.2 B2B Needs and AXMEDIS Information 16
3.1.3 Basic Model for Composition of protected and unprotected objects 17
3.1.4 Extended Model for Composition of protected and unprotected objects 19
3.1.5 Considerations on PM2 model 21
3.1.6 AXMEDIS Object Manager 22
3.1.7 MPEG-21 and AXMEDIS Data Models 26
3.1.8 MPEG-21 Load and Save 28
3.1.9 Using AXOM, AXMEDIS Object Manager 28
3.1.10 AXOM Commands and usage 29
3.1.11 Defining custom commands for using AXOM 31
3.1.12 Using AXOM for a player 33
3.2 AXMEDIS OBJECT MANAGER: WORK PERFORMED SUMMARY 35
3.3 AXMEDIS OBJECT MANAGER: PROBLEMS AND ACTIONS 35

4 AXMEDIS OBJECT PREPROCESSOR AND POSTPROCESSOR (EPFL, DSI) 36

4.1 OBJECT PREPROCESSOR AND POSTPROCESSOR: STATE OF THE ART 36
4.2 OBJECT PREPROCESSOR AND POSTPROCESSOR: THE PROBLEMS 37
4.3 OBJECT PREPROCESSOR AND POSTPROCESSOR: WORK PERFORMED 38
4.4 OBJECT PREPROCESSOR AND POSTPROCESSOR: PROBLEMS AND ACTIONS 41

5 AXMEDIS EDITOR, AUTHORING TOOL (DSI) 41

5.1 EDITOR: WORK PERFORMED 42

6 HIERARCHY EDITOR AND VIEWER (DSI) 44

6.1 HIERARCHY EDITOR AND VIEWER: WORK PERFORMED 44

7 VISUAL AND BEHAVIORAL EDITOR AND VIEWER (EPFL) 45

7.1 VISUAL AND BEHAVIORAL EDITOR AND VIEWER: STATE OF THE ART 45
7.1.1 GRiNS Pro Editor for SMIL 2.0 46
7.1.2 LimSee2 47
7.2 VISUAL AND BEHAVIORAL EDITOR AND VIEWER: WORK PERFORMED (EPFL) 48
7.2.1 Module design for SMIL structure 48
7.2.2 Module design for SMIL editor 48
7.2.4 VISUAL VIEW PART 52
7.2.5 BEHAVIOUR VIEW PART 54
module design: 55

8 METADATA EDITOR AND VIEWER (UNIVLEEDS) 56

8.1 METADATA EDITOR AND VIEWER: STATE OF THE ART 56
8.2 METADATA EDITOR AND VIEWER: WORK PERFORMED 56
8.3 METADATA EDITOR AND VIEWER: PROBLEMS AND ACTIONS 60

9 AXMEDIS ACTIVEX CONTROL AS EDITOR (DSI) 60

9.1 ACTIVEX CONTROL: WORK PERFORMED 60
9.2 ACTIVEX CONTROL: PROBLEMS AND ACTIONS 61

10 AXMEDIS PLUG-IN INTO MOZILLA (SEJER) 62

10.1 PLUG-IN INTO MOZILLA: WORK PERFORMED 63

10.2 PLUG-IN INTO MOZILLA: PROBLEMS AND ACTIONS 64

11 AXMEDIS PLUG-IN INTO MULTIMEDIA PLAYERS (DSI) 65

11.1 PLUG-IN INTO MULTIMEDIA PLAYERS: WORK PERFORMED 65
11.2 PLUG-IN INTO MULTIMEDIA PLAYERS: PROBLEMS AND ACTIONS 65

12 AXMEDIS PC PLAYER STATE OF THE ART (DSI, EPFL) 65

12.1 PC PLAYER: STATE OF THE ART 65
12.1.1 Flash 66
12.1.2 Windows Media 67
12.1.3 QuickTime 68
12.1.4 Real 69
12.1.5 iTunes 70
12.1.6 Winamp 71

13 AXMEDIS PC PLAYER: WORK PERFORMED (DSI) 73

14 AXMEDIS PC PLAYER ACTIVEX BASED (DSI) 74

15 SMIL PLAYER IN AXMEDIS PLAYER (EPFL) 78

15.1 SMIL PLAYER IN AXMEDIS PLAYER: WORK PERFORMED (EPFL) 78
15.2 SMIL PLAYER IN AXMEDIS PLAYER: PROBLEMS AND ACTIONS (EPFL) 81

16 MPEG-4 OSMO PLAYER IN AXMEDIS PLAYER (EPFL) 81

16.1 MPEG-4 OSMO PLAYER IN AXMEDIS PLAYER: WORK PERFORMED (EPFL) 82
16.2 MPEG-4 OSMO PLAYER IN AXMEDIS PLAYER: PROBLEMS AND ACTIONS (EPFL) 84

17 COMMENTS ON THE FULL ROUND OF AXMEDIS PROCESS 85

18 AXMEDIS PDA PLAYER (TISCALI, DSI) 86

18.1 AXMEDIS PDA PLAYER: WORK PERFORMED 86
18.2 AXMEDIS PDA PLAYER: PROBLEMS AND ACTIONS 87

19 AXMEDIS MOBILE PLAYER (DSI) 87

19.1 AXMEDIS MOBILE PLAYER: WORK PERFORMED 87
19.2 AXMEDIS MOBILE PLAYER: PROBLEMS AND ACTIONS 88

20 BIBLIOGRAPHY 88

21 OTHER BIBLIOGRAPHY 89

Please note that:

- You can become affiliated with AXMEDIS. This will give you the access to a huge amount of knowledge, information and source code related to the AXMEDIS Framework. If you are interested please contact P. Nesi at nesi@dsi.unifi.it. Once affiliated with AXMEDIS you will have the possibility of using the AXMEDIS specification and technology for your business.
- You can contribute to the improvement of AXMEDIS documents and specification by sending the contribution to P. Nesi at nesi@dsi.unifi.it
- You can attend AXMEDIS meetings that are open to public, for additional information see WWW.AXMEDIS.org or contact P. Nesi at nesi@dsi.unifi.it

1 Executive Summary and Report Scope

The AXMEDIS consortium (producers, aggregators, distributors and researchers) aims at creating an efficient and complete framework to speed up and optimize content production and distribution with innovative methods and tools compliant with international standards - such as MPEG-21 - and de-facto standards. In the course of realization of such a platform for *production-on-demand*, content modeling and managing play a key-role since the heterogeneity of content formats and representations may create serious interoperability issues if not treated appropriately. In this sense, the AXMEDIS approach is based on the MPEG-21 concept of Digital Item which is the fundamental unit of distribution and transaction manipulated by the users of the framework [1].
Several are the technologies and the approaches currently adopted by the AXMEDIS partners according to their specific business sectors. This document provides then the state of the art of Content Managements Systems (CMS) which can range from simple software solutions to very complex architecture comprising multiple applications.
Three are the main elements of a CMS:

- Data model.

  In order to implement interoperable content manipulation, standard interfaces for content descriptions must be specified. In AXMEDIS such interfaces will be implemented using the XML language in accordance with the major trend of multimedia standard bodies, open initiatives and the industry. XML will be adopted as AXMEDIS file format and tools able to open, manipulate, save and possibly adapt such format will be implemented.
- Authoring toolset.

  Several are the tools necessary to enable content manipulation from its creation to the final distribution passing through several steps of editing/modification. These tools include:
  - *Viewers* able to open content objects and display the carried resources is terms of hierarchy of elements and sub-elements, expected behavior in time once decoded and visual organization in virtual bi and tri-dimensional spaces.
  - *Editors* enabling content creation and modification in user-friendly graphical environment. These tools would provide a powerful environment to potential content creators without the need to directly edit XML files. Moreover, several multimedia objects may need to be organized in complex “scenes” where the relationships among them are coded in a standard format. Being SMIL (Synchronized Multimedia Integration Language) a very popular and widely adopted format, appropriate SMIL editors and decoders will be implemented.
  - *Content (pre/post)processors*. An optimal content distribution may require the binarization of multimedia content in order to fully exploit the available resources in terms of bandwidth and storage space. This implies the need of pre-processors able to convert binary XML to textual format (and possibly resolve external references) and post-processors producing binary content ready to be streamed or downloaded.
- Decoding device (the player).

  In the MPEG-21 vision the end user may become content producer and distributor himself. The specification of the client terminal architecture and functionality is then part of the CMS specification. Differently from MPEG-2 and MPEG-4, in MPEG-21 a terminal architecture cannot be defined as an aggregation of functional blocks (decoders, buffers, demultiplexers, etc.) whose behaviors are driven by specified rules coded as standard descriptors. An MPEG-21 compliant terminal is rather a device able to correctly manipulate Digital Items by invoking a series of tools devoted to specific actions to be performed. These may include: content governance (DRM), adaptation and processing. The specification and implementation of an MPEG-21 compliant device requires then the specification and implementation of: (i) interfaces among the involved MPEG-21 components, (ii) mechanisms enabling the invocation of the required standard or proprietary decoders once the carried resources need to be accessed by the end user.

2 Introduction

WP 4.1 addresses the problem of content modeling using an MPEG-21 approach, providing the suitable research, development and then overall enabling technology to demonstrate this approach. Compatibility with existing CMS tools of AXMEDIS partners will be considered. This introductory section recollects the specification of research for WP4.1.

WP4.1 – Content Modelling and managing -- DSI responsibility –

T4.1.1 CMS Analysis for AXMEDIS Model definition and Content structure Mapping DSI responsible (planned closure for M36)
An integration map will be produced highlighting CMS information, metadata, content type and DRM, presently in the first period a metadata mapper has been planned to be integrated into the AXCP and into the AXDB loader so as to process the content and metadata in automatic manner in the AXCP and AXEPTool areas. This allows using different sets of metadata in different AXMEDIS objects and factories. The aspects that will be taken into account include (but not limited to) the following issues: Content Formats, Content production Workflow, DRM models and interaction, Database structure and interfacing tools, Ingestion mechanisms, Compositional tools for combining content components, Communication capabilities, formatting tools with genetic algorithms, administrative information availability such as licensing, monitoring and versioning capabilities, automatic production capabilities, metadata added to content, database model.

- M30: maintenance of the metadata model
- M36: finalization of the metadata model

T4.1.2 AXMEDIS data model -- DSI responsible (planned closure for M36)
The improvement of AXMEDIS Content Model as derived and improved with respect to MPEG-21 content approach will be performed. In the first period a quite strong model and software have been developed and called AXOM. The aim of this WP for this period is to find a component model that is general enough to include more models, so that it can be profitably used for content production and distribution on multi-channel, and that may create interoperability among the models. The open problems are presently related to the protection aspects, to the protection processor, to the integration with the authoring tools (preserving protection), to the integration with the content processing tools again preserving protection (including external editors, and GRID developed in WP4.3). With ILABS: the integration of IMS and SCORM into the AXMEDIS model in terms of model and metadata.

- M30: updated version of the AXOM, refactory of the AXOM to be compiled on mobiles, reduction of the foot print.
- M36: completed manual for its usage including performance analysis

T4.1.3 Design and development of an authoring tool for AXMEDIS content -- DSI responsible (planned closure for M36)
The AXMEDIS authoring tool needs to be improved adding additional editors for manipulating extended aspects such as metadata, behaviour and visual with SMIL, formatting issues such as style, DRM license modelling, protection information, etc. The usability of the authoring tools will be assessed. Robustness to the attacks will be also assessed. Another aspect to be studied is the integration of the AXMEDIS authoring tool with content processing tools of third parties. This task includes the development of AXMEDIS plug-ins for other editors/players, and an additional study of the availability, integration, and security level of other editing and playing tools such as Ghost Script, GIMP, Mozilla, Windows Media, Acrobat Reader, Photoshop, Macromedia tools, Nero tool kit, etc. With ILABS: the integration of IMS and SCORM into the AXMEDIS model in terms of editor for SCORM and IMS information, integration of AXMEDIS plug into

an ILABS authoring tool and players, including DRM and metadata aspects. Integration of AXMEDIS players into the Authoring Environment.

- M30: an improved version of the AXMEDIS Editor, removing the problems in the protection/unprotection process, integration with OSMO and SMIL players. Integration of verification tools for licenses, protection information editor, visual and behavioral editor and viewer based on SMIL.

EPFL: Considering the Specification and the use cases in which the requested features are well described and not implemented into the present version of the Visual and Behavioural editor, EPFL plans to provide the tools with a precise support of the following main components: SMIL Data Model, Tree View for SMIL, Visual View, Behaviour View, Link from one SMIL button to other SMIL scenarios, Integration with AXOM. The plan of the work goes as follows:

- M25: Implementation of the basic classes of the SMIL data model compliant with the UML schema agreed by EPFL and DSI, allowing the usage of all the views described below.
- M25: Integration of the SMIL data model into the AXMEDIS Editor to allow save and load with respect to the disk.
- M25: Save and load of the SMIL from the AXOM.

SMIL Tree View and Visual View (EPFL):

- M26: Visualisation of the SMIL model as a tree view: editing of the attributes of the tree view structure and elements: allowing adding and deleting new elements, containers.
- M26: Visualisation of the SMIL model as a visual view with the visual editor. Organisation of the tree view and visual view in resizable windows independent in the AXMEDIS editor.
- M26: The visual editor will be available with the following features:
  - Usage and update of the information coming form the model
  - Definition and change of the background size and info
  - Usage of the AXMEDIS configuration manager to save background info
  - Positioning, moving, and resizing rectangles via mouse
  - Definition of the association with resources and rectangles
  - Visualisation of the resource names on the rectangles
  - Scroll and zoom of the visual area
  - Association of rectangles to other SMIL files, not only to resources to be played

SMIL Behaviour View (EPFL):

- M28: Visualisation of the SMIL data model as a behavioural view with the behaviour editor. Organisation of the behavioural, tree and visual view in resizable windows independent in the AXMEDIS editor.
- M28: The behavioural editor will be available with the following features
  - Usage and update of the information coming form the model
  - Usage of the AXOM manager to save zoom and resolution time information
  - Moving and resizing of time scheduled segment for the resources via mouse
  - Visualisation of the resource names on the corresponding time line
  - Scroll and zoom of the time schedule area.
  - Time schedule are with reference grid, relative and absolute (pending)
  - Management of the container and visualisation of them as boxes which hide the collapsed time lines of the contained resources
  - Definition of the expiration time for the scenarios and when has to be done.

- M36: an improved version of the AXMEDIS Editor, improvement of the protection/unprotection process and of the integration with OSMO and SMIL players, integration of other players for documents, etc.

Addition of the Functional editor based on AXCP Editor for the DIP/DIM formalization. Improvements of the content production for PDA and mobiles, including HTML files and SCORM.

- M36: final version of the AXMEDIS authoring tool, performance analysis.

T4.1.4 Implementation of an MPEG-21 compliant client terminal interface – DSI responsible. Collaboration with DSI, SEJER, EPFL, TISCALI, UPC (planned closure for M36).

The client terminal (player) functionality may be seen as a subset of the Editor functionality; this subset only allows playback according to DRM AXMEDIS rules and interaction according to the same rules and to the rules embedded in the multimedia composite scenes. The capabilities of the player also depend on the platform since different platform may impose limitation on the available resources (disk space, memory, operating system capabilities, multithreading, libraries, communication, etc.) The terminal/player may nevertheless allow, for his limited capabilities (in comparison to the Editor), a simpler porting to other platforms like, e.g., Linux OS. At the same time, it supports additional media formats for which playback-only code or libraries are available.

Work is essentially based on the use of SMIL as multimedia description language supporting media spatial-temporal synchronization. Starting from an available SMIL player, initial work has been performed to adapt it to libraries and APIs used in AXMEDIS. The activity in this task includes: integrate the player in the editors to allow content authors to preview their content; integrate the AMBULANT SMIL player with the several AXMEDIS internal players, in this way AMBULANT will take care of the synchronization of the several invoked players, recalling resources located into the AXOM; implement the appropriate interfaces with the AXOM, the Configuration Manager, the Error Manager; the metadata and hierarchy viewers, the internal resource viewers; implement functionality of player customization such as changing colours-set and skins; porting of the player to MS Pocket PC on PDA; implement a module in charge of communicating item adaptation requests to the media provider, whenever this may be necessary for terminal limitations or overloading conditions; support of content protection by means of the use of the Protection Processor and the Protection Manager Support Client; selection and support of a part of the MPEG-21 DI Processing functionality for the provision of interoperability at the processing level by means of DI Methods implementing basic operations (e.g., the user will be able to see a list of available operations at the level of content access and processing such as play, pause, stop, rewind, etc.); integration of OSMO MPEG-4 Player and other players into the AXMEDIS terminal; test Player and terminal implementation, final version of AXMEDIS Player for PDA, final version of the plug-ins for other players and viewers, implementation of additional test terminal interfaces, validation experiments for different content distribution test cases.

Selection and support of MPEG-21 DI Processing (DIP) for the provision of the needed functionality at the processing level by means of DI Methods (DIM) that use DI Basic Operations (DIBOs), already defined by MPEG-21 is foressen. Some examples of operations are play, pause, stop, etc. These methods and operations should be part of the AXMEDIS editor and player. In the editor, to allow its insertion into the AXMEDIS object and in the player, in order to be able to invoke the corresponding method present into the AXMEDIS object (for instance, play). The insertion of the DIM into the AXMEDIS objects will be based on the standardised relationship between Digital Items and Digital Item Processing

- M30: improved version of the AXMEDIS players including: SMIL playing, MPEG-4 playing for both PC and PDA (also including Active X and Mozilla plug ins). Full support of DRM on PC players, support of DRM on PDA player without local cache. The integrated SMIL player will allow to specify paths for using the resources in the AXMEDIS object hiding the complexity of objects to the users. The same activity can be performed by using HTML pages or XML. The advantage of SMIL is the possibility of synchronizing different resources. Definition of profiles for the single players. SMIL and OSMO will be integrated using resources in AXOM via stream. Revision and completion of specification, design, use and test cases for the MPEG-21 Digital Item Processing activity.
- M36: AXMEDIS player for PC with skin and nice user interface, inclusion of DIP/DIM functionalities in major PC players. Integration of the different internal players of AXMEDIS: video, audio, document, html, mpeg-4, SMIL, XML, etc. Enforcement into some of the PC players some capabilities of content adaptation. Integration of PDF player for DRM control.

- M36: Final version of the AXMEDIS players (PC, Active X, Mozilla plug-in, PDA).

3 AXMEDIS Object Manager and Protection Processor (AXOM) (DSI)

In this prototype the main features regarding the management of AXMEDIS objects are included. The demonstrator gives evidence of the production of the following sub-components:

- AXMEDIS Data Model Schema definition
- AXMEDIS/MPEG-21 Data Model implementation
- Command manager
- XML Loader and Saver

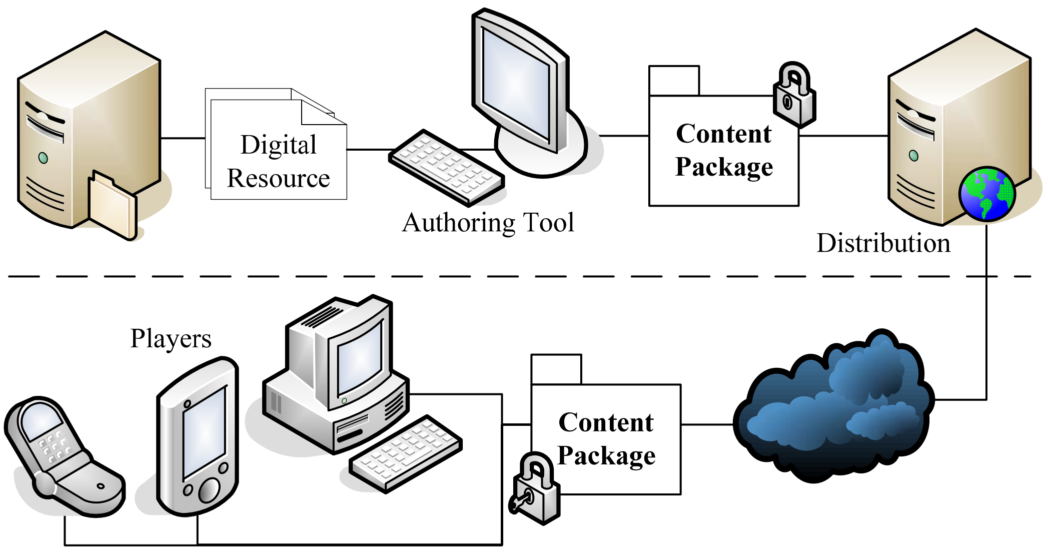
The purpose is to show the typical way of programming an application which use the AXOM in order to manipulate objects. It shows the typical cycle load-manipulate-save. This test is intrinsically performed on the other demonstrators which perform this operation, but this specific demo is needed to understand how the applications are built against the object model which has been realized.

The details regarding the implementation of the AXOM can be recovered into the following documents publicly accessible on the AXMEDIS web site:

DE3.1.2.3.2	Specification of AXMEDIS Command Manager
DE3.1.2.3.3	Specification of AXMEDIS Object Manager and Protection Processor

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 (Motion Picture Group, International Standard Organization, www.chiariglione.org/mpeg, [Chiariglione, MPEG], [Burnett et al., 2005]), WEDELMUSIC (Web Delivering of Music, http://www.wedelmusic.org), [Bellini et al., 2003], SCORM (Sharable Content Object Reference Model, http://www.adlnet.org/, [Mourad et al., 2005]), OMA (Open Mobile Alliance, http://www.openmobilealliance.org/), TV-AnyTime Forum (http://www.tv-anytime.org/, [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) management, 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 early 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-end-users (final users): the so called Business to Consumer (B2C) distribution. Among them, 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 the following Figure. 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, etc.). The distributor on the basis of the contract with the producer (that, in turn it is based with its contract with the content owner, the 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 cashed/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.



Scenario on content package and its usage

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; etc., 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/execution (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. Thus the final combination of requirements leads to the needs to provide different modalities of managing the package structure and organization of digital resources and metadata and digital rights management [Lin et al., 2005], [Koushanfar et al., 2005], [Mourad et al., 2005]. Moreover, also the protection information to protect and unprotect the package and the licenses to grant the authorization to the users about the usage of digital resources and packages have to be managed accordingly.

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]. This allows making accessible a large number of different content types and formats for the same content. This approach is a sort of precooking paradigm for producing in advance a large set of versions (e.g., different for type format and/or resolution) of the same digital resource, protected and ready to be used by the final users according to their requests.

More recently, 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 is pervading/penetrating the business areas; thus producing the needs of associating with the digital content suitable protection models for Business to Business (B2B) transactions. These new needs are putting in evidence a large set of new requirements that need to be satisfied by the packaging formats discussed in this paper. 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. This allows to establish rules for the content usage according to some business model renting: pay per play, free access, combined with the user status: student, teacher, industry, etc., [Wang., 2003];
- reducing promotion and business distribution costs since (according to the previous point) the content can be distributed in protected formats among the business users by means of the network and the same content package can be reused by others for creating more valuable packages (including the package in other content) 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, thus avoiding the production of any combination of precooked packages on the basis of all the possible combinations of user's preferences and device capabilities;
- 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 is not provided. This is only the begin of what is going to happen since the circulation of protected content is provoking the needs of reusing it for creating other products in the respect of the fixed rights. On the other hands, these models are not presently capable to cope with nested levels of protection and metadata.

In order to access at the above mentioned features for B2B and B2C actors the development of a set of enabling technologies is needed, and among them:

- **the extension of the package model to cope with B2B requirements** and in particular for addressing:
    - any kind of digital resources, including professional formats typically non delivered to the final users and not only audiovisual, but also 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 quickly 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 nested levels of packaging as well as the whole package, preventing unauthorized manipulation of packaged content/resources, and at the same time, leaving intact 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 preventing/controlling rights infringement;
  - 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.

Due to the complexity of the requirements imposed by specific domains, in many cases, the above mentioned packaging models have been oriented on some specific area such OMA on mobiles, SCORM/IMS for educational content, Tv-AnyTime for video, WEDELMUSIC for multimedia music integration, etc., and none of them is capable to cope with B2B aspects.

Among the mentioned packaging models, the MPEG-21 results to be the closer to the above mentioned needs and allows to be customized/extended. The MPEG-21 standard provides a unified standard to address packaging, adaptation profiles, protection and licensing formats [Burnett et al. 2003]. 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 very complex task since it presents 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 digital rights management 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 us to reasoning and verifying the model validity against extended requirements of the B2B area. The study has produced the AXMEDIS package model as a specialization of MPEG-21 model. The paper explained also the rationales that recently have been applied to MPEG-21 model to cope with B2B requirements and that were identified by the AXMEDIS consortium. 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 produced package model is capable to cope with the above mentioned requirements and it is supported by a concrete architecture and tools based on a core module called AXMEDIS Object Model, AXOM. It can be used to develop MPEG-21 as well as AXMEDIS compliant tools coping with the mentioned problems and more specifically those about flexibility and security and B2B needs. 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) to satisfy the above reported requirements. Many issues discussed and solved in AXMEDIS are not addressed at level of MPEG standard since the latter has a different purpose. In particular, aspects related to the definition of a specify 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].

The paper is organized as follows. In section 2, a short overview of MPEG-21 is given, recalling the main elements of the standard which are related to the work of this paper. A description of the AXMEDIS model including the features added to cope with B2B aspects, packing model for protected and protected objects, and relationships with MPEG-21 are provided in Section 3. In the same Section 3, the formal model for describing and reasoning about content packages and related operations (protection, extraction, etc.) is presented. Section 4 reports a description of the AXMEDIS Object Manager (AXOM) including data model, definition of the MPEG-21 model, expandability of the model, relationships from AXMEDIS and MPEG-21, MPEG-21 Load and Save, etc. In Section 5, an example of how the AXOM allows the access and the manipulation of content elements preserving the control about the authorization of grant and un-protection is presented. In the same Section, It also discusses the formalization of custom commands respecting the enforcement of rights, thus demonstrating how the AXOM can be used for creating content authoring/playing tools in a safe manner. Conclusions are drawn in Section 6.

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 [Chiariglione, MPEG]; presently most of the parts are mature while others are under evolution. The two parts of the standard which are the most relevant 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 of the standard have to be considered, and in particular: Rights Expression Language (REL) and Right Data Dictionary (RDD) [MPEG-21 RDD]. These are only marginally relevant for the purpose of this paper.

The DID defines how DIs (Digital Items) 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 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.). Along with these two fundamental parts of MPEG-21 model, comes a set of features that involves collateral functionalities of the model, such as inclusion of components from external XML documents and composition of digital items through references. MPEG-21 model carries out this task through the use of XInclude paradigm acquired from XML W3C.

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

3.1 AXMEDIS Object Manager: Work performed

3.1.1 AXMEDIS Package Model and the MPEG-21 standard

The MPEG-21 standard has been defined to satisfy different application domains. It has been defined in order to address many B2B and B2C requirements to support the value chain from creators to consumers (end-users), on the other hand, it specifically does not provide an easy support for the detailed needs reported 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, some 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.

The other requirements mentioned above can be satisfied only if the package model is expanded and supported by additional tools 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. The satisfactory of 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;

Therefore, the AXMEDIS model has been built on top of MPEG-21 to cope with the above mentioned issues for modeling:

- B2B relevant metadata for improving content distribution and accessibility for B2B partners;
- content structure, including aspect regarding the access to content description and metadata without the need of un-protecting the content package. This feature is described in the following.

These features are concretized in the model with the MPEG-21 extensions described in the next subsections. Please note that none of the extensions applied in AXMEDIS to the MPEG-21 are transforming AXMEDIS objects in non MPEG-21 compliant objects. Moreover, any AXMEDIS object is an MPEG-21 compliant object, while the opposite is not true (not all the MPEG-21 objects are AXMEDIS objects).

3.1.2 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 information/metadata to cope with the object trading has been added and grouped in the so called AXInfo (AXMEDIS Information). The AXInfo defined a set of specific descriptors/metadata (formalized in XML) to cope with:

- 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;
- Workflow and historical information about changes. A set of data describing the evolution of the content along the value chain: creation, improvements, changes, integration, etc.

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.

In the formalization of package models and functionalities the following notation is used:

- data types are represented with capital letter and their definition is marked with the keyword "Type";
- Variables are defined with keyword "Var";
- `a:Agg` represents variable a of Type Agg. Please note that, when the variable Type can be inferred by the context the Type is omitted to make the formalization lighter;
- symbol ":=" represents a definition and/or an assignment to the left;
- symbol "=" represents a production of a result on the right;
- a type can be defined providing the possible values of the type separated by | , each value may have associated, between [ ], a list of values of a specified type, for example an integer list can be specified as:

  `Type list Int := int-list[int, IntList] | nil` and
  `Var li:list Int :=int-list[0,int-list[1,nil] ]`

  is a list variable *`li`* with values 0 and 1. A type may be also defined as the union of other types (e.g. *`Type A := B | C`*);
- A list of elements is defined with the pre-fixed keyword "list" (e.g., *`list Int`*, is an integer list);
- *`<a1, a2, a3>`* represents a constant list of three elements, *`a1, a2`* and *`a3`* and < > is an empty list.

3.1.3 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…
```

Where: *`Enc`* is an encoded (e.g., encrypted) version of a *`Pack`* . 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 digital rights management 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 (e.g., encrypted) 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 model `pi` which produces the encoded chunk by using (in this case, for example) the function `encrypt()`. The opposite operation of `unprotect()` allow us to get back again 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
```

As a result, 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>
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
```

In the following section, a different package model is proposed to solve the problem, enhancing the model.

3.1.4 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. It has to be capable of (i) distinguishing from public and non public descriptors, and of (ii) automatically reporting at the higher levels the nested public information, in order to guarantee the consistency of structure when complex packaged objects are passed back and forward from non-protected and protected representation models.

In order to cope with these aspects, two kinds of descriptors modeling respectively public and private content descriptions have to be present in the structure of the packaged objects. 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 in the sequel as PM2:

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 to retrieve 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` according to 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.

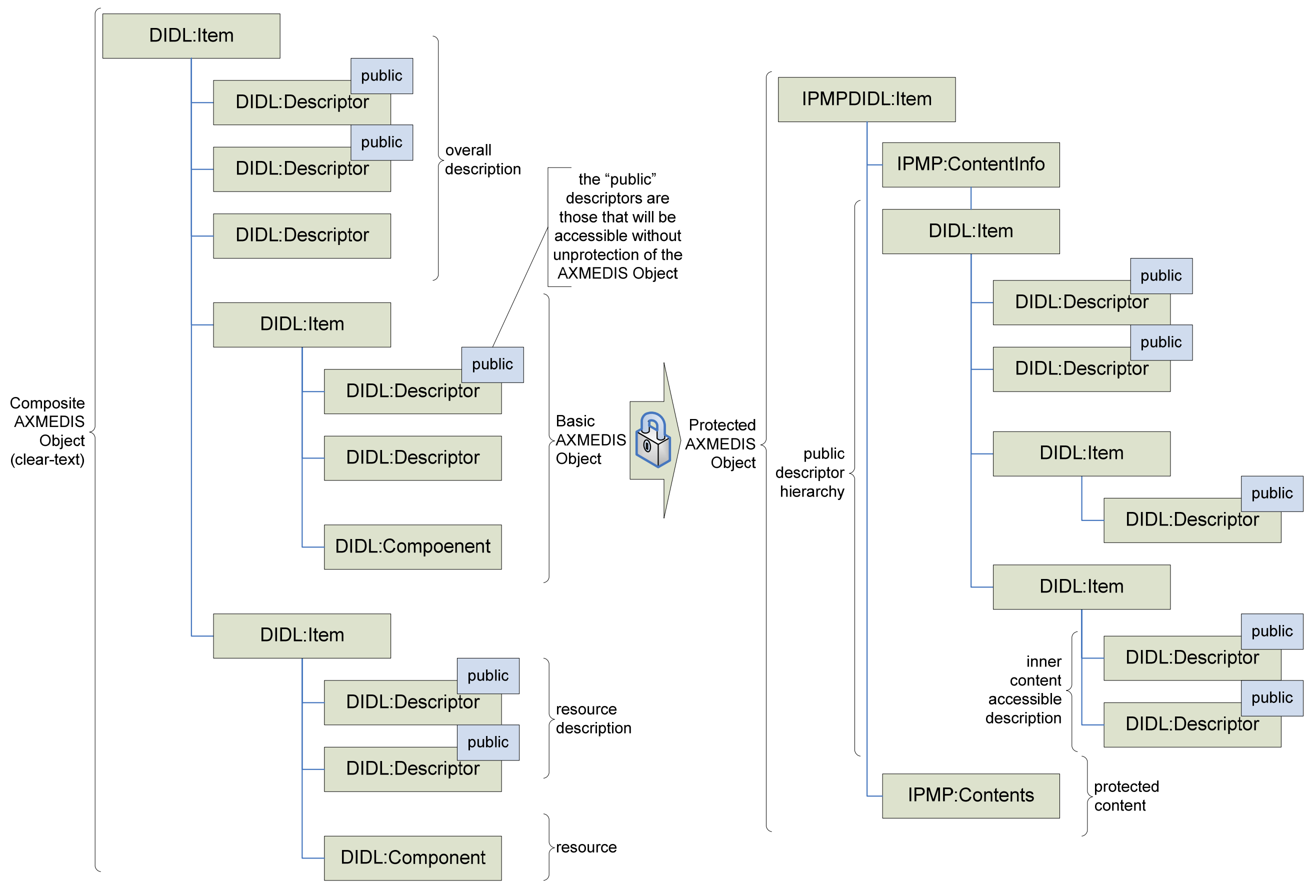
```
getmetadata(o2-bis) = <d3>
getmetadata(getcontent(o2-bis, 0)) =
      getmetadata(prot-obj[pi3, pdtree3, e3]) =
            desc-tree[<d4>, < desc-tree[<d5>, < >], desc-tree[<d6, d8>, < >] >]
```

3.1.5 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 Digital Item parts, AXMEDIS has selected which content parts (represented as MPEG-21 DI parts) have to be protected. In such a way, protected AXMEDIS Objects remain valid MPEG-21 Digital Items.

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 and defined in a proper XML Schema. 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 application domain problems. 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.

In following Figure, the content transformation after a protection process in terms of MPEG-21 elements has been sketched. Only the relevant MPEG-21 elements have been considered from both DIDL and IPMP standards to make the figure simpler and immediate. The Descriptor element, as an example, is not a leaf since it can contains a Statement as its child element. The diagram has been simplified to put in evidence which part of the content structure remain accessible even after the protection.



AXMEDIS Object after protection in terms of MPEG-21 elements

Please note that some *Descriptor* elements have been tagged with "public" attribute. The use of the "public" attribute has been introduced by AXMEDIS and it has to be used to be sure that some content information will be accessible without needing an un-protection after any protection process. The attribute "public" allows to define the *`Pub-desc`* of the formal 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. In this way, the hierarchy of content composition is in somehow replicated to organize the corresponding public metadata when the content is protected.
AXMEDIS authoring tool is capable to apply multiple protection algorithms on each content element, by specifying how many "protection tools" have to be used and in which order. These tools can target the whole content asset or just a subpart. A protection tool is practically a content processing algorithm which can be retrieve and installed as a plug-in at run-time by the playing/editing application [Nesi et al., 2006].
As in MPEG-21 IPMP, the protection procedure requires the encoding of a given portion of multimedia content on the basis of the chosen protection algorithm (protection tool). The result of the protection process is then embedded in a XML element. The protection procedure is reversible if the information on how content has been protected is retrieved at run-time when the authorization is obtained on the basis of the license.

3.1.6 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

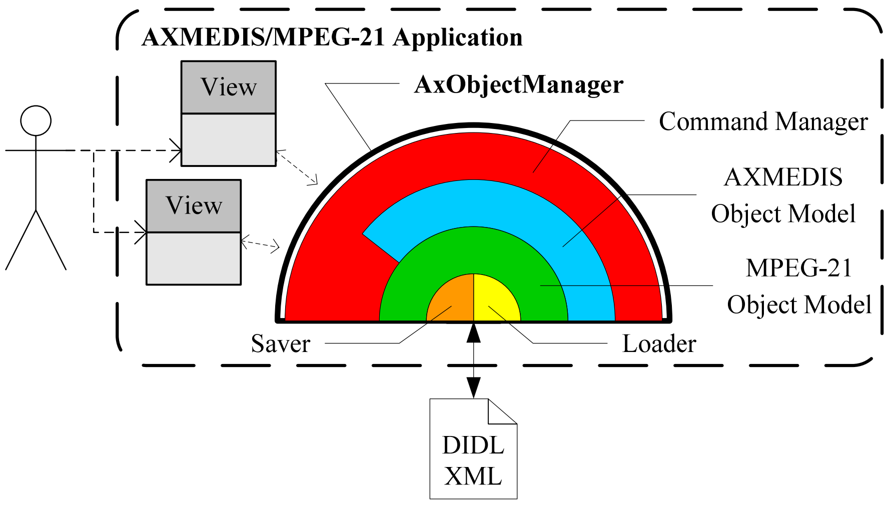
create software tools which handle DIs in the respect of the DRM rules. The AXOM is used in the AXMEDIS tools that have to cope with content objects and it is compliant with MPEG-21, extending it with additional features and data structures. Thus, it can be used as a basis to design and develop a large 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, covering the whole content life from production to consumption.
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 leaving intact the possibility of accessing to visible metadata of nested included levels;
- 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 the basis of AXOM system 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 the following Figure, 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.



AXMEDIS Object Manager (AXOM) layers

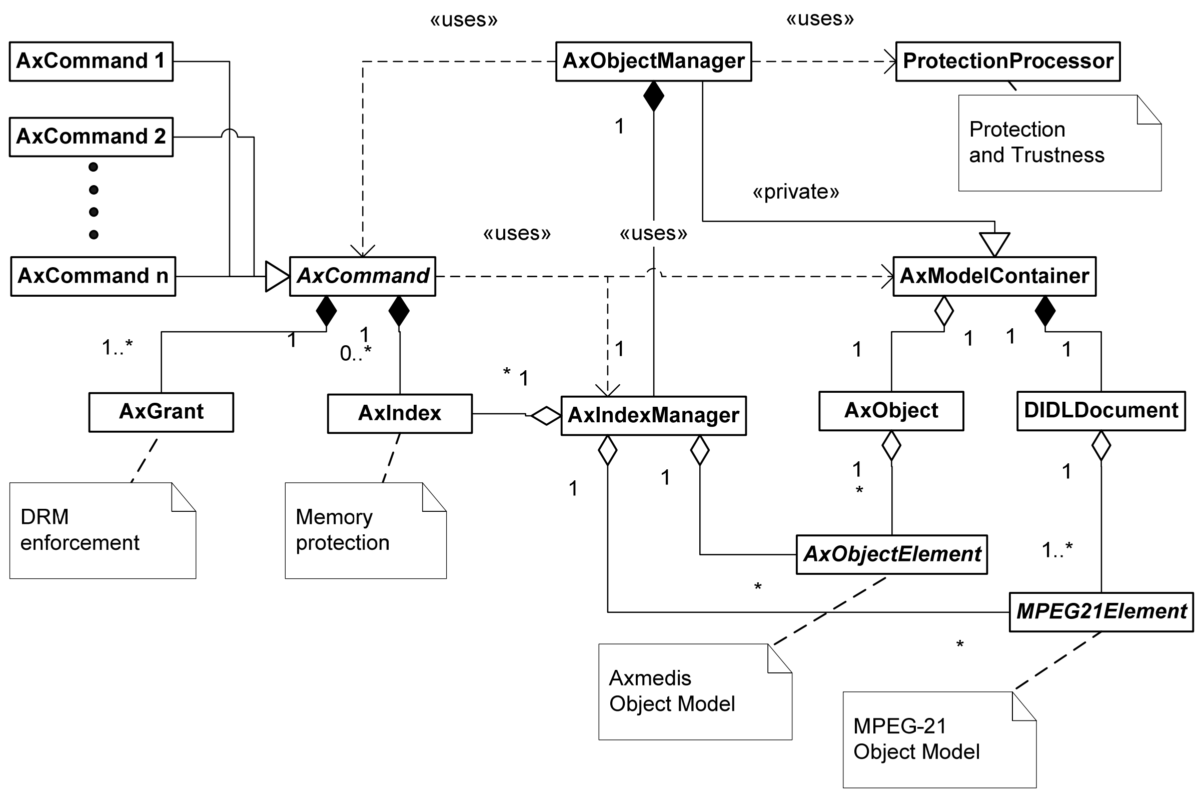
Layering these modules has been important to maintain compatibility with MPEG-21 standard. Defining features on object model that manage MPEG-21 content packages grants to the system the ability to deal with all kinds of MPEG-21 objects and not only with AXMEDIS content packages. It is indeed mandatory that the provided AXOM API has to cope with MPEG-21 object model directly (as shown in the diagram) to allow manipulation of MPEG-21 DIs. The commands are basically of two kinds depending on which object model they target.

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 layering. 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 mean of *AxObjectManager* and *AxCommand* set.

These classes offer:

- a range of executable commands targeting the manipulation of both AXMEDIS and MPEG-21 object models
- 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.



AXOM class hierarchy

As depicted in the above Figure, the Command pattern [Gamma et al., 1995] has been applied. 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, preventing 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 targets have to be indicated using logical references that prevent the access to the physical addresses referring to the digital resources. This avoids accessing via pointer at the memory of the whole object model. 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) it autonomously performs 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) it exposes functionalities to protect/unprotect content elements and to query for authorization of requested manipulations in a transparent manner with respect to the connected services.

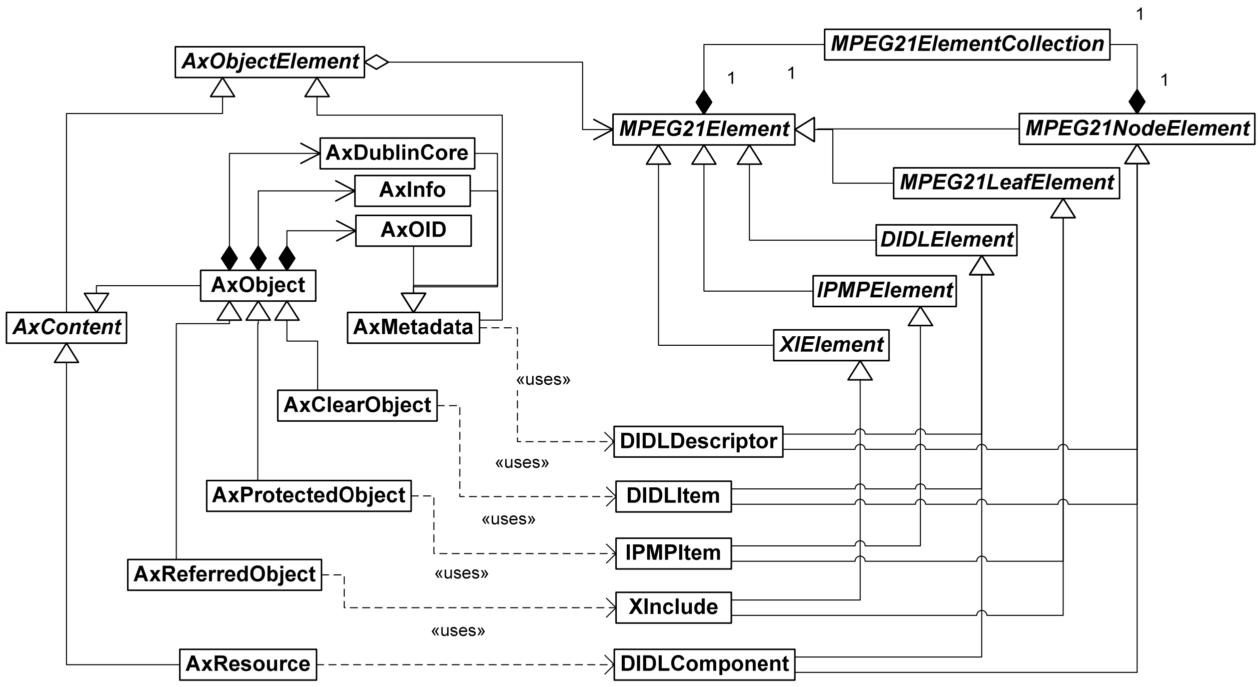
When digital resources have to be accessed (e.g., for their rendering on a rendering component/module view) the chain of un-protection tools is activated, thus allowing to establish a direct stream from the encapsulated resource and the rendering component/module view. Command execution returns content elements information, providing a 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.

3.1.7 MPEG-21 and AXMEDIS Data Models

The MPEG-21 Object Model (reported in the following Figure) 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 as content elements. 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. Some 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.

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., ordering), 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).



AXMEDIS and MPEG-21 Object Models relationships

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 called 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. The AXMEDIS Object Model is composed by the following classes:

- ***AxMetadata*** represents generic metadata. It mainly provides methods to access 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 contains 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 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.

3.1.8 MPEG-21 Load and Save

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. In fact, this approach opens to exploit 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 straight forward sequence of events, that notify the occurrence of XML entities (i.e. elements, characters, etc.), which has to be managed by a unique handler. When loading a DI, that contains 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 outcome from handling more than one event. To solve this problem, the object loading has been realized by 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 defined. Concrete classes have been implemented, achieving different loading behaviors. The loading process leans on a factory mechanism. Abstract Factory pattern has been used in conjunction with Factory Method pattern in order to create elements given a namespace and an element name [Gamma et al., 1995]. 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 the MPEG-21 Loader, it is 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 worth to point out that the writers do not have to directly write elements in XML syntax. In order to avoid this triviality, *XMLWriter* class has been realised. 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.

3.1.9 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 three 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. In the following, the functional model of some operations is presented.

In the following example, the function `add()` changes the content package structure by adding a new content element in the proper list, on the basis of its type. The following two examples are for the addition of a new descriptor and of a new component:

```
add(clear-obj[d:list Desc, c:Comp], new-desc:Desc) =
      clear-obj[addtolist(d, new-desc), c]
add(clear-obj[d:list Desc, c:list Comp], new-comp:Comp) =
      clear-obj[d, addtolist(c, new-comp)]
```

Then, a resource (`res2`) and a descriptor (`d3`) are added to package `o1`.

```
Var o1:Pack := clear-obj[<d1, d2>, <res1>]
Var o1 := add(o1, <res2>) = clear-obj[<d1, d2>, <res1, res2>]
```

Thus:
```
add(o1, <d3>) = clear-obj[<d1, d2, d3>, <res1, res2>]
```

An operation that modifies the information inside content elements is the `edit().` It gives the possibility of editing content element attributes, such as the metadata attributes of the content, or other details. The new attributes are given by means of a new content element, from which the attributes are copied. The following two examples are for the change of a descriptor and of a component:

```
edit(clear-obj[d, c], old-desc, new-desc) =
      clear-obj[replace(d, old-desc, copyattributes(new-desc, old-desc)), c]

edit(clear-obj[d, c], old-comp, new-comp) =
      clear-obj[d, replace(c, old-comp, copyattributes(old-comp, new-comp)) ]
```

where `copyattributes()` is a function which substitutes attributes on the left element by copying them from the right and `replace()` function replaces an element in a list with another one.

An important operation for creating content package is the function of embedding an existing digital resource into a content package. The embed function fills an empty resource/place with a given source (`asset`), this could be a byte stream or a file URL. As a result the inclusion of the resource inside the content package is obtained:

```
embed(clear-obj[d, c], empty-res:Res, asset) =
      clear-obj[d, replace(c, empty-res, fillres(empty-res, asset)]
```

3.1.10 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*– similarly to the definition of *add* in the functional model, this command class has been defined to allow augmenting the content package with descriptions, digital resources or inner packages. 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. It mainly implements the *edit* of the functional model, and has a constructor:
    - *AxCommandEdit(AxObjectElement dataElement, AxIndex editIndex)* change the attributes of the target element by copying them from *dataElement*;

The full list of available commands can be downloaded from http://www.AXMEDIS.org in the specification documents area.

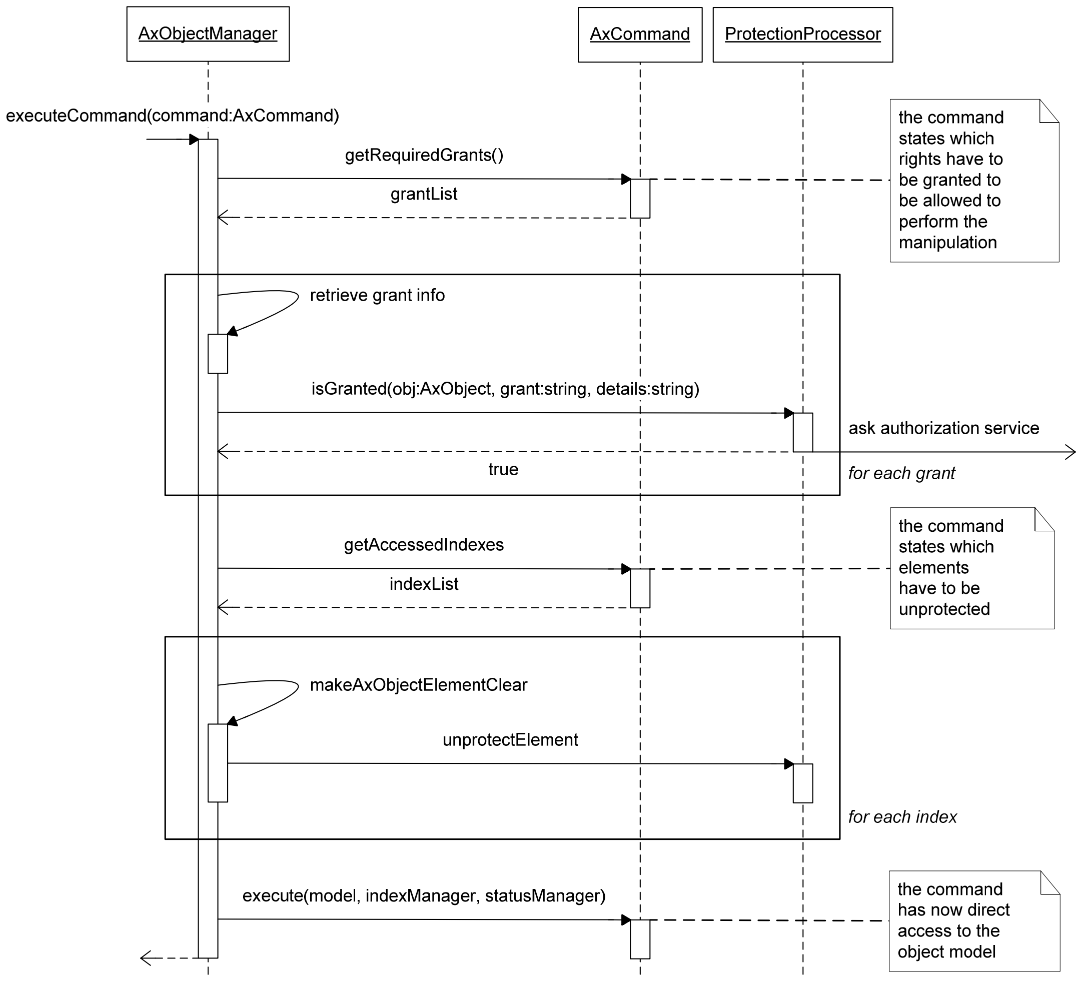
In this section, an example on how the AXOM capabilities can be exploited by authoring and player tools is presented. The usage of AXOM functionalities is quite simple. For the manipulation of content package the creation of a new command and its execution are needed. Targets of the content access and manipulation are expressed with logical references: *AxIndex* objects. The AXOM is responsible of providing such 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 just 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 following statement explains how the *add() (AxCommandAdd)* is performed on a protected package automatically unprotecting the target element and re-protecting again the result of the addition.

```
add(prot-obj[pinfo, pdtree, e], new-elem) =
protect(add(unprotect(prot-obj[pinfo, pdtree, e]), new-elem), pinfo)
```

The operation of `unprotect()` is performed only if who/what is invoking/performing it has been successfully authorized. 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 (see boxes in the following Figure). Sequence diagram of Figure 6 shows how a command is queried for both authorization and un-protection. In fact, the *getRequiredGrants()* method of *AxCommand* is polymorphic and every command can define a list of granted rights. Every right has to be checked against the proper authorization service. Only after the authorization phase has succeeded, the *AxObjectManager* asks to the command which content package elements have to be unprotected in order to perform their manipulation. After the un-protection, the execution leaves the responsibility of performing the needed change to the command, passing to it the whole object model, already unprotected in the required parts.


AxObjectManager
AxCommand
ProtectionProcessor
executeCommand(command:AxCommand)
getRequiredGrants()
grantList
the command states which rights have to be granted to be allowed to perform the manipulation
retrieve grant info
isGranted(obj:AxObject, grant:string, details:string)
ask authorization service
true
for each grant
getAccessedIndexes
indexList
the command states which elements have to be unprotected
makeAxObjectElementClear
unprotectElement
for each index
execute(model, indexManager, statusManager)
the command has now direct access to the object model

Sequence diagram of a command execution

3.1.11 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, in which the meaning of each term used for defining rights in licenses is formally specified. Examples of rights data dictionaries are the MPEG-21 RDD [MPEG-21 RDD] and that defined by Mi3P [MI3P DICT]. The AXOM model has the needed flexibility to formalize the enforcement of both the models for the aspects related to authoring and rendering/playing.

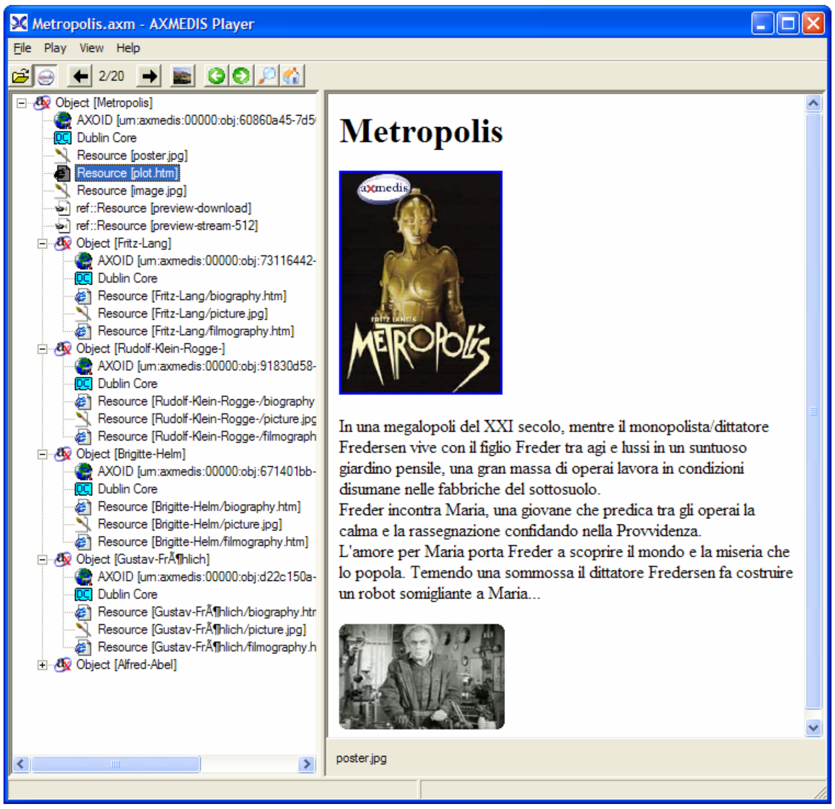
```
class AxCommandImageMirror : public AxCommand
{
   private AxIndex targetIndex;

   //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);

        //a resource attribute is accessed to gather some information
        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.

3.1.12 Using AXOM for a player

The following pseudo-code is a simplification to explain how simple is realizing a player application which is able to access packages which include multi-format resources and related descriptions. The script creates an instance of the *AxObjectManager* giving the filename (or URL) of the object to be loaded, it makes access to the root element (rootObj) using the root index and retrieves the AXOID. Afterwards all the metadata of the object are accessed (via DOM structure) and finally the index of the first content is obtained (with the assumption that it is a resource) and with this index a stream to read the resource content is obtained.

```
…
// creating a manager to manipulate “myobject.axm” file
AxObjectManager myObjectManager = new AxObjectManager(“myobject.axm”);

// obtaining a clone of root object general information
AxObject rootObj = myObjectManager.getAxObjectElement(myObjectManager.getRootIndex());

// retrieving content identifier
string axoid = rootObj.getAXOID();

// accessing content metadata obtaining a clone of each content metadata
foreach index in myObjectManager.getAxMetadataIndexes(myObjectManager.getRootIndex())
{
   AxMetadata meta = myObjectManager.getAxObjectElement(index);

   // retrieving XML that represents content description
   DOMElemente metaDOM = Meta.getDOMElement();
   ... metadata access ...
}
// accessing children of the root level in the object structure
vector AxIndex children = myObjectManager.getChildIndexes(myObjectManager.getRootIndex())

//(assuming first children is a digital resource) obtaining a clone of resource
information
AxResource res = myObjectManager.getAxObjectElement(children[0]);

// retrieving MIMEType of the target resource
string mimeType = res.getMIMEType();

// obtaining a byte stream directly from the digital resource (e.g. for rendering)
stream resStream = myObjectManager.getResourceAsset(children[0], “play”);
…
```

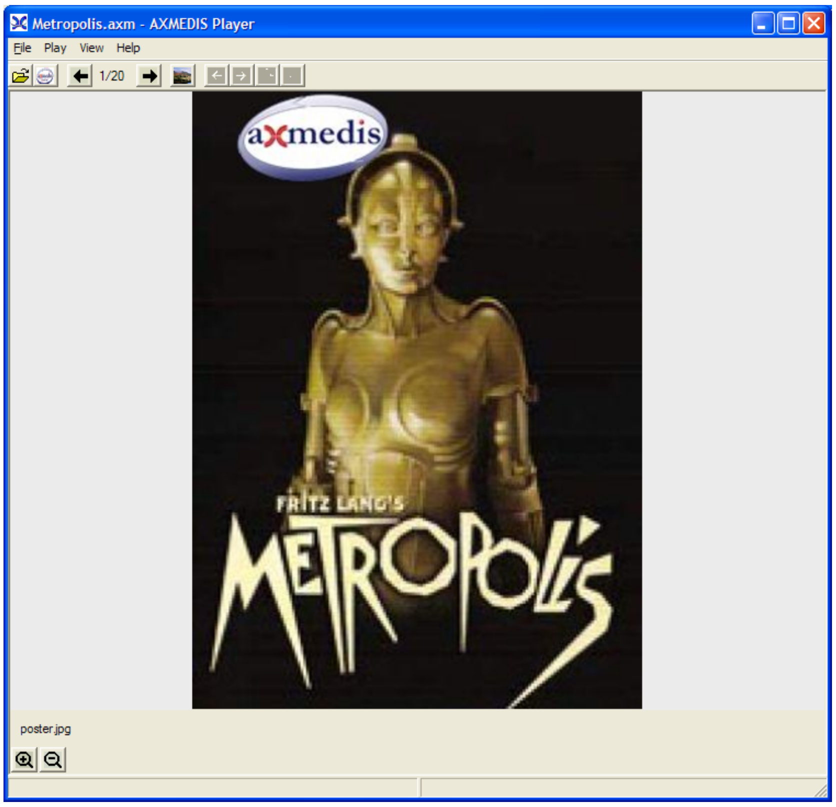


Object [Metropolis]
AXOID [urn:axmedis:00000:obj:60860a45-7d5
Dublin Core
Resource [poster.jpg]
Resource [plot.htm]
Resource [image.jpg]
ref::Resource [preview-download]
ref::Resource [preview-stream-512]
Object [Fritz-Lang]
AXOID [urn:axmedis:00000:obj:73116442-
Dublin Core
Resource [Fritz-Lang/biography.htm]
Resource [Fritz-Lang/picture.jpg]
Resource [Fritz-Lang/filmography.htm]
Object [Rudolf-Klein-Rogge-]
AXOID [urn:axmedis:00000:obj:91830d58-
Dublin Core
Resource [Rudolf-Klein-Rogge-/biography
Resource [Rudolf-Klein-Rogge-/picture.jpg
Resource [Rudolf-Klein-Rogge-/filmograph
Object [Brigitte-Helm]
AXOID [urn:axmedis:00000:obj:671401bb-
Dublin Core
Resource [Brigitte-Helm/biography.htm]
Resource [Brigitte-Helm/picture.jpg]
Resource [Brigitte-Helm/filmography.htm]
Object [Gustav-FrÃ¶hlich]
AXOID [urn:axmedis:00000:obj:d22c150a-
Dublin Core
Resource [Gustav-FrÃ¶hlich/biography.htr
Resource [Gustav-FrÃ¶hlich/picture.jpg]
Resource [Gustav-FrÃ¶hlich/filmography.h
Object [Alfred-Abel]

Metropolis

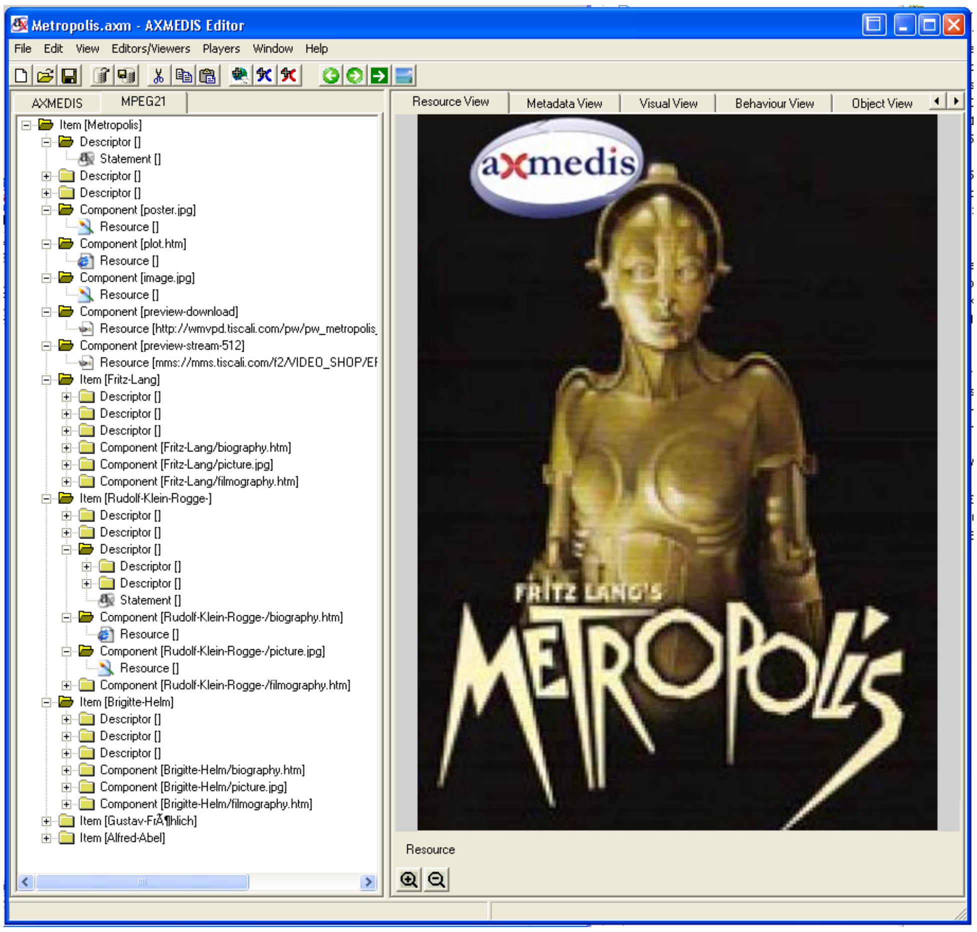
In una megalopoli del XXI secolo, mentre il monopolista/dittatore Fredersen vive con il figlio Freder tra agi e lussi in un suntuoso giardino pensile, una gran massa di operai lavora in condizioni disumane nelle fabbriche del sottosuolo.
Freder incontra Maria, una giovane che predica tra gli operai la calma e la rassegnazione confidando nella Provvidenza.
L'amore per Maria porta Freder a scoprire il mondo e la miseria che lo popola. Temendo una sommossa il dittatore Fredersen fa costruire un robot somigliante a Maria...

(a)



(b)

AXMEDIS player: (a) AXMEDIS hierarchy view and HTML text, (b) view of resources



```
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/f2/VIDEO_SHOP/EF
  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]
```

Views of the AXMEDIS Editor for the same object shown in the above Figure by the player. 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.

3.2 AXMEDIS Object Manager: work performed summary

- The event-driven mechanism has been defined and implemented in the AXMEDIS/MPEG-21 hierarchy elements.
- The support for MPEG21 File Format have been implemented
- The support MPEG 21 DIP has been added
- Now it is possible to move an object portion into another object: the “move” command has been implemented and support moving from one object to another oject
- Now it is possible to copy a sub-tree of the object into another position of the object itself.
- IPMP tool parameters (e.g., key) have been implemented
- The device-fingerprint extraction procedures have been implemented on PDA.
- The device-fingerprint extraction procedures have been tested on Linux
- AXMEDIS Object signature generation has been partially implemented
- Resource consumption from protected assets allows seeking along the resource stream, maintaining the unprotection buffer at the correct status.

3.3 AXMEDIS Object Manager: problems and actions

Some of the desired functionalities have not been yet realized:

- B2B enforcement
- Finalize and integrate object signature generation

- AXOM for mobile

Other functionalities are present but they are not implemented in an efficient way:
- Memory leaks are due to the uncorrect usage of XMLTranscode function of XercesC library

4 AXMEDIS Object Preprocessor and Postprocessor (EPFL, DSI)

This technology has been substituted with MPEG21 File Format based on ISOMedia File format.

The set of tools implementing the AXMEDIS Objects Preprocessing and Postprocessing provide two main functionalities. On one hand, conversion from textual XML to binary form and vice-versa has to be implemented in order to allow both objects presentation in a human readable (and editable) format and objects download or stream through bandwidth-critical channels. On the other hand, when objects are converted from a binary format to textual XML on the end-user terminal, the existent references to external objects must be resolved.

4.1 Object Preprocessor and Postprocessor: State of the art

As a textual format, XML tends to be rather verbose since it has not been designed to work ideally in a real-time, constrained and streamed environment such as those found in the i-TV, in the multimedia or in the mobile industry. As long as structured documents (HTML, for instance) were basically composed of only few embedded tags, the overhead induced by textual representation was not critical. Applications are now dealing with larger and highly structured XML documents. More and more XML files are exchanged in many fields such as in i-TV, database synchronization or enterprise application integration (EAI). In these cases, the XML textual representation is verbose and its processing very inefficient.
Therefore, optimized binary encoding mechanisms have been developed in order to reduce the bandwidth needed to transmit XML objects when necessary. To overcome this lack of efficiency of textual XML, MPEG-7 Systems has defined a generic framework to facilitate the carriage and processing of MPEG-7 descriptions: BiM (Binary Format for MPEG-7) [17]. It enables the streaming and the compression of any XML documents. Among the advantages of BiM there is its flexibility. In fact, a BiM decoder can deal with evolution of XML languages. Technically, at encoding phase a level of compatibility is chosen for the bitstream. The encoding process adds necessary information to ensure that an old decoder will be able to skip unknown part of the bitstream. This feature allows also XML private extensions to be easily inserted within the original XML document without breaking interoperability. If forward compatibility is not needed, the redundancy is removed and the bitstream becomes more compact. Given the proliferation of MPEG standards and other external organizations referring to MPEG-7 Systems BiM technology, it seemed that a dedicated standard would considerably ease the referencing and maintenance of the standard independently of the MPEG-7 Systems specification. These are the reasons which led to the creation of a new project of the MPEG group (ISO/IEC WG11): the so called MPEG-B (23001-1) MPEG-B -- Part 1: Binary MPEG format for XML.

Open source implementations of BiM include:

- the BIM Open Source project which is entirely Java based and composed by four different subprojects. The first project is the BIM Database Standard Edition, a personal information manager. The second project is the BIM Dynamic Development Environment. The third project is the BIM Telnet Client. The telnet client is a smaller scale project. It is useful for storing aliases, triggers, patterns, and some artificial intelligence in a basic telnet interface. The fourth project is the BIM Obfuscator used to obfuscate Java code variables and methods. It basically protects name integrity across packages.
- The MPEG-B reference software is available but its efficiency is very limited. It can be used for testing the output of other implementations but it could hardly be integrated in larger frameworks which may need to use it as a plug-in.

- The most efficient BIM implementation is the EXPWAY BiM commercial implementation called BinXML™ [20]. This product is very efficient in terms of compression performances provided. On the MPEG-7 test set, BinXML™ reaches an average compression ratio of 85 %. The average number of bits per elements and attribute is 2.93 bits (min = 0.003, max = 13, standard deviation = 1.76) against 136.7 bits for the original XML files.

The only open source implementation in C++ of BiM is then the MPEG reference software. If its efficiency should not be adequate to the AXMEDIS purposes another solution called XMill is available. XMill is a special-purpose compressor for XML data that typically achieves twice the compression rate over existing compressors, such as gzip. MILL is an open source program delivered under a BSD License, it is written in C++, and it is available for all 32-bit MS Windows (95/98/NT/2000/XP), and all POSIX (Linux/BSD/UNIX-like OSes) [21] platforms.
Similar to gzip, XMill is a command-line tool that works on a file-by-file basis. A given file with extension '.xml' is compressed into a file with extension '.xmi'. Any other file without extension '.xml' is compressed into a file by appending extension '.xm'. Reversely, the original file is obtained by replacing extension '.xmi' with extension '.xml' or by removing extension '.xm'. Alternatively, the user writes the output to the standard output and optionally read from the standard input.

A widely deployed open source project for XML validation, authoring and manipulation is the Xerces toolset. Xerces (named after the Xerces Blue butterfly) provides XML parsing and generation. Currently, there are two validating versions, in Java and C++, implementing the W3C XML and DOM (Level 1 and 2) standards, as well as the de facto SAX (version 2) standard. The parsers are highly modular and configurable. There is also some support for XML Schema (draft W3C standard). Xerces-C++ is a validating XML parser written in a portable subset of C++. It provides a shared library for parsing, generating, manipulating, and validating XML documents.

4.2 Object Preprocessor and Postprocessor: The problems

At the 71st MPEG meeting held in Hong Kong (China) the MPEG-21 DID standard [6] was amended to substitute the REFERENCE element with the W3C Xinclude schema. This is due to the fact that in December 2004 W3C published the XML Inclusions (XInclude) 1.0 Recommendation (see W3C XINCLUDE). The functionality provided by the REFERENCE element in the first edition of ISO/IEC 21000-2 can be substantially provided by XInclude. XInclude also provides functionality that extends beyond that of the REFERENCE element. For example, the document author can provide a fallback in case the referenced elements are unable to be retrieved. Hence in this second edition the REFERENCE element has been removed and similar functionality is achieved using XInclude.
Some important differences between REFERENCE and XInclude have to be considered when using XInclude instead of REFERENCE. These include:

1. A REFERENCE element operates on its parent element, in that the content of the referenced element is included into the content of the parent element of the REFERENCE element. Whereas an XInclude include element operates on itself, in that the included content replaces the XInclude include element.

2. XInclude processing can result in addition of xml:base and xml:lang attributes to the top-level included elements.

3. XInclude processing does not include the attribute value inheritance of the REFERENCE element.

The above mentioned amendment to the MPEG-21 schema requires an extension of the Xerces capabilities in order to support the validation of the Xinclude schema. In particular, the Xerces Schema Validator needs to be extended to accept INCLUDE element coming for XInclude Namespace. A normal schema validator cannot manage this element since they are xml:base and xml:lang attributes. A special validator has to be provided in order to validate the XInclude syntax and the whole schema by considering the referred elements

4.3 Object Preprocessor and Postprocessor: Work performed

The work for the development of the AXMEDIS Reference Resolver is based on the Xerces toolset for the authoring and manipulation of XML files. The application opens an XML file and searches for references inside it. So far, a reference is defined as something like this:

```
<reference file="C:\referenced_file.xml"  XPath="node[2]/something[0]/node[3]/something">
```

where:

- the `file` attribute is the referenced file.
- the `XPath` attribute is the path inside the referenced file of one specific node. This path is in format XMLPath.

Then the Reference Resolver opens all the referenced files and searches for the referenced nodes and, finally, it substitutes the link or reference with the referenced element.

The following snapshot displays the user interface of the reference resolver.

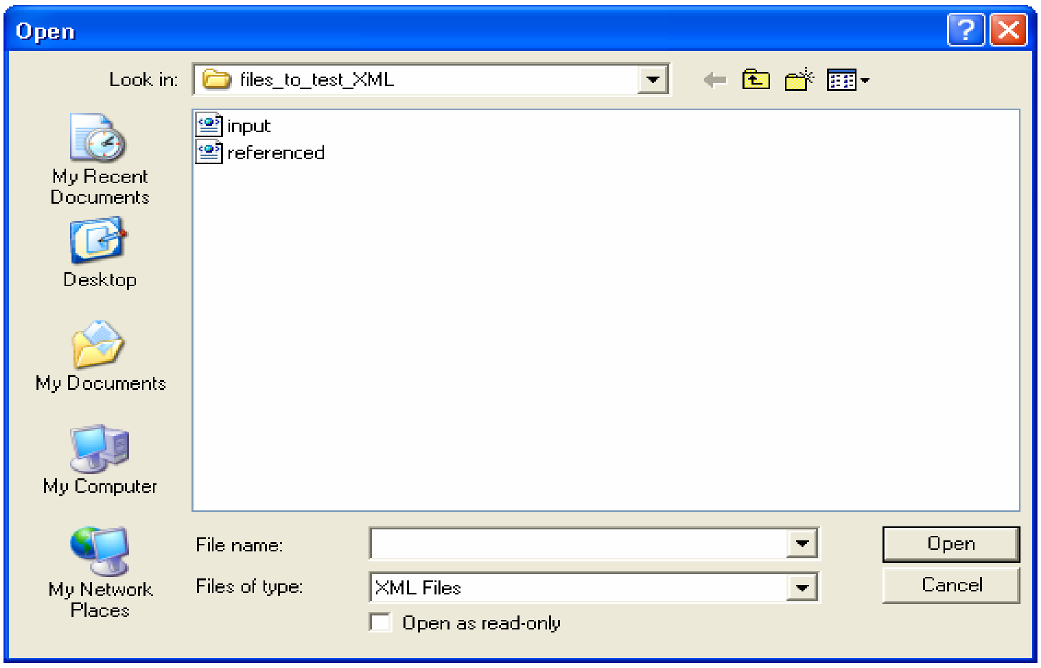


Figure 1 - The user is required to open the input file which is referencing the "referenced.xml" file

Suppose the input XML is named as input.xml and the referenced XML file is named as referenced.xml. The input XML file has the following content.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE personnel>
<!-- DOCTYPE personnel SYSTEM "personal.dtd"-->

<!-- @version: -->

<personnel>

  <person id="Big.Boss" >
    <name><family>Boss</family> <given>Big</given></name>
    <email>chief@foo.com</email>
    <link subordinates="one.worker two.worker three.worker four.worker
five.worker"/>
  </person>


  <person id="one.worker">
    <reference file = "referenced.xml" XPath =
"/AUCTIONBLOCK/ITEM/BIDS/BID[0]" />
    <name><family>Worker</family> <given>One</given></name>
    <email>one@foo.com</email>
    <link manager="Big.Boss"/>
  </person>

  <person id="two.worker">
    <name><family>Worker</family> <given>Two</given></name>
    <email>two@foo.com</email>
    <link manager="Big.Boss"/>
  </person>

  <person id="three.worker">
    <name><family>Worker</family> <given>Three</given></name>
    <email>three@foo.com</email>
    <link manager="Big.Boss"/>
  </person>

  <person id="four.worker">
    <name><family>Worker</family> <given>Four</given></name>
    <email>four@foo.com</email>
    <link manager="Big.Boss"/>
  </person>

  <person id="five.worker">
    <reference file = "referenced.xml" XPath =
"/AUCTIONBLOCK/ITEM/BIDS/BID[3]/BIDDER" />
    <name><family>Worker</family> <given>Five</given></name>
    <email>five@foo.com</email>
    <link manager="Big.Boss"/>
  </person>

</personnel>
```

The referenced XML file has the following content:

```
<?xml version="1.0"?>
<AUCTIONBLOCK>
   <ITEM>
      <TITLE>Vase and Stones</TITLE>
      <ARTIST>Linda Mann</ARTIST>
      <DIMENSIONS>20x30 inches</DIMENSIONS>
      <MATERIALS>Oil</MATERIALS>
      <YEAR>1996</YEAR>
      <DESCRIPTION/>
      <PREVIEW-SMALL    SRC="images/burl-s.jpg"    WIDTH="300"    HEIGHT="194"
ALT="Vase and Stones"><!--The small image--></PREVIEW-SMALL>
      <PREVIEW-LARGE SRC="images/burl.jpg" WIDTH="640" HEIGHT="413" ALT="Vase
and Stones"/>
      <BIDS>
         <BID>
            <PRICE>3300</PRICE>
            <TIME>9:19:32 AM</TIME>
            <BIDDER>John</BIDDER>
            <TIMESTAMP>1315</TIMESTAMP>
         </BID>
         <BID>
            <PRICE>3200</PRICE>
            <TIME>8:18:31 AM</TIME>
            <BIDDER>Andrew</BIDDER>
            <TIMESTAMP>1308</TIMESTAMP>
         </BID>
         <BID>
            <PRICE>3100</PRICE>
            <TIME>2:48:08 PM</TIME>
            <BIDDER>Chris</BIDDER>
            <TIMESTAMP>1307</TIMESTAMP>
         </BID>
         <BID>
            <PRICE>3000</PRICE>
            <TIME>2:47:58 PM</TIME>
            <BIDDER>opening price</BIDDER>
            <TIMESTAMP>1298</TIMESTAMP>
         </BID>
      </BIDS>
      <MTIME>
         <TIMESTAMP>1315</TIMESTAMP>
         This is an example of mixed content
      </MTIME>
   </ITEM>
</AUCTIONBLOCK>.
```

In the input XML file there are two parts associated with the referenced XML file. They are the following in the input XML file.

```
<reference file = "referenced.xml" XPath = "/AUCTIONBLOCK/ITEM/BIDS/BID[0]" />
```

and

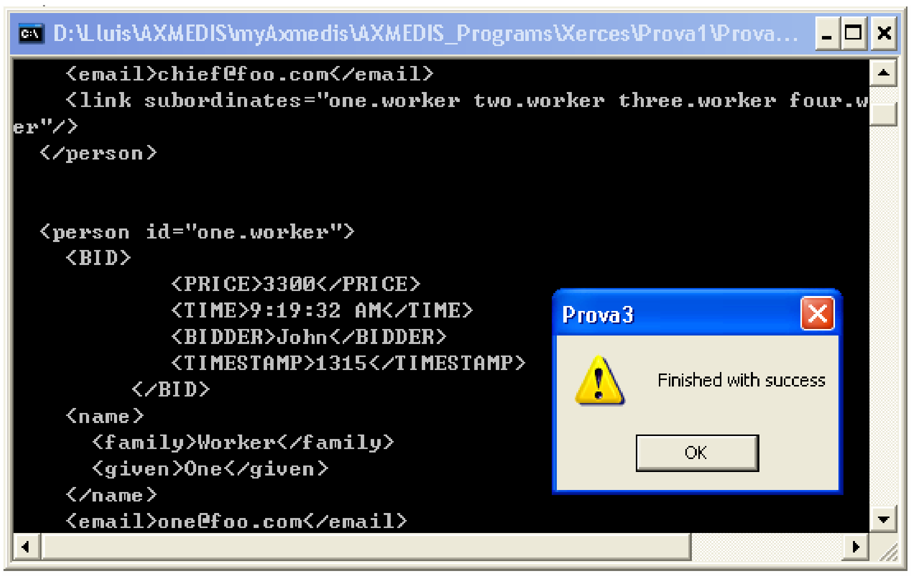
```
<reference file = "referenced.xml" XPath =
      "/AUCTIONBLOCK/ITEM/BIDS/BID[3]/BIDDER" />
```

After resolving, the output file replaces these two parts with the corresponding elements in the referenced XML file. For example:

```
<reference file = "referenced.xml" XPath = "/AUCTIONBLOCK/ITEM/BIDS/BID[0]" />
```

This part is replaced by the following code from the referenced XML file. The result is demonstrated in the following snapshot.

```
<BID>
   <PRICE>3300</PRICE>
   <TIME>9:19:32 AM</TIME>
   <BIDDER>John</BIDDER>
   <TIMESTAMP>1315</TIMESTAMP>
</BID>
```


D:\Lluis\AXMEDIS\myAxmedis\AXMEDIS_Programs\Xerces\Prova1\Prova...
Prova3
OK

```
    <email>chief@foo.com</email>
    <link subordinates="one.worker two.worker three.worker four.w
er"/>
  </person>


  <person id="one.worker">
    <BID>
            <PRICE>3300</PRICE>
            <TIME>9:19:32 AM</TIME>
            <BIDDER>John</BIDDER>
            <TIMESTAMP>1315</TIMESTAMP>
        </BID>
    <name>
      <family>Worker</family>
      <given>One</given>
    </name>
    <email>one@foo.com</email>
```

Finished with success

Figure 2 - Output of the reference resolver after processing input.xml

Following the recent modifications of the MPEG-21 DID standard the current version of the resolver described above needs to be modified and extended accordingly.

Therefore, we build such a function std::istream* XinculdeResovler(Xinclude &xi) which use the parameter Xinclude defined from DSI and returns an standard istream. For such a thing, we introduced Libxml2 library since the xerces C++ does not support the Xinclude.

Libxml2 is the XML C parser and toolkit developed for the Gnome project (but usable outside of the Gnome platform), it is free software available under the MIT License. XML itself is a metalanguage to design markup languages, i.e. text language where semantic and structure are added to the content using extra "markup" information enclosed between angle brackets. HTML is the most well-known markup language. Though the library is written in C a variety of language bindings make it available in other environments.

4.4 Object Preprocessor and Postprocessor: problems and actions

The following tools and solutions have been analysed:
- XML to BIN and BIN to XML (the partial implementation of BIN to XML produced by DSI)
- Resolution of references (an experiment produced by EPFL using a library)
- MPEG-21 file format

For the purpose of demonstrating a proof of concept and for the purpose of realizing an effective tool the XML-BIN solution is not viable since too complex and far from the research activity needed for the AXMEDIS project with the aim of producing an efficient tool for downloading content. Presently neither the inventor of that technology in MPEG are capable of providing a working tool to test and integrate. They are working on this problem since several years.
The consortium decided to go in the direction of realizing an MPEG-21 file format saver and loader (for binary files) in addition to the AXMEDIS/MPEG-21 XML saver and loader already in place. This will allow us to produce and distribute simple and compact files.

5 AXMEDIS Editor, authoring tool (DSI)

The AXMEDIS Editor should allow the user to create and manipulate AXMEDIS objects.

5.1 Editor: Work performed

The AXMEDIS editor allows to:

- load and save protected and unprotected AXMEDIS object as a xml file
- make a query on the AXMEDIS Database and get the content from the db.
- create a new AXMEDIS object
- acquire a definitive AXOID from AXCS
- register the protection information and metadata to the AXCS
- visualize and manipulate the hierarchy of both the AXMEDIS and MPEG-21 (see Hierarchy Editor and Viewer)
- view resources (images, documents, video, audio) in an AXMEDIS object using the internal resource viewers
- manipulate images using the internal image editor
- use content processing plugins to manipulate resources
- edit metadata in xml using the Metadata Editor
- define the PAR (Potential Availabe Rights) for the object using the DRM editor
- define protection information, the MPEG-21 IPMP information (some parameters are still missing) using the Protection Editor
- produce the SMIL resource relating the other basic resources present in the AXMEDIS Object using the Visual Editor
- define DIP methods using the Behaviour View
- see the workflow status using the Workflow Viewer
- be activated from the workflow and to notiy the completion of an activity
- save the AXMEDIS object as AXM file (XML file with resources encoded using base64)
- save the AXMEDIS object using the MPEG21 File Format.

The AXMEDIS editor has been revised to use the AXOM events to allow to keep in synch the different views on the AXMEDIS object.

The following is the AXMEDIS editor window for a new object, on the left are present the hierarchy views (AXMEDIS and MPEG-21), in the center the different views on an element and on the right some activities that can be performed.

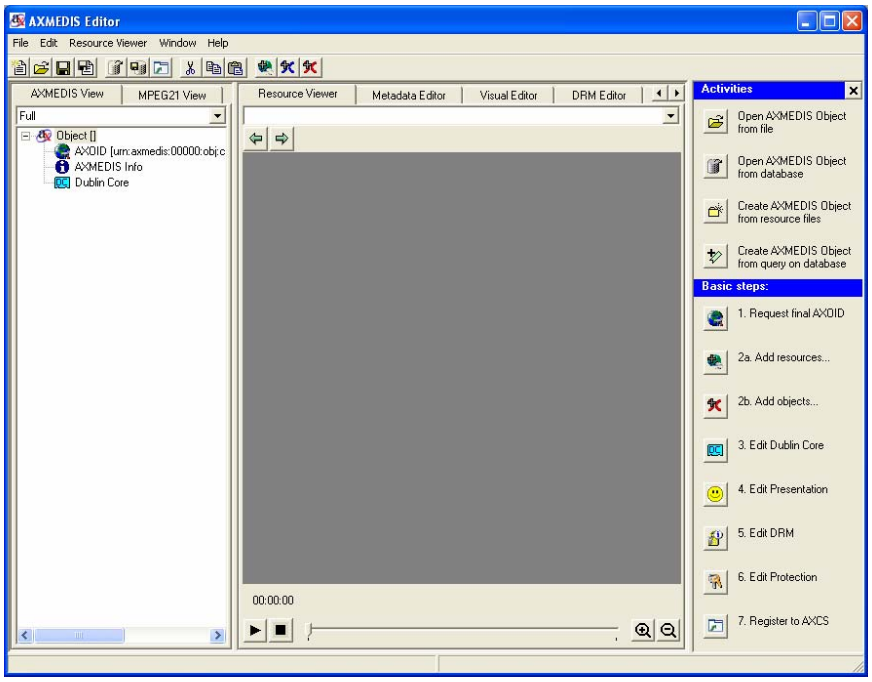


Figure 3 - AXMEDIS editor window for a new object

In the following picture an AXMEDIS object was loaded and a resource is opened (double click)

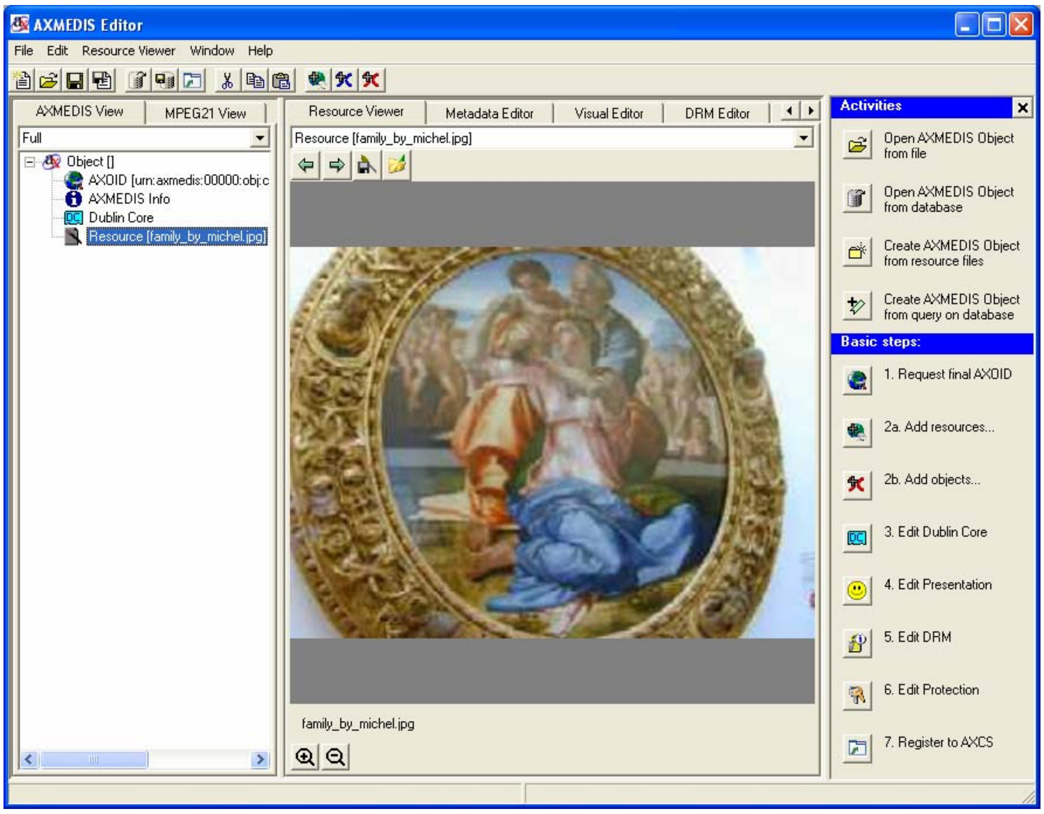
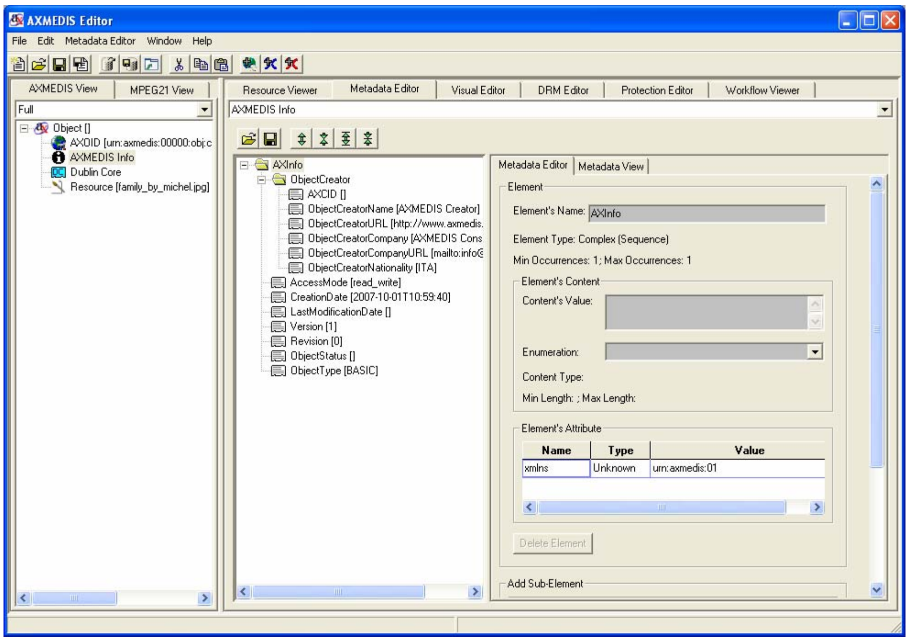


Figure 4 - An AXMEDIS object is opened and the resource is rendered

In the following image is opened with Metadata Editor the AXInfo of an object



AXInfo
ObjectCreator
AXCID []
ObjectCreatorName [AXMEDIS Creator]
ObjectCreatorURL [http://www.axmedis.
ObjectCreatorCompany [AXMEDIS Cons
ObjectCreatorCompanyURL [mailto:info@
ObjectCreatorNationality [ITA]
AccessMode [read_write]
CreationDate [2007-10-01T10:59:40]
LastModificationDate []
Version [1]
Revision [0]
ObjectStatus []
ObjectType [BASIC]

Element
Element's Name: AXInfo
Element Type: Complex (Sequence)
Min Occurrences: 1; Max Occurrences: 1
Element's Content
Content's Value:
Enumeration:
Content Type:
Min Length: ; Max Length:
Element's Attribute

Name	Type	Value
xmlns	Unknown	urn:axmedis:01

Figure 5 - The AXMEDIS editor displays a metadata element.

For a more complete description of the functionalities provided see the "DE5.0.1.1 AXMEDIS Major Tools User Manuals"

6 Hierarchy Editor and Viewer (DSI)

The Hierarchy Editor and Viewer show the AXMEDIS and MPEG-21 hierarchy and allow manipulating them.

6.1 Hierarchy Editor and Viewer: Work performed

The AXMEDIS Hierarchy Editor and Viewer allows to:

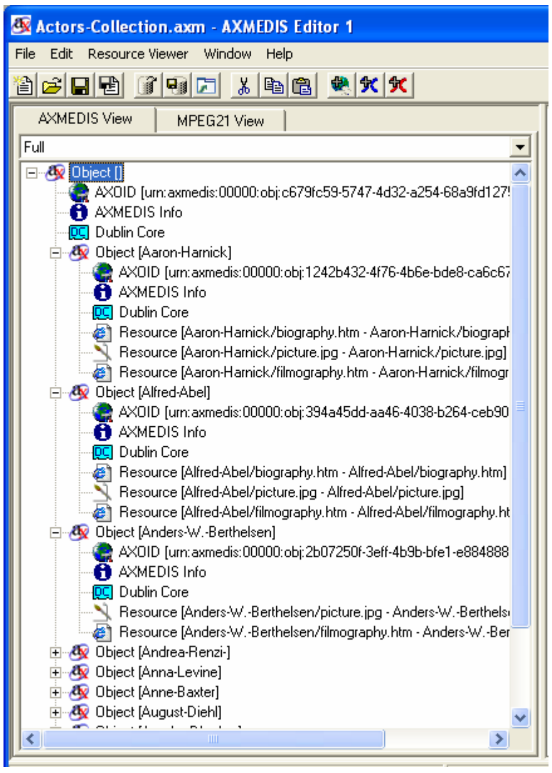
- visualize the AXMEDIS hierarchy as tree structure
- add a new AXMEDIS resource, AXMEDIS metadata or AXMEDIS object to an AXMEDIS object
- remove any element inside the object
- move the elements up or down
- drag & drop elements to move elements inside the object or from an object to another one
- drag & drop a file on the hierarchy to automatically add a resource
- copy & paste

The MPEG-21 Hierarchy Editor and Viewer allows to:

- visualize the MPEG-21 hierarchy as tree structure
- add a new MPEG-21 element (container, item, descriptor, etc.)
- remove any element inside the object
- move the elements up or down
- drag & drop elements to move elements inside the object or from an object to another one

The AXMEDIS and MPEG21 Hierarchy Editors have been revised to use the AXOM events to allow to keep in synch the two views on the AXMEDIS object.

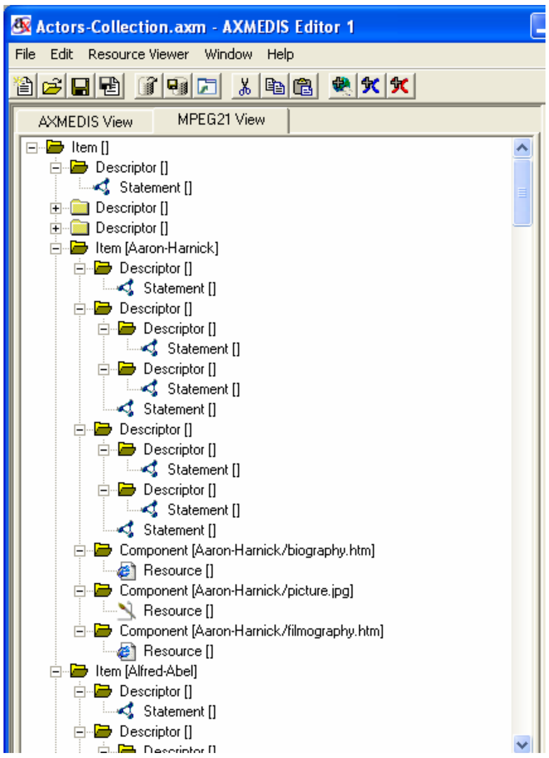
The following is an example of AXMEDIS Hierarchy:


Actors-Collection.axm - AXMEDIS Editor 1
File Edit Resource Viewer Window Help
AXMEDIS View
MPEG21 View
Full

```
Object []
    AXOID [urn:axmedis:00000:obj:c679fc59-5747-4d32-a254-68a9fd127
    AXMEDIS Info
    Dublin Core
    Object [Aaron-Harnick]
        AXOID [urn:axmedis:00000:obj:1242b432-4f76-4b6e-bde8-ca6c67
        AXMEDIS Info
        Dublin Core
        Resource [Aaron-Harnick/biography.htm - Aaron-Harnick/biograp
        Resource [Aaron-Harnick/picture.jpg - Aaron-Harnick/picture.jpg]
        Resource [Aaron-Harnick/filmography.htm - Aaron-Harnick/filmogr
    Object [Alfred-Abel]
        AXOID [urn:axmedis:00000:obj:394a45dd-aa46-4038-b264-ceb90
        AXMEDIS Info
        Dublin Core
        Resource [Alfred-Abel/biography.htm - Alfred-Abel/biography.htm]
        Resource [Alfred-Abel/picture.jpg - Alfred-Abel/picture.jpg]
        Resource [Alfred-Abel/filmography.htm - Alfred-Abel/filmography.ht
    Object [Anders-W.-Berthelsen]
        AXOID [urn:axmedis:00000:obj:2b07250f-3eff-4b9b-bfe1-e884888
        AXMEDIS Info
        Dublin Core
        Resource [Anders-W.-Berthelsen/picture.jpg - Anders-W.-Berthels
        Resource [Anders-W.-Berthelsen/filmography.htm - Anders-W.-Ber
    Object [Andrea-Renzi-]
    Object [Anna-Levine]
    Object [Anne-Baxter]
    Object [August-Diehl]
```

Figure 6 - Example of AXMEDIS hierarchy

The following is an example of MPEG-21 Hierarchy:



```
Item []
    Descriptor []
        Statement []
    Descriptor []
    Descriptor []
    Item [Aaron-Harnick]
        Descriptor []
            Statement []
        Descriptor []
            Descriptor []
                Statement []
            Descriptor []
                Statement []
            Statement []
        Descriptor []
            Descriptor []
                Statement []
            Descriptor []
                Statement []
            Statement []
        Component [Aaron-Harnick/biography.htm]
            Resource []
        Component [Aaron-Harnick/picture.jpg]
            Resource []
        Component [Aaron-Harnick/filmography.htm]
            Resource []
    Item [Alfred-Abel]
        Descriptor []
            Statement []
        Descriptor []
            Descriptor []
```

Figure 7 - Example of MPEG-21 hierarchy

For a more complete description of the functionalities provided see the "DE5.0.1.1 AXMEDIS Major Tools User Manuals"

7 Visual and Behavioral Editor and Viewer (EPFL)

The Behaviour Editor together with the Visual Editor will provide the user with the infrastructure to produce multimedia presentations. A multimedia presentation can be composed of many media objects (text, audio, video, vector graphics...). The user will use the Behaviour and Visual Editors to organize the media objects in space and time. The Behaviour Editor will complement the Visual Editor by adding time boundaries to the media objects. This means that every media object will be visible only for a period defined by the user. The simplest example for this is a slide show: the user specifies a group of slides and each slide is only visible during a slot of time defined by the user.
The Behaviour Editor will use a subset of the SMIL language to describe the multimedia presentation.
The Visual Editor will arrange the media objects (video, photo, text, etc) on the screen. The Visual Editor allows object placement in a 2-D (or possibly 3-D) environment. Moreover, it permits managing (i.e. moving, deleting, adding, etc…) object subparts which have spatial properties or constraints. The layout description will be done according to the SMIL W3C standard [9].

7.1 Visual and Behavioral Editor and Viewer: State of the art

SMIL authoring tools are implemented by different companies and consortia. Some of them are commercial tools whereas some are also available as free software or even open source toolsets.

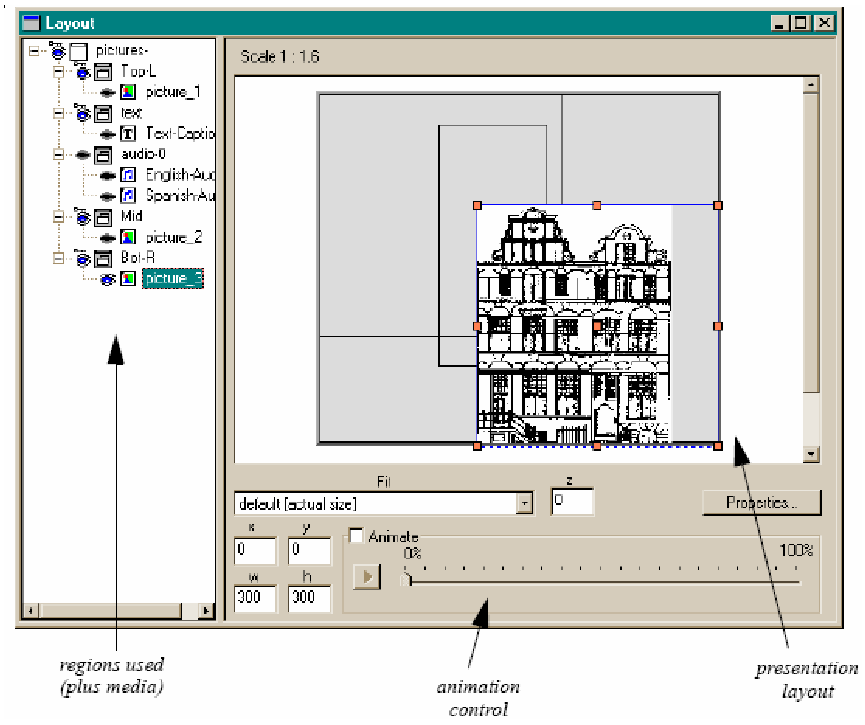
SMIL authoring tools implemented in C++ could not be found as open source projects. A powerful SMIL authoring tool commercialized by Oratrix [10] – GriNS2 - has been examined and can be used as reference implementation supporting most of the functionality needed by AXMEDIS. On the other hand, the most interesting open project is LimSee2 developed by INRIA [11] and it is implemented in Java. LimSee2 can also be used to test the conformance of the implemented Visual Editor and Viewer.
At the heart of any SMIL implementation lays a module in charge of managing the Timing and Synchronization of the decoded scenes. Orchestrating the timing of the various media elements in the presentation is, after all, what makes SMIL different from traditional Web media.

The following sections provide a brief overview and some snapshots of the GRiNS and LimSee2 features.

7.1.1 GRiNS Pro Editor for SMIL 2.0

GRiNS Pro Editor for SMIL 2.0 is a SMIL authoring application developed by Oratrix Software Development Company for creating, editing and maintaining streaming media documents for the RealOne player and also for other SMIL 2.0 compliant players, such as Internet Explorer6.0 and 3GPP. GRiNS Pro Editor can take source materials such as audio, video, image and text assets, and integrate them into a presentation which can be distributed via the Web.

GRiNS has an internal layout editor which manages visual and auditory resources. When working with template-based presentations, the template contains a number of regions to organize screen and other resources in the presentation. When the content creator uses these regions, he will typically not need to visit the Layout view. For more complex presentations, or when creating custom templates, the Layout view provides a means of creating, sizing, editing and placing new regions.



Fit: default [actual size]
z: 0
x: 0
y: 0
w: 300
h: 300

Figure 8 - Layout of the GRiNS editor

When activated, the Layout view shows a window similar to the GR*i*NS Player window and a separate editing window for selecting layout regions.
More information is available on the Oratrix Web site [10].
As one of the four views supported by the GRiNS SMIL authoring environment, the Timeline View is for editing the SMIL file in terms of time structure. The hierarchical structure is intended to represent the coarse

timing relationships reflected in the `<par>` and `<seq>` constructs. While the attributes associated with an element can be used to define more fine grained relationships which are often difficult to visualize with the hierarchical view alone. For this reason GRiNS also supports a timeline projection of a presentation. Unlike some systems which use the timeline as the initial view of the application, the GRiNS timeline displays relationships derived from the hierarchical structure. This means that the user is not tied to a particular clock or frame rate since the actual timing relationships will only be known at execution time. When working with the timeline view the user can define exact start and end offsets within `<par>` groups by using a synchronization arc shown as an arrow.

7.1.2 LimSee2

LimSee2 is an open authoring application for SMIL 1.0 and 2.0 languages conducted by the WAM team (Web Adaptation Multimedia) at INRIA, Grenoble, France.

LimSee2 has a multi-view solution (Layout view/Attributes view/Structure View/Timeline view) that renders the structure of the SMIL document at different levels during the authoring process: timing and synchronization, spatial layout, XML tree etc. The different views are synchronized (a modification in one view is immediately rendered in all the other views) and provide functionality that allows a user to manipulate and fine-tune a SMIL document without requiring a full knowledge of the language.

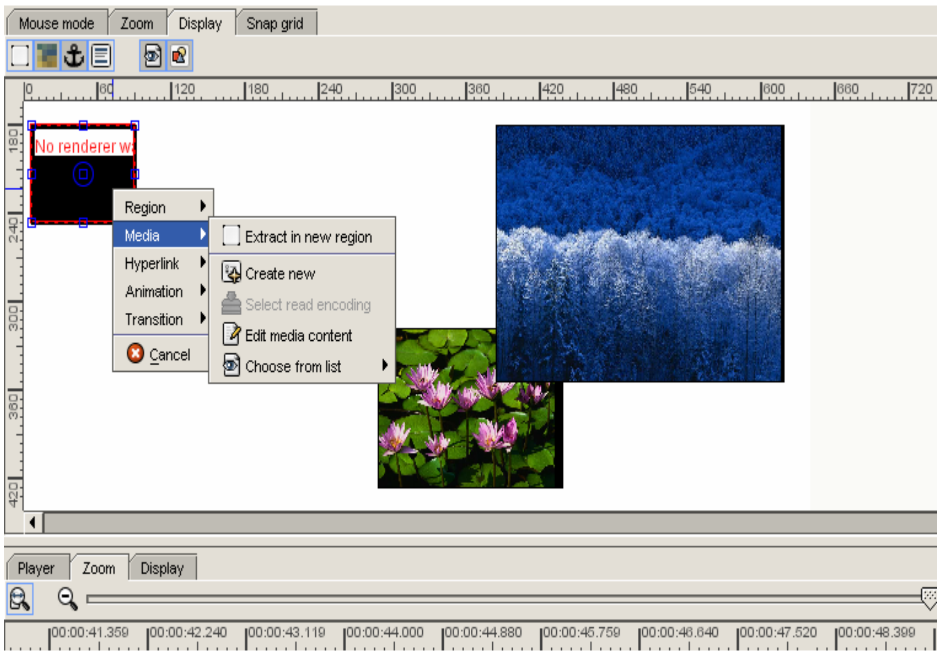


Figure 9 - Layout of the LimSee2 editor

The spatial layout of a SMIL document can be edited in LimSee2 in a 2D canvas, which constitutes a WYSIWYG (What You See Is What You Get) environment for a fixed time in the temporal scenario. SMIL regions can be easily moved, resized or created in a few clicks, media content can be directly previewed (for images, texts and videos), and region z-indexes can be adjusted with an intuitive drag-and-drop mechanism. The 2D canvas also provides traditional features such as a zooming tool or a customizable snap grid.
More information is available on the LimSee2 Web site at INRIA [11].

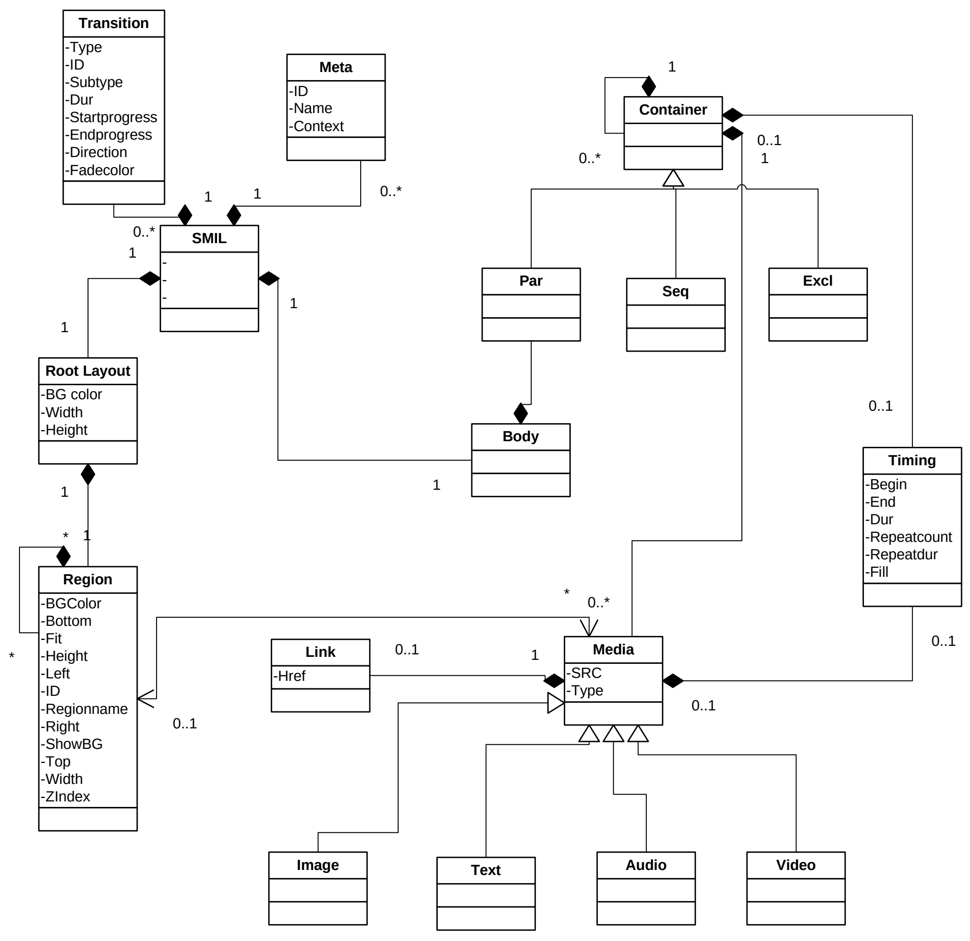
LimSee2 is one of the most efficient authoring tools in terms of editing the SMIL time structure. The timeline view is a faithful representation of the temporal behavior of the document. In this view the author can perform all editing operations related to synchronization between elements. The mouse cursor informs the user on the editing operations he can perform on the selected SMIL element. When the cursor takes the shape of a cross, an user action moves the object along the time axis (thus changing the begin and end attributes); if on the other hand it looks as a double arrow, the action changes the length of the object (duration attribute). Durations expressed as relative time placements between elements and represented by red arrows can be modified directly by moving the mouse while it is on these arrows.

It is important to notice that every editing action implies a consistent update on all the elements of the document. For instance changing the duration of an object can induce changes of time positions of other objects of the same composite or of ascendent composites. This consistency checking is continuously performed during user actions thanks to the use of the Cassovary constraint solver. This feature allows users to perceive on the whole document and in real time all the consequences of the modification they are performing.

7.2 Visual and Behavioral Editor and Viewer: Work performed (EPFL)

7.2.1 Module design for SMIL structure

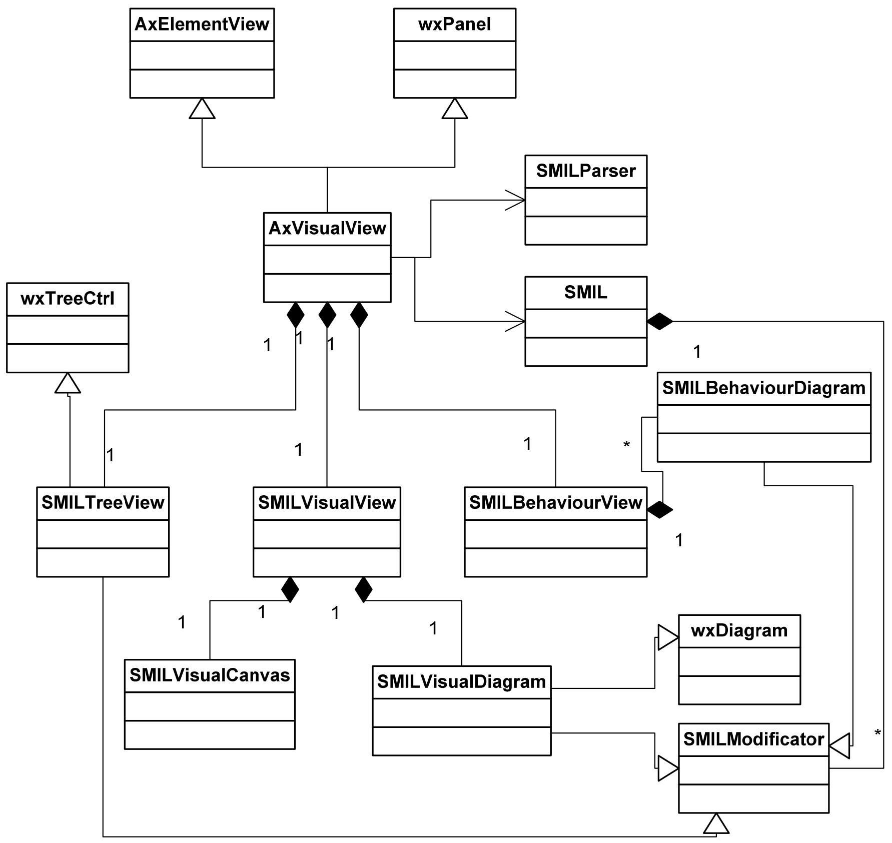
The SMIL structure has been modeled and implemented to recover the previous problem.



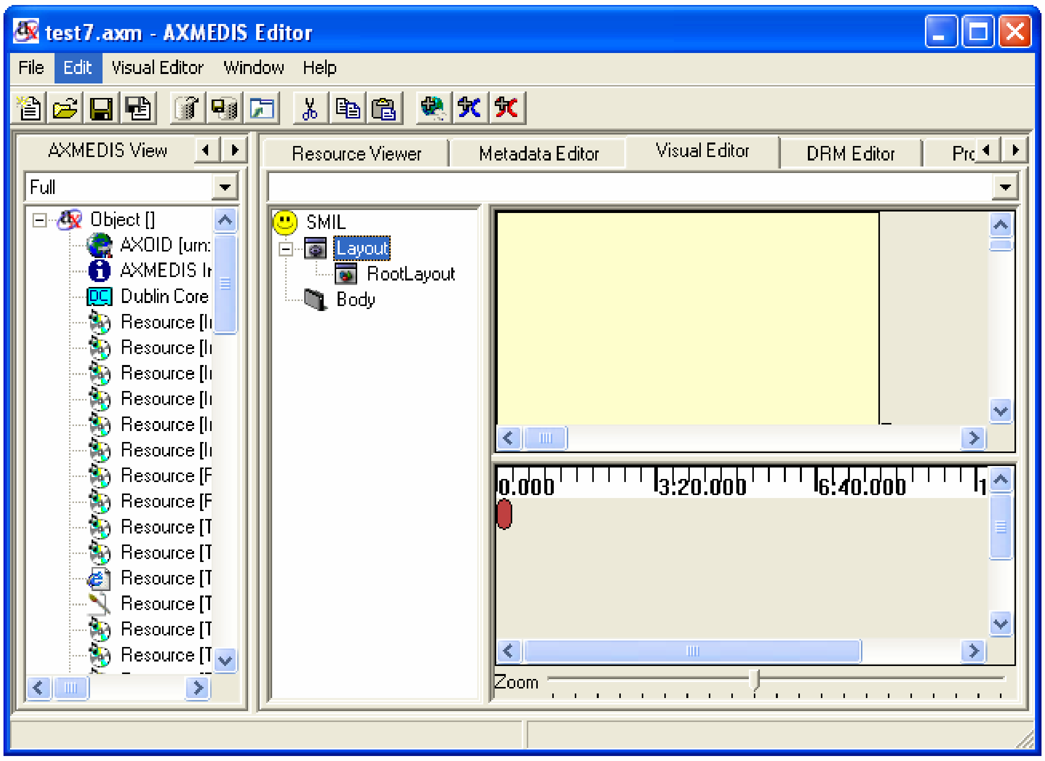
7.2.2 Module design for SMIL editor

The work on the AXMEDIS SMIL Editor has been finished. It is based on the WxWidgets OGL library for cross-platform graphical user interfaces implementation. The SMIL Editor allows editing of SMIL resources from the AxEditor by a visual interface. It is divided in three parts: the tree view part that shows the whole SMIL structure, the visual part that shows the regions used for resources displaying and the behaviour part that shows the timing structure and properties.

Module design:



Interface design:



7.2.3 Tree View part of SMIL editor

Hierarchy authoring of SMIL with Tree View Editor: the program allows the user to construct and edit of SMIL hierarchy components.

All the SMIL components are stored inside the hierarchy structure between the tags **<body>** and **</body>**.

There could be any combinations of tags of <par> and <seq> to support parallel and sequential temporal behaviour of the media objects. The following are some examples of Hierarchy structure of objects in SMIL

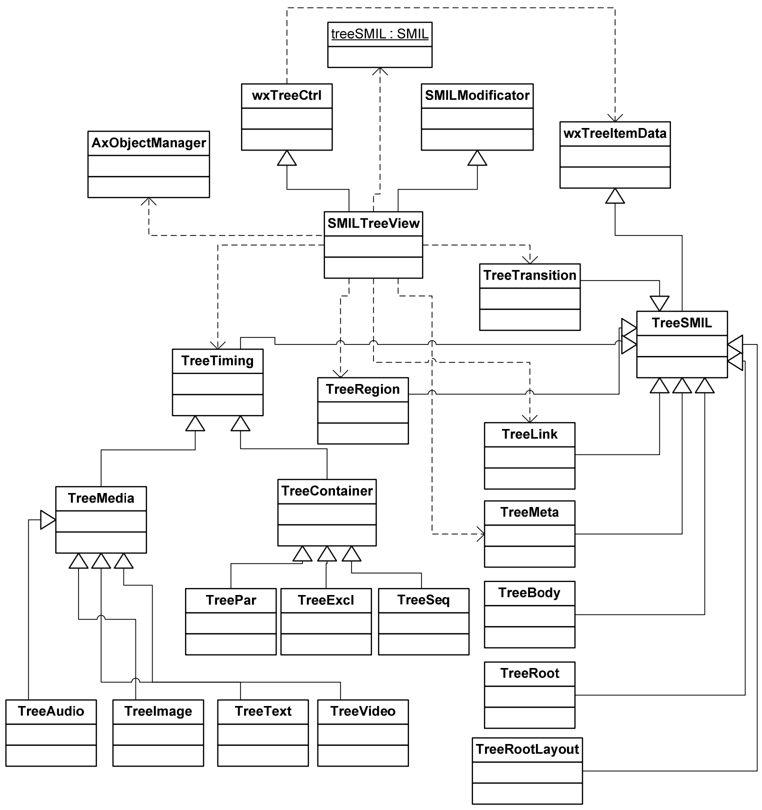
```
<body>
      <par>
            <seq>
               <img id="lamp-picture" region="content-image"
          src="LITEdataCE/lamp1.jpg"/>
               <audio id="Welcome-US" region="LogoRegion"
          src="LITEdataCE/Welcome-US.mp3"  dur="21.4s"/>


            </seq>

          <audio id="lamp-audio" region="content-audio"
          src="LITEdataCE/lamp1.mp3"/>
     </par>
 </body>
```

Or

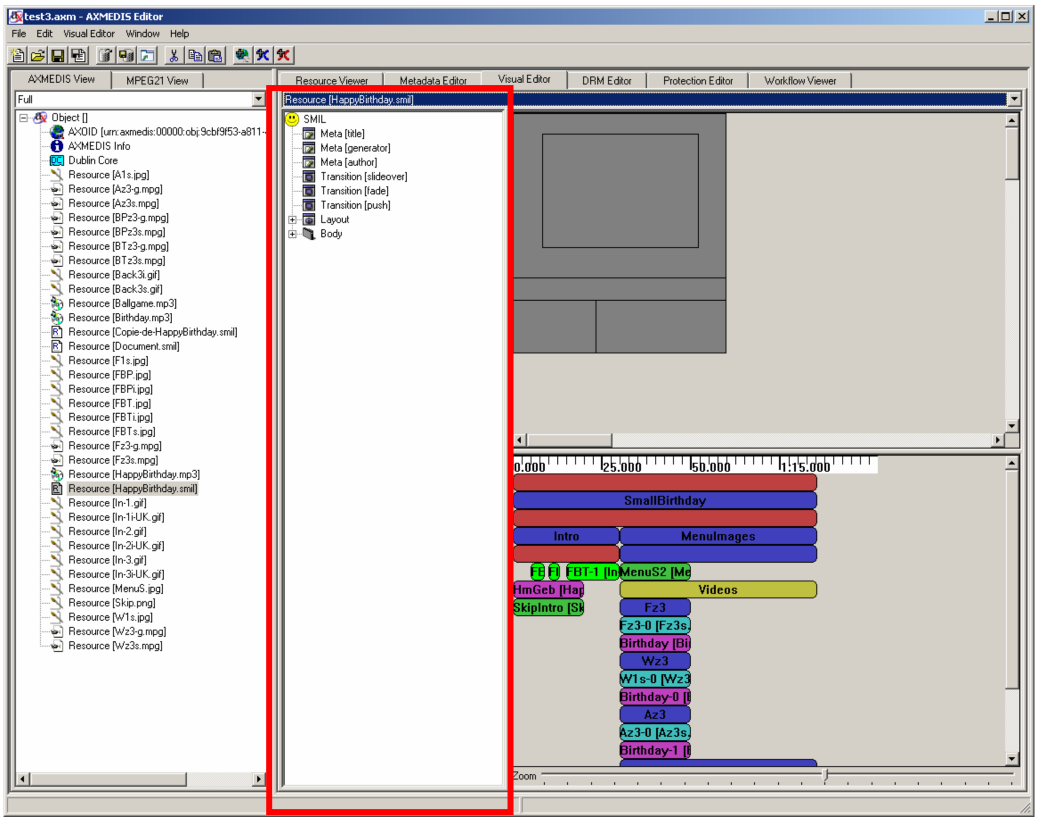
```
<body>
      <par>
            <par dur="30s">
               <img id="lamp-picture" region="content-image"
          src="LITEdataCE/lamp1.jpg"/>
               <audio id="Welcome-US" region="LogoRegion"
```

```
        src="LITEdataCE/Welcome-US.mp3"  dur="21.4s"/>

           </par>
           <seq>
              <img id="picture" region="content" src="LITEdataCE/lamp1.jpg"/>
              <audio id="Welcome" region="LogoRegion" src="LITEdataCE/Welcome-
        US.mp3"  dur="2s"/>
        </seq>

    </par>
 </body>
```

Module design:



Interface design: the tree part is located at the left of the SMIL Editor.

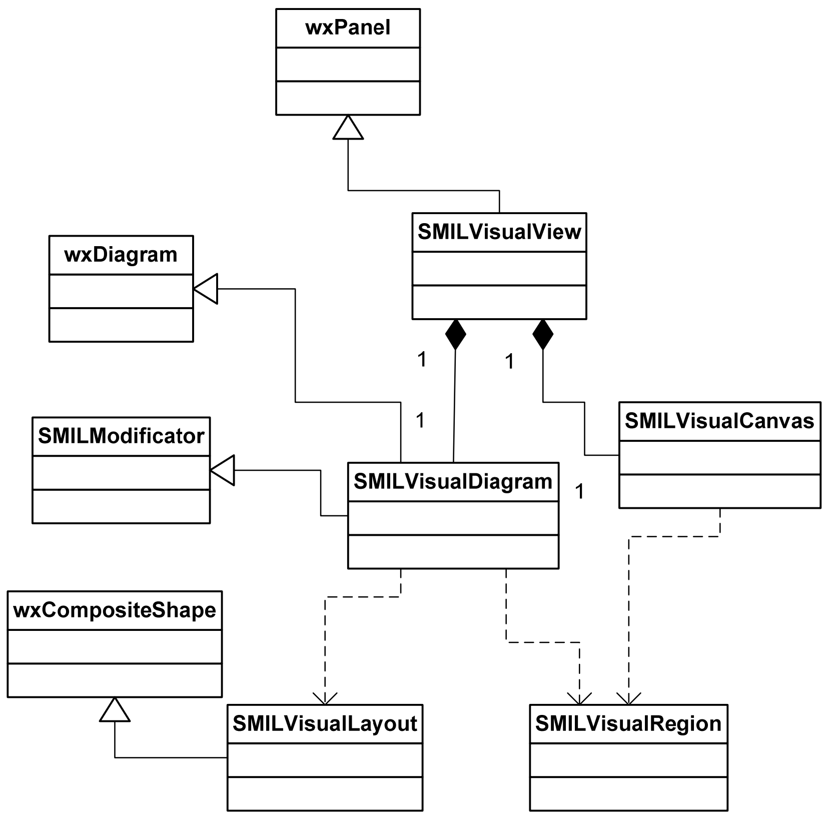


7.2.4 Visual View part

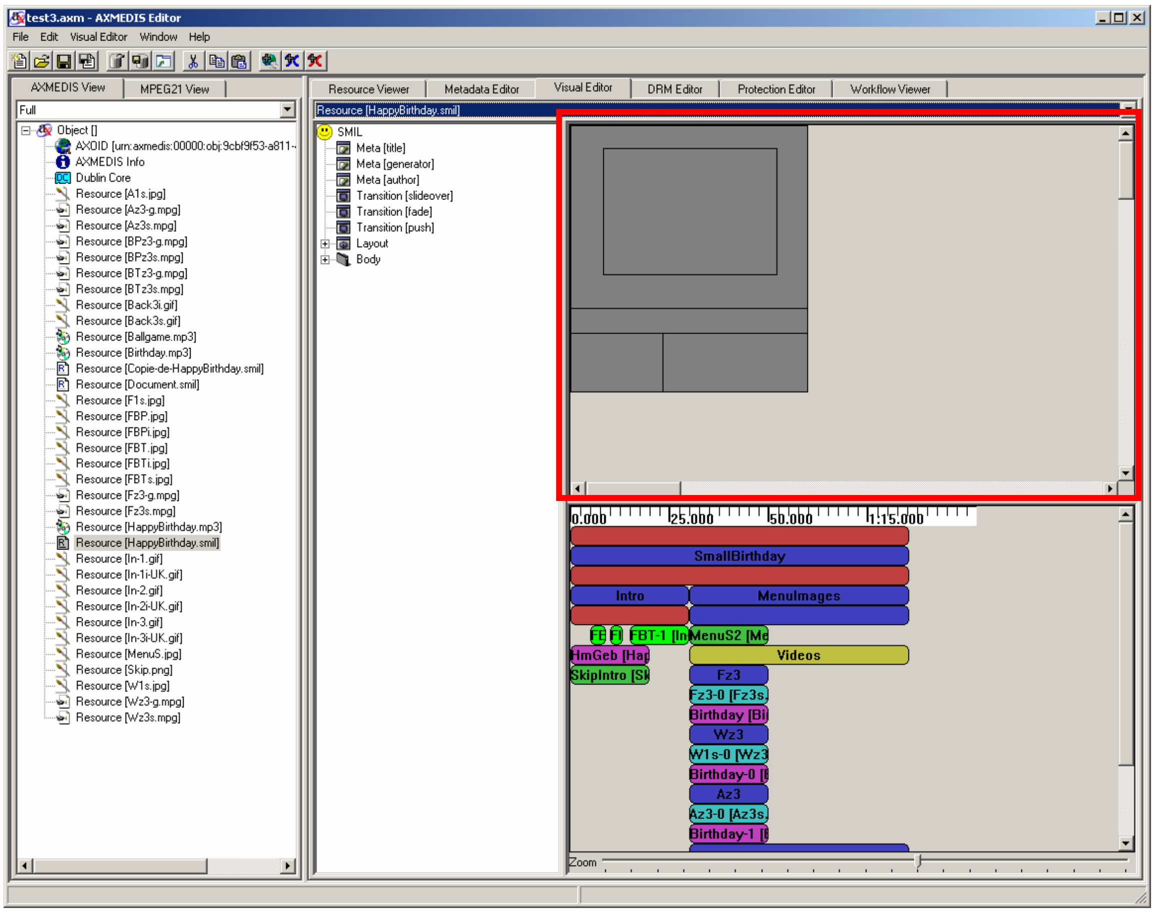
Visual View shall show object placement in a 2-D (or possibly 3-D) environment.

```
<?xml version="1.0" encoding="UTF-8"?>
<smil>
<head>
  <layout>
    <!-- The Visual Editor will handle this part: 2D layout-->
  </layout>
</head>
<body>
  <!-- The Behavour Editor will handle this part: time scheduling-->
</body>
</smil>
```

Module Design:

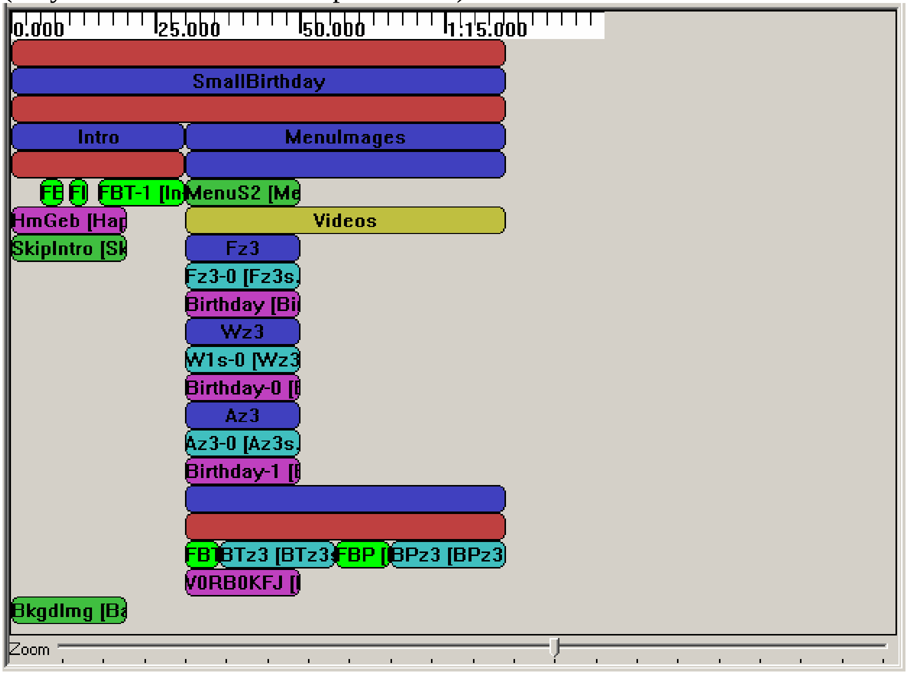


Interface Design: the visual part is located at the right top of the SMIL Editor. The visual view part of the SMIL editor is used for manipulating the SMIL layout which including the Rootlayout, Regions, Sub-regions of regions.

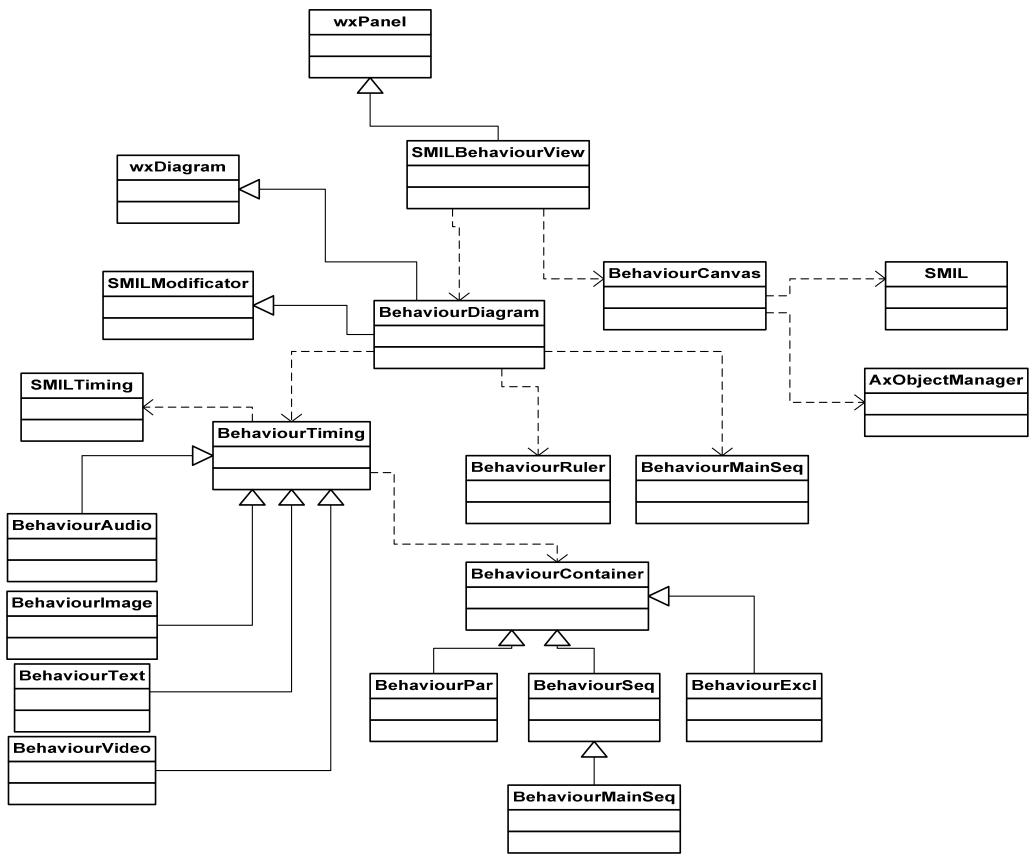


7.2.5 Behaviour view part

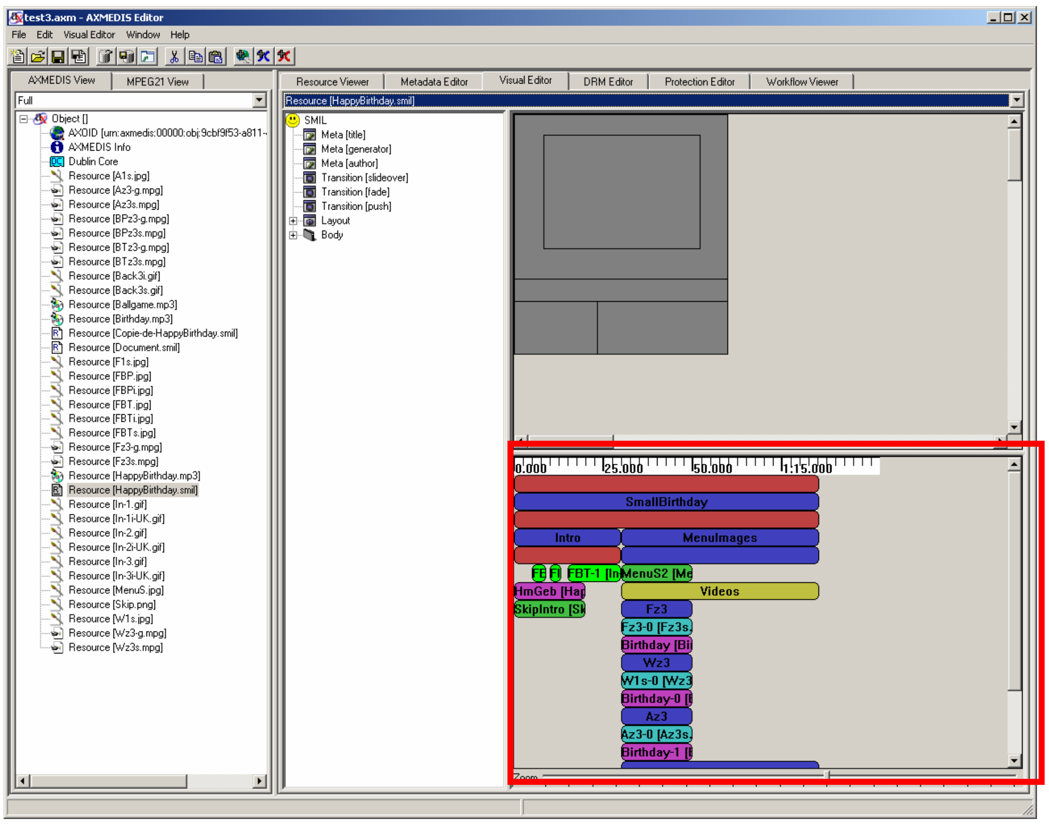
The Behaviour view part of the editor is to modify the timing attributes of the SMIL components. To be specific, you can add/modify/delete others containers like “Par” for parallel playing (all the contained together), “Seq” for Sequential playing (all the contained one after another) and “Excl” for Exclusion playing (only one of the contained upon a time).



module design:



Interface design: the behaviour part is located at the right bottom of the SMIL Editor.



8 Metadata Editor and Viewer (UNIVLEEDS)

The AXMEDIS Metadata Editor and Viewer aim to adequately display and edit metadata information contained within an AXMEDIS object. Due to the complex nature of AXMEDIS objects, there may be one or more metadata sections with different schemas, including MPEG-21, Dublin Core, and AxInfo.

- Metadata Viewer shall be able to adapt itself (e.g. by analysing the data-related XML schema) automatically to the metadata structure;
- Specific set of valuable metadata, such as authoring MPEG-7 metadata, should be included into AXMEDIS Editor basic release;

For Metadata visualisation, two possible approaches can be done: (1) using the above editor without the manipulation and save functionality activated; and (2) generating a HTML file (with CSS) and use a browser to display the file

The Metadata Editor main aims to add, delete, move and edit elements for Dublin Core and AxInfo metadata with validation. This is achieved by utilising the methods and functionalities provided by Xerces and the AXMEDIS MPEG-21 compliant model. The Metadata Editor and View is integrated into the AXMEDIS Editor as described in section 6 of the report.

8.1 Metadata Editor and Viewer: State of the art

Currently there is an influx of high performance, fully compliant XML parsers including Xerces and Expat, as it is further discussed in the deliverable DE4-3-1 [12]. While there are editors and viewers available for creating, specifying and editing XML, tools are required for editing various metadata standards (e.g. Dublin Core). This is an ongoing problem currently being addressed by working groups and specifically for the Dublin Core standard, an ongoing series of workshops to address these problems are ongoing by the Dublin Core Metadata Initiative (DCMI) Workshop Series [13].

The AXMEDIS metadata editor is required to apply manipulation functionalities such as adding, deleting and moving XML elements and their attributes with schema validation. This manipulation applies not only for the Dublin Core, but AXMEDIS AxInfo and other user defined metadata for the AXMEDIS objects. This requires the development of generic functionality that can be applied to multiple metadata standards including the user defined metadata. The metadata editor and viewer will utilise the new tools developed for the transcoding of metadata as discussed in DE4-3-1 and will also utilise the functionalities provided by the Xerces [14] libraries. Future development utilises the new AXMEDIS MPEG-21 compliant model as discussed in section 3, AXMEDIS Object Manager.

8.2 Metadata Editor and Viewer: Work performed

The work on the Metadata Editor and Viewer has been started with metadata viewing, mapping and editing functionalities near completion for demonstration. Testing is still performed on generic metadata and the functionalities for metadata mapping are to be enhanced and improved.. The main stages of the Metadata Viewer and Editor are as follows:

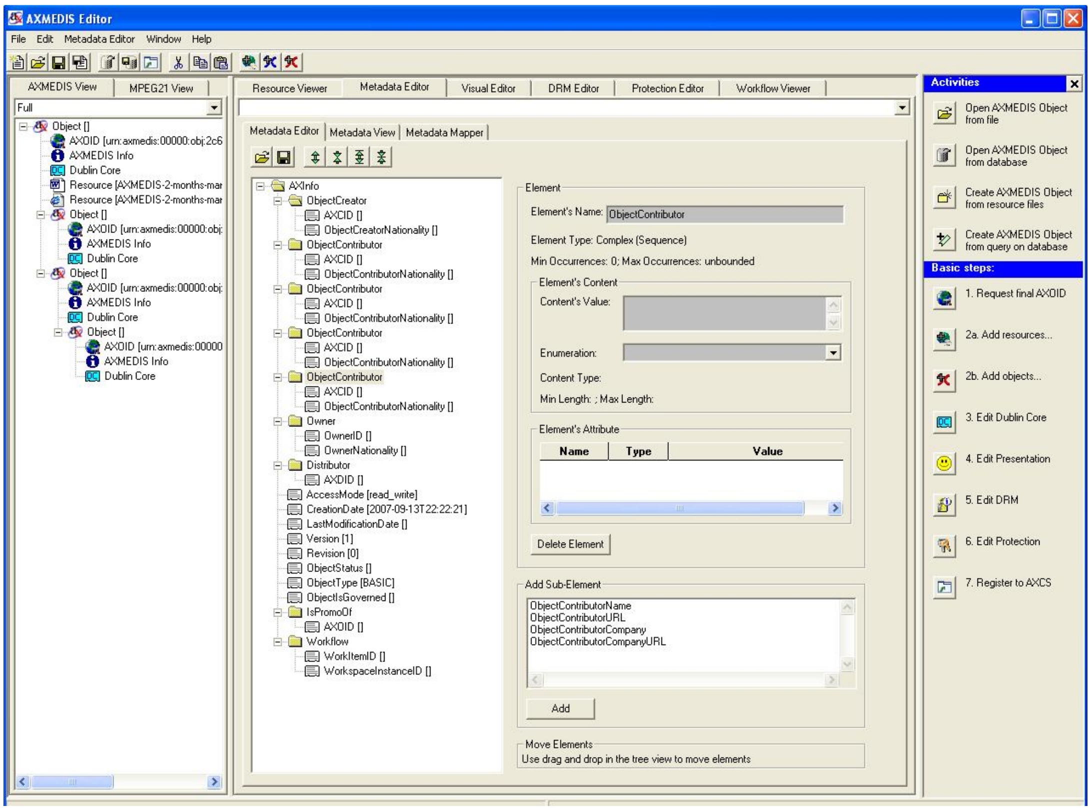
- Visualize the XML as a tree structure
- Loading and saving of the metadata XML file using the library Xerces-C++
- Parsing XML and validating using Xerces Grammar loading from the XML Schema
- GUI functionalities including local and global expansion of the tree view
- Adding, inserting and deleting metadata elements with validation using XML schemas
- Providing users with valid metadata editing options within the Metadata GUI
- Adding, inserting and deleting metadata attributes with validation using XML schemas
- Integrate Metadata Editor and Viewer into the AXMEDIS Editor
- Creating a HTML view of the metadata elements for viewing object metadata (integrated AXMEDIS document viewer) using CSS for user specified style for the metadata view.
- Automatically committing Metadata Changes into the AXMEDIS object

- Importing generic metadata with schema validation with user specified XML schemas
- Integration of Metadata Mapper GUI into the Metadata Editor Viewer to map metadata elements for the transcoding/transformation of metadata

Planned future enhancements include:

- Filtering viewing elements by namespace or schema identification
- Creating generic metadata from scratch with a user specified XML schemas

In the following image the AXMEDIS Editor Metadata View is shown. The tree view provides functionalities to edit and move metadata elements and the panel are used to edit metadata elements and add values to the appropriate leaf elements e.g. title.



Element's Name: ObjectContributor

Element Type: Complex (Sequence)

Min Occurrences: 0; Max Occurrences: unbounded

Content Type:

Min Length: ; Max Length:

ObjectContributorName
ObjectContributorURL
ObjectContributorCompany
ObjectContributorCompanyURL

Use drag and drop in the tree view to move elements

Figure 10 – AXMEDIS Editor with the Metadata View Editor panel to edit AxInfo.

In the following image the Metadata Elements are added using the tree menu. Validated items from the associated schema are used to generate the element list for both adding and insertion of new metadata elements. The left image demonstrates editing metadata using the Metadata Editor tree menu. Functionalities include adding, inserting and deleting elements. The right image demonstrated editing the metadata using the Metadata Editor Panel. Functionalities include adding (multiple elements can be added), inserting and deleting elements as well as editing the text and attributes fields for the leaf element nodes e.g. title.

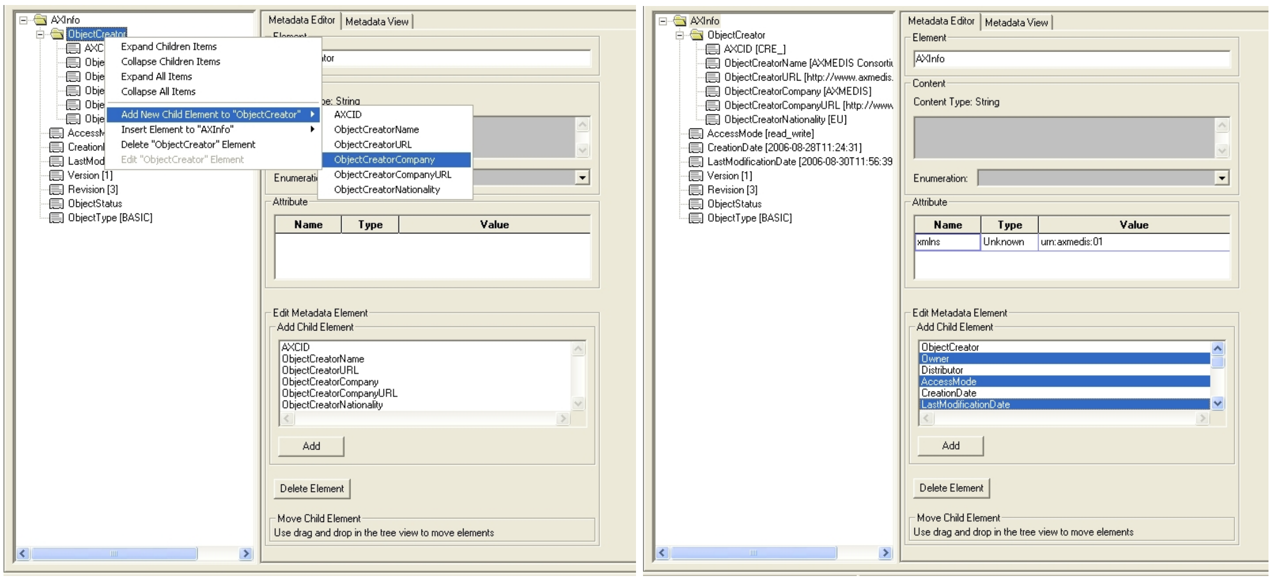


Figure 10a – AXMEDIS Editor with the Metadata Editor "Metadata Edit tab" containing the AxInfo metadata.

The image below shows the AXMEDIS Editor viewing the AXMEDIS Info in the Metadata View. This view is generated HTML from the XML and the user can use their own stylesheet to format the view within the window

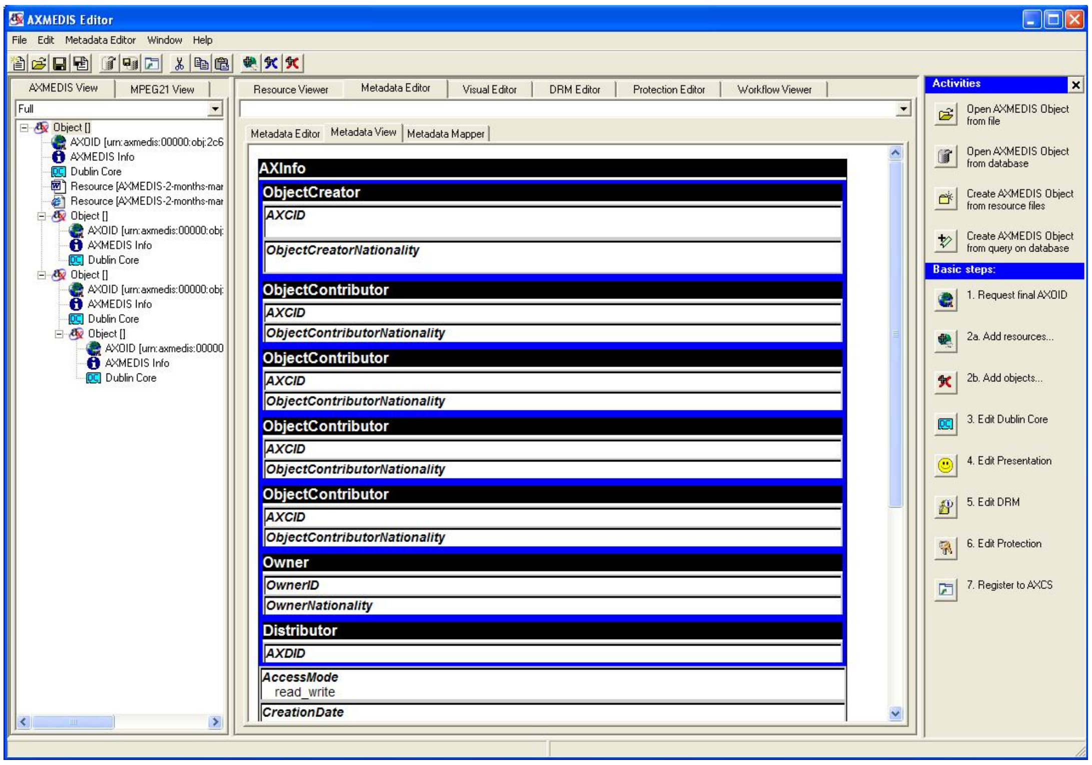


Figure 11 – AXMEDIS Editor with the Metadata Editor "Metadata View tab" presenting the AxInfo metadata.

The axdefault.css has the following content style:

```
.level0 {
   background-color: white;
   border: 1px solid black;
   position: absolute;
```

```
    padding:0px;
    margin: 0 auto;
    width: 95%;

}

.level1 {
  background-color: blue;
  padding: 4px;
}

.level2 {background-color:blue; padding:4px}

.level3 {background-color:green; padding:4px}

.leaf {
  background-color: #cccccc;
  padding: 2px
}

.elementname {
  font-family:sans-serif, helvetica, verdana, geneva,arial;
  background-color:#000000;
  border-bottom: 3px solid #ffffff;
  font-weight: bold;
  font-size: 100%;
  color: #ffffff;
}

.leafname {
  font-family:sans-serif, helvetica, verdana, geneva,arial;
  background-color:#ffffff;
  font-style: italic;
  font-weight: bold;
  font-size: 90%;
  border-top: 2px solid #000000;
  border-left: 1px solid #000000;
}

.leafcontent {
  font-family:sans-serif, helvetica, verdana, geneva,arial;
  background-color:#ffffff;
  font-size: 90%;
  border-left: 1px solid #000000;
  padding-left: 1em;
}
```

The image below shows the AXMEDIS Editor viewing the AXMEDIS Metadata (AxInfo, Dublin Core and Generic metadata) in the Metadata Mapper View. The Metadata Mapper tab provides two tree views for mapping data from one metadata element to another. The Metadata Mapper provides the option to save an XSLT which can be utilised within the AXMEDIS Content Processing (AXCP) area and the metadata adaptation algorithms available for the automatic translation/transcoding of AXMEDIS metadata. The Mapper view can also be used to save the AXMEDIS object with the transformed metadata.

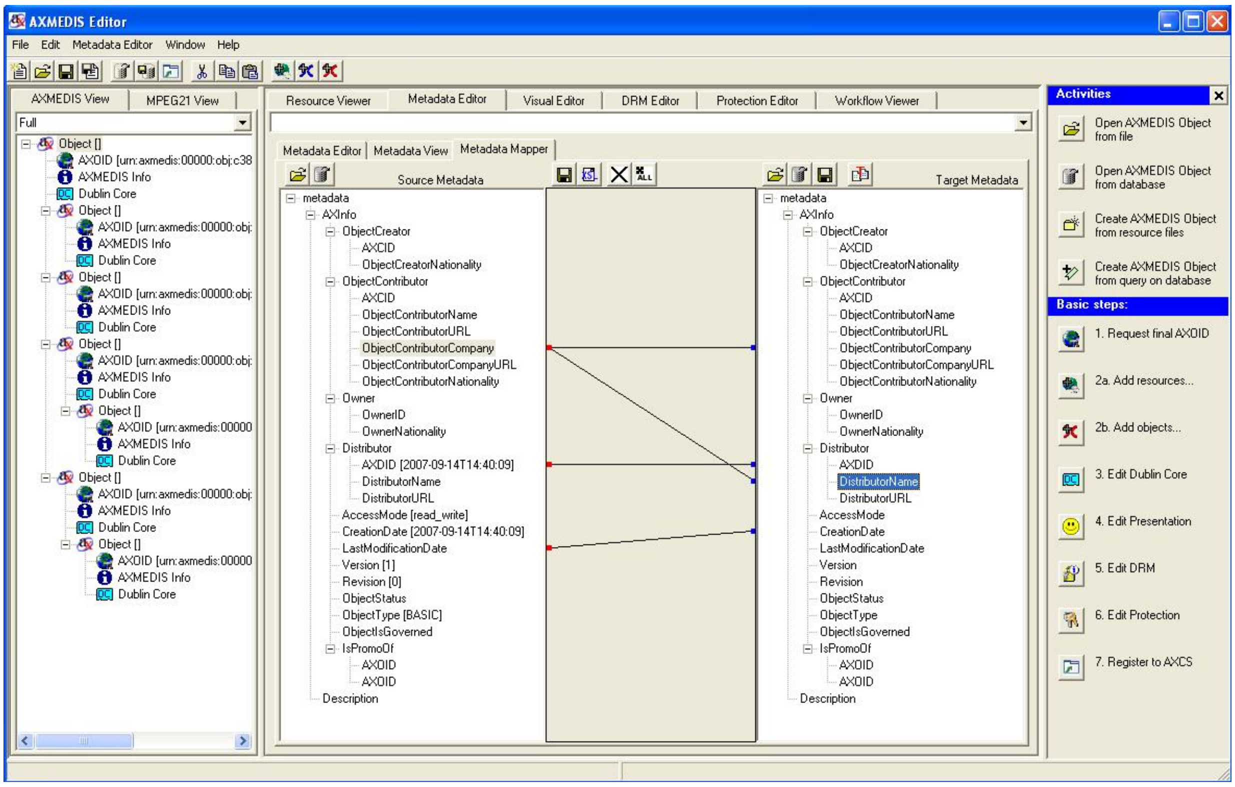


Figure 11a – AXMEDIS Editor with the Metadata Editor "Metadata View tab" presenting the AXMEDIS public metadata and demonstrates the linking of elements to map the metadata.

8.3 Metadata Editor and Viewer: problems and actions

Some of the desired functionality has not been implemented:

- User defined filtering of the tree structure
- Problems editing MPEG7 metadata with schema

Cannot create generic metadata from existing schemas, only import generic metadata

9 AXMEDIS ActiveX Control as editor (DSI)

The AXMEDIS ActiveX control allows exploiting AXMEDIS objects in windows applications and in web pages.

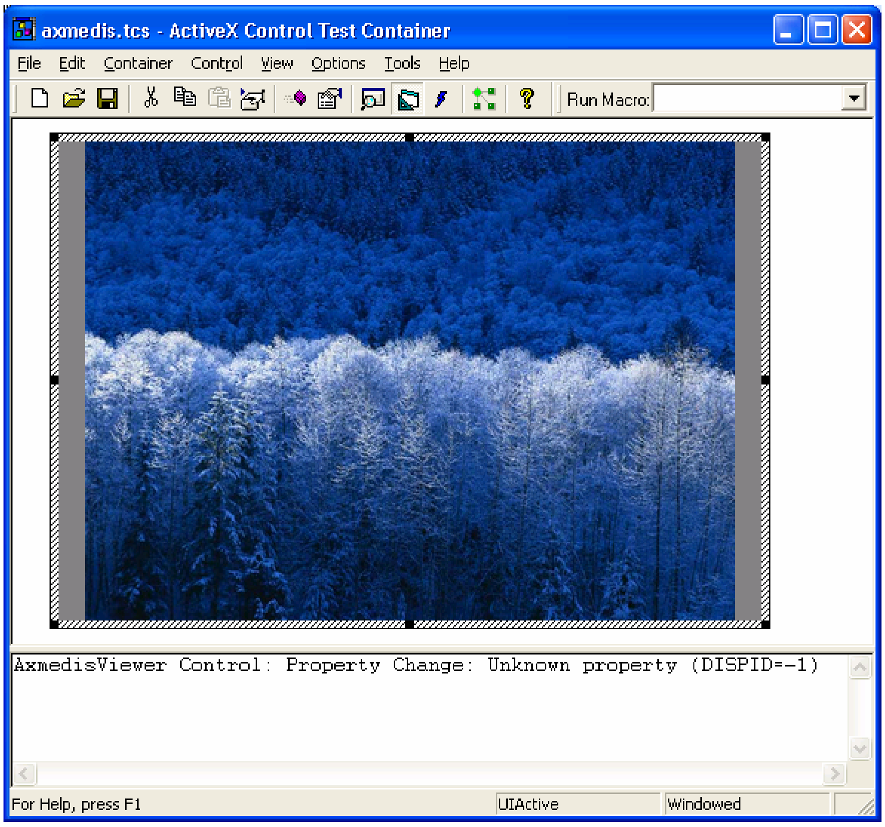
9.1 ActiveX Control: Work performed

The initial prototype of AXMEDIS ActiveX Control allows to:

- load an AXMEDIS object from file (.axm or .mp21) or from url (http)
- display the AXMEDIS hierarchy (for end user) and the resources in the object
- hide/show the hierarchy
- get the content count (how many resources/objects are present in the object)
- open a content in the resource view
- get some minimal information on the content
- control the execution of the content play/stop
- view metadata information
- import user certificate

- certify the activex to allow using protected content

The following picture shows the ActiveX used in the ActiveX Control Test Container


axmedis.tcs - ActiveX Control Test Container
File Edit Container Control View Options Tools Help
Run Macro:

```
AxmedisViewer Control: Property Change: Unknown property (DISPID=-1)
```

For Help, press F1
UIActive
Windowed

Figure 11 - ActiveX Control Test Container

See the AXMEDIS PC Player prototype as another example of AXMEDIS ActiveX use.

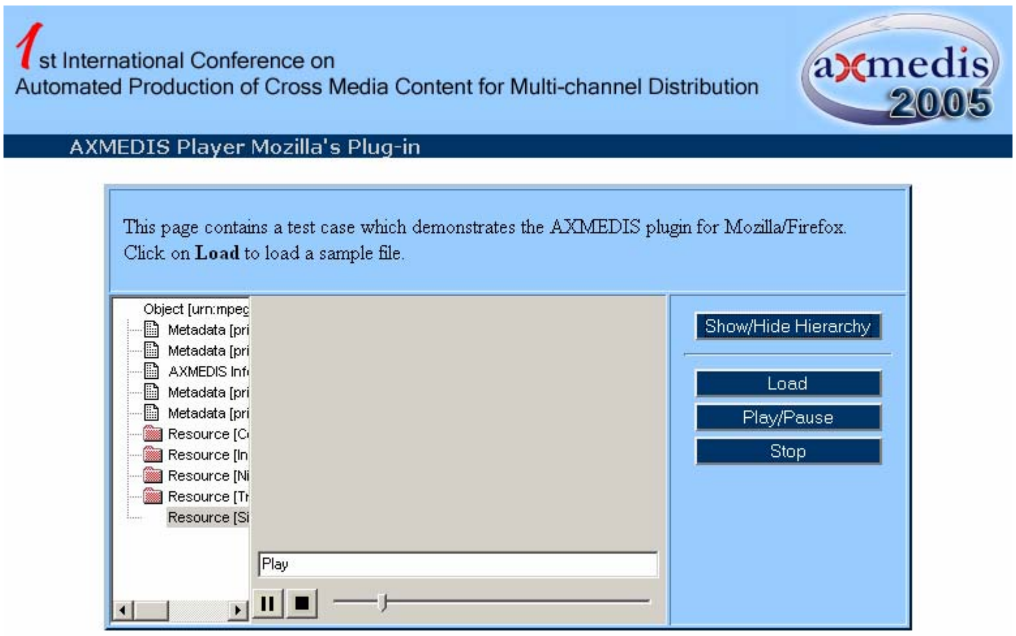
9.2 ActiveX Control: problems and actions

Some of the specified functionality has not been implemented:

- licence display and control

10 AXMEDIS Plug-in into Mozilla (SEJER)

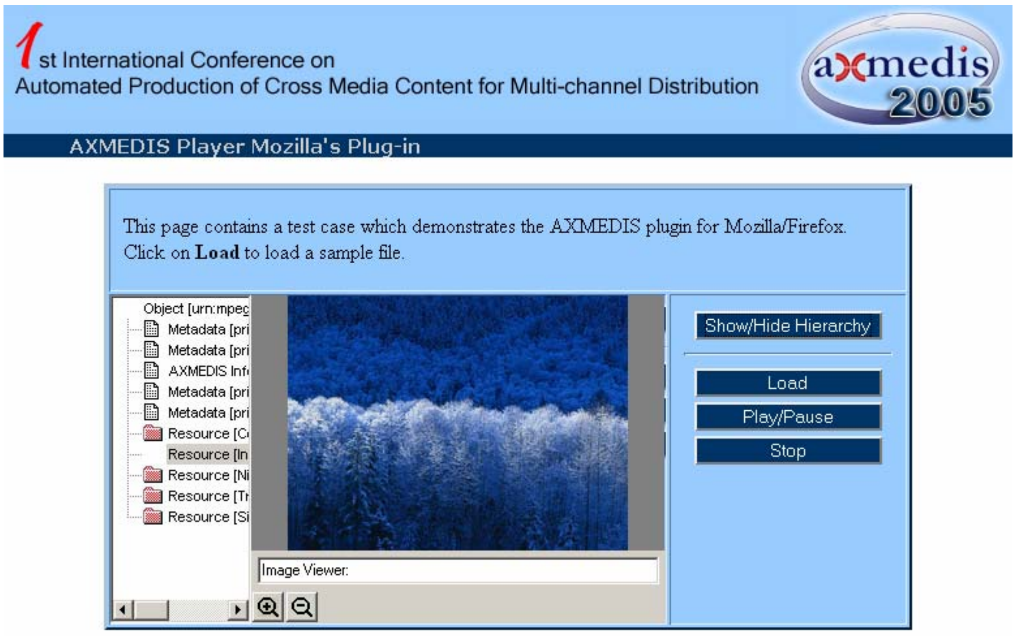
The AXMEDIS plug-in into Mozilla [15] is intended to integrate the AXMEDIS player inside HTML pages. The three screenshots below demonstrate the rendering of several contents, with the hierarchy visible or not.


1st International Conference on
Automated Production of Cross Media Content for Multi-channel Distribution

AXMEDIS Player Mozilla's Plug-in

This page contains a test case which demonstrates the AXMEDIS plugin for Mozilla/Firefox.
Click on **Load** to load a sample file.

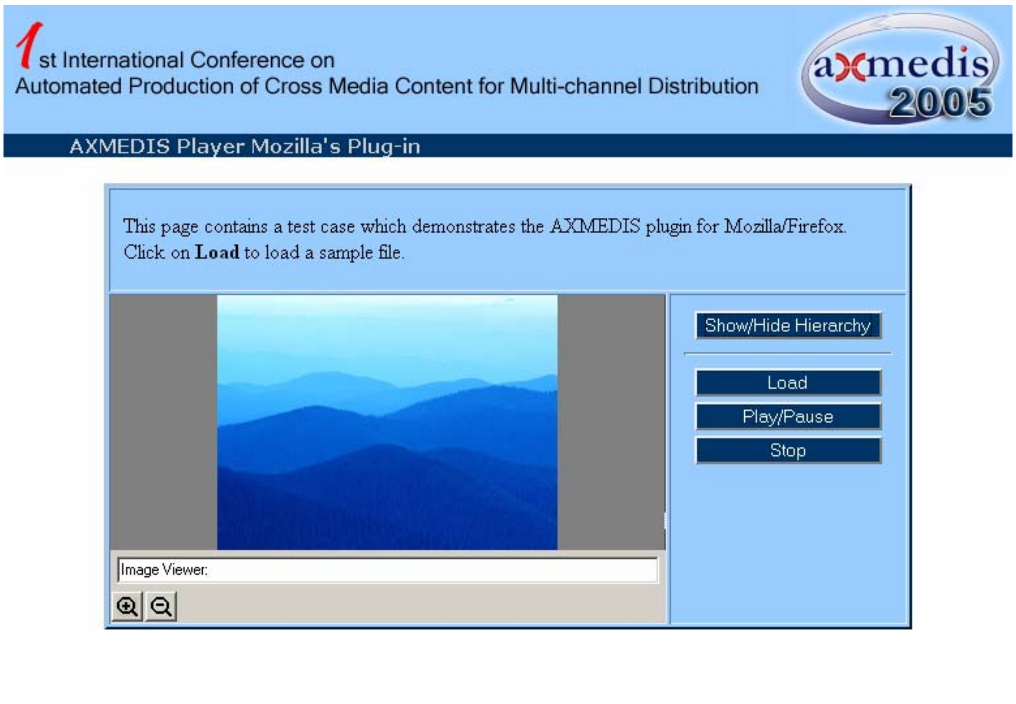
Playing music


1st International Conference on
Automated Production of Cross Media Content for Multi-channel Distribution

AXMEDIS Player Mozilla's Plug-in

This page contains a test case which demonstrates the AXMEDIS plugin for Mozilla/Firefox.
Click on **Load** to load a sample file.

Displaying a picture


1st International Conference on
Automated Production of Cross Media Content for Multi-channel Distribution
axmedis 2005
AXMEDIS Player Mozilla's Plug-in
Show/Hide Hierarchy
Load
Play/Pause
Stop
Image Viewer:

This page contains a test case which demonstrates the AXMEDIS plugin for Mozilla/Firefox.
Click on **Load** to load a sample file.

Displaying a picture with the Hierarchy panel hidden

Figure 12 - Rendering of several contents into Mozilla

The plug-in renders AXMEDIS objects, as the AxPlayer does, and exposes some methods to the JavaScript. Web developers can easily embed the player in HTML pages and use DHTML functionalities to control the player.
The JavaScript exposes properties and methods for navigation, media time control, media visual control, support and generic command management.

10.1 Plug-in into Mozilla: Work performed

The work on the AXMEDIS plug-in into Mozilla has been started and is almost in sync with the ActiveX Control.
The following properties and methods have been implemented:

```
/* behaviour methods and attributes */
attribute boolean viewHierarchy;

/* loading and content methods and attributes */
readonly attribute short contentCount;
attribute string src;

short load(in string filename);
string getContentType(in short contentRef);
void openContent(in short contentRef);

void play();

/* media time control methods and attributes */
void stop();
```

The plug-in is also able to read the value of the viewHierarchy and the path to the file to load from the parameters in the HTML page.

10.2 Plug-in into Mozilla: problems and actions

The code of the plug-in is based on the sample “scriptable” from GeckoPluginSDK. The first problem was to have mozilla/firefox to recognize the plug-in, even after the library has been copied into the plugin directorory of the application. In order to do that, the library name **must** start with “np”

Then, when the code from the AxActiveX was inserted into the plug-in, three problems arise:

- the resources of the player were not displayed correctly. The HINSTANCE of the application has to be set to the one of the library;
- the initial layout of the controls composing the player is not correct until the user resizes the plug-in. This was corrected by forcing the plug-in to layout at the end of the initialization;
- the background of the player was not drawn correctly. Ignoring the `WM_ERASEBKGND` solved the problem.

The plug-in should be able to raise events that JavaScript code in the HTML page should listen to. Little documentation is available for this specific task. A solution was found by using nsIDOMEventListener an implementing nsIDOMEventTarget.

The main difficulty about the plug-in is the ever changing interfaces of mozilla and the fact that it is dependend of the version of the instance it is running in, as the Mozilla’ dlls are already loaded and that the plug-in cannot specify them.

Firefow 2.0 will be a challenge as some interfaces are changing and very few documentation about the changes is available.
This will be quite tedious for the document viewer based on mozilla.
Currently, it is very difficult to recompile wxMozilla for FireFox 1.5, as some of the interfaces or classes are marked obsolete or are now simply removed. So far, we were able to recompile wxMozilla but not to link a simple plugin with it.

The following methods and properties have to be implemented:

```
/* behaviour methods and attributes */
attribute string backgroundColor;

/* navigation methods and attributes */
void goNext();
void goPrev();

/* media time control methods and attributes */
void pause();
readonly attribute boolean isPlaying;
void jumpToTime(in long time);
readonly attribute long currentTime;
readonly attribute long duration;
attribute long startTime;
attribute long endTime;

/* media visual control methods and attributes */
void fit(in double width, in double height);
void zoomIn(in double ratio);
void zoomOut(in double ratio);
attribute boolean fullScreen;
void print();
void scale();
attribute double zoom;
attribute boolean autoFit;
```

```
/* support methods and attribute */
readonly attribute boolean supportTimeControl;
readonly attribute boolean supportVisualControl;
void showLicense();
readonly attribute boolean needLicense;
readonly attribute boolean haveLicense;
void acquireLicense();
void showMetadata(in long ref);

/* command management */
boolean execCommand(in string cmdID, in boolean showUI, in string value);
boolean queryCommandEnabled(in string cmdID);
boolean queryCommandIndeterm(in string cmdID);
boolean queryCommandStatus(in string cmdID);
boolean queryCommandSupported(in string cmdID);
boolean queryCommandValue(in string cmdID, out string pCmdValue);
```

11 AXMEDIS plug-in into Multimedia Players (DSI)

In this area several experiments have been performed on Windows Media Player trying to customize the default behaviour of the player in order to:

- enforce AXMEDIS digital rights;
- render multimedia content packaged in an AXMEDIS object.

For more details on Customizing Windows Media Player see Report in Appendix (file Customizing-WMPlayer.pdf included with this deliverable).

11.1 Plug-in into Multimedia Players: Work performed

Several experimentations have been performed with the Windows Media Technologies: a prototype of customized Windows Media Player has been realized. The customization has been focused on the digital rights enforcement and on AXMEDIS object decoding and rendering on the WM Player.

11.2 Plug-in into Multimedia Players: problems and actions

Two main issues have been identified in examining the Windows Media Technology:

- It is not possible to control play of resource in terms of avoiding unauthorized operations. No customization are possible in terms of DRM.
- It is possible to create a render plug-in and a UI plug-in, but is not possible to demand play of well-know resources as soon as they have been extracted by the composed AXMEDIS object. Only a device canvas is at disposal to be drawn by the render plug-in.

12 AXMEDIS PC Player state of the art (DSI, EPFL)

The AXMEDIS PC Player application allows opening and rendering AXMEDIS objects in the end user terminal. The AXMEDIS player will use the AXMEDIS internal Players to render the different types of content: audio, video, text, and SMIL presentations.

In these terms, given the nature of an AXMEDIS Object, the Player shows a strong relationship and interconnection with the concept of MPEG-21 Client Terminal.

A Visual Basic application has been realized to test the ActiveX, it could be considered as a preliminary prototype of the PC Player.

Furthermore, the Internal SMIL Player (see DE5.1.1, AXMEDIS Framework Infrastructure) can at the same time be exploited to synchronize the several media objects according to precise spatial and temporal stamps, and as such constitutes the core of the scheduling and playback functionality of the AXMEDIS Player.

12.1 PC Player: State of the art

Other multimedia tools, mainly based on proprietary formats, are currently spreading on the market. They are based on frameworks that are not as flexible and powerful as MPEG-21, however they constitute a state

of the art in the domain, since as noticed above MPEG-21 is still in development stage and current tools are poor. Reviewing these frameworks is important as it provides a direct investigation about what currently means a state of the art player terminal in terms of media support, security, flexibility and other aspects; this is useful to learn about successful features of client terminal players for PC platforms and also to show how much advanced the MPEG-21 based AXMEDIS client will result.
We review here the main multimedia frameworks in their actual state. We do not speculate about their future, or about their possible evolutions. The landscape is likely to evolve quickly, some of the main actors will probably disappear, and new actors will also appear.

12.1.1 Flash

Flash is a proprietary system developed by Macromedia, Inc. [22]. It is composed of an authoring tool, and a viewer available for free on the most widely available platforms (Macintosh and Windows based). The viewer can be integrated in an HTML page, so Flash content can be easily integrated in Web content.

Flash is based on vector graphics. In conjunction with scripting (Actionscript), and animation (timeline based as well as scripting based), this has made of Flash the most widely used multimedia framework for the Web. The very low bit rate induced by the use of vector graphics, scripting, and timeline based animation has made of Flash a very convenient format for Web-based animation, storyboards, high quality graphics, and a suitable alternative to HTML for the development of Internet web sites.

Media support

The support of audio can be considered as being poor in Flash. The only compression scheme supported is mp3, together with a proprietary compression scheme known as “Nelly Mosser” for which no information is available to our knowledge. Flash supports also uncompressed schemes, PCM based, such as WAV or AIFF. For structured audio, no format – even MIDI - is supported. Video support in Flash has continued to evolve since its introduction in Flash MX and Flash Player 6. Flash Player 7 greatly improves video quality, supports higher frame rates, and provides additional opportunities for loading dynamic media at runtime.

System support

Flash supports the operating systems of Windows and MacOS. It cannot support the Unix, Linux, BSD system.

Authoring and production

Flash benefits from a proprietary authoring tool developed by Macromedia.

Interactivity and animation

Interactivity and animation can be implemented in Flash by using ActionScript, a proprietary scripting language

Openness and extensibility

Flash is in principle a closed system on the client’s side. No extensions can be developed, no decoders can be added, and no interactivity other than interactivity defined on the authoring side with the ActionScript scripting language can be defined to enhance the standard viewer (this is not to be confused with the extensibility functions available in the new Flash MX 2004, which are available in the Flash authoring application).

Flash is open to XML, and able to exploit XML data in a client-server architecture, via http-based protocol, or via XML socket based, real-time exploitation of data. With this functionality, it is for example possible to imagine a Flash client application exploiting XML data available on line, for example XML-based metadata such as RDF, or Dublin Core, or even MPEG-7 metadata in their XML format.

The Flash file format is itself now open, as well as some parts of the source code, and many developers are developing new Flash based solutions. For example, the NorthCode company [23] has developed SWFStudio (http://www.northcode.com/swfstudio/), a software which makes possible to build stand-alone executables from Flash content. In this configuration, it becomes possible to build plugins to Flash executables. The same society has developed a plugin development kit in order for other developers to build their own extensions to Flash (with the restriction that this works only with Flash stand alone applications – it is always impossible with the Flash standard client).

Security and privacy

Flash implements a browser-like security sandbox scheme in order to ensure the security and privacy of Flash movie and the client machine. The sandbox defines a limited space in which a Flash movie running within the Macromedia Flash Player is allowed to operate. Its primary purpose is to ensure the integrity and security of the client's machine, and as well as security of any Macromedia Flash movies running in the player. Basically, the sandbox idea is the following: A Macromedia Flash movie executes inside a sandbox. Any information inside the sandbox can be communicated only to the domain from which the movie came. Access to information within the sandbox from outside of the sandbox is severely limited.

12.1.2 Windows Media

Being preinstalled with every version of Microsoft Windows sold, Windows Media Player is becoming increasingly widespread on the web.

Media support

Windows Media Player supports most of media formats, but it does not support the RealNetworks content such as the .ra, .rm, .ram media file. It cannot support the QuickTime content like .mov, .qt format. As for the MPEG-4 (.mp4), Windows Media Player gives no support.

System support

Windows Media Player supports the operating systems of Windows and MacOS. It cannot support the UNIX, Linux, BSD system.

Authoring and production

Production of Windows Media content can be done in multiple ways: by the mean of Windows Media Encoder, or by the mean of the toolkits provided by Microsoft for this purpose. This toolkit can be accessed by the mean of the C++ language, the Visual Basic language, or even by the mean of an HTML interface.

Interactivity and animation

No support of interactivity – scripting, controls – is directly available in Windows Media.

Openness and extensibility

Customization of the Windows Media Player is possible by using the Software development Kit provided to this end by Microsoft. By using the SDK, it is possible to develop a customized end-user interface driving the Windows Media content, in any language supported by the Windows Media SDK (C++, Visual Basic, HTML, .net with C#…).

Security and privacy

Windows media frameworks implement the Windows Media digital rights management (DRM) platform to protect and securely deliver a la carte and subscription content for playback on a computer, portable device, or network device. Windows Media DRM is comprised of multiple components including Windows Media DRM for Portable Devices and Windows Media DRM for Network Devices, as well as an updated Windows Media Rights Manager SDK. These components allow for the seamless flow of content to almost any device, offer the widest range of purchase and rental options for digital media, and ensure the security of premium content as it flows from device to device. Windows Media DRM works in the following five steps: 1.Packaging 2.Distribution 3.Establishing a license server 4.License acquisition 5.Playing the digital media file to secure content providers to protect their content and maintain control over the entire process of the media distribution.

12.1.3 QuickTime

Apple's products

They are the following:

- QuickTime 7 Pro [24] enables H.264 video creation, audio and video capture, multi-channel audio creation and multiple files export. It is an easy-to-use tool for creating AAC audio files and 3G files for mobile viewing, editing videos and exporting movies.
- QuickTime MPEG-2 Playback Component provides QuickTime users with the ability to import and play back MPEG-2 content, including both multiplexed and non-multiplexed streams. It is suited for content creators with projects such as Professional content production and transcoding video content (from MPEG-2 video to MPEG-4 for example).
- QuickTime Broadcaster is a tool for producing live broadcast events.

Media support

QuickTime 7 Player supports a wide-range of industry-standard audio formats, including AIFF, WAV, MOV, mp3, MP4 (AAC only), CAF and AAC/ADTS. For structured audio, QuickTime supports MIDI. There is no support for audio effects or 3D audio. Multichannel audio is supported by QuickTime 7 up to 24 audio channels, enabling standard surround formats. QuickTime 7 supports H264.

System support

QuickTime supports the operating systems of Windows and MacOS. It cannot support the UNIX, Linux, BSD system.

Authoring and production

Adobe's Premiere or Macromedia Director can generate QuickTime content. There are also a number of production tools available, like FinalCut Pro for instance. Apple does not provide authoring tools, but software like Adobe's Premiere or Macromedia Director are able to produce QuickTime content, generally by the mean of a plug-in. Apple's QuickTime-related products (QuickTime Pro, QuickTime MPEG-2 Playback and QuickTime Broadcaster) are not literally authoring tools, but provide however a few creating, encoding and editing functionalities.

Interactivity, animation

No scripting language is available for defining interactivity, but interactivity can be defined by using the QuickTime Software Development Kit provided by Apple.

Openness, extensibility

Extensions to QuickTime can be defined on the user's side by using the QuickTime Software Development Kit provided by Apple. It provides interfaces in C or Java QuickTime content can be embedded in a web page, but only a restricted set of functions are available from scripting languages such as JavaScript, making QuickTime not very well suitable for development of interactive content on the Web. Timeline-based, raw graphics animation is provided by authoring tools such as Adobe Premiere or Macromedia Director.

Security and privacy: tbd

12.1.4 Real

RealNetworks media are limited to audio, from speech, mono-channel to surround, channel music, and video. There is no native support for interactivity, vector graphics, but the Real Player [25] supports the W3C's standard for synchronized multimedia SMIL, and thus interactivity, animation and support of vector graphics can be integrated this way.

Media support

Real plays every major media format Including, AVI, MP3, RealAudio, RealVideo, WAV Audio, and Windows Media. This feature makes it a very popular media player.

System support

Real not only supports the operating systems of Windows and MacOS, but also support the UNIX, Linux. But it cannot support BSD system.

Authoring and production

The Helix producer enables encoding of streaming media (audio, video), in the native Real formats, with different bit rates. The Helix producer cannot generate SMIL animations – but can generate the included media.

Interactivity, animation

Real Media does not include any native support for interactivity or animation, but these functionalities can be integrated in SMIL animations by integration of Flash or SVG content.

Metadata

There is no support for metadata in Real Media.

Openness and extensibility

Real Media does not include any native support for scripting or extensibility, but the SMIL support for these features must be taken in account.

RealPlayer

- Audio: The Real Player includes support for its audio proprietary format as well as for mp3 format. It is also possible to include MIDI content.
- Vector graphics, animation: Real Player supports the integration of Flash content (only Flash 3 and Flash 4), with some restrictions such as for audio content, which must be integrated using another channel (mp3, or rm). Interaction with Flash content is also supported, enabling in this manner capabilities of interactions with timeline from the user. It is for example possible to develop a simple

user interface in Flash, composed of some buttons for playing, stopping, fast reviewing or forwarding an audio track, but this kind of interaction will be limited to interaction with the timeline.

- Real Player supports also integration of SVG.

Security

Real implements the Helix DRM platform for secure media content delivery over PC and non-PC devices, including mobile devices and home equipments. Helix DRM includes a set of products and services enabling business models through secure rights managed distribution of movies, music and other digital content. Helix DRM provides secure media packaging, license generation and content delivery to a trusted media player base across all major platforms to multiple devices. It extends the RealPlayer and Helix Platform open architecture to accommodate the incorporation of a wide range of rights management systems. It integrates into existing infrastructures and back-end systems, supporting a broad set of business models including purchase, rental, video on-demand, and subscription services.

The above media players are all developed for the multiple kinds of content files which cover over the range of document, images, audio, video, multimedia, etc. Here, we also want to enclose two famous audio players iTunes and Winamp with the purpose of learning their good features of Graphical User Interface which could be a good example for the development of AXMEDIS Player.

12.1.5 iTunes

iTunes [26] is also a digital media player developed by Apple Computer, for playing and organizing digital music and video files. Additionally, the program connects to the iTunes Music Store (sometimes referred to simply as "iTunes" or "iTMS") which allows users to purchase digital music files that can be played by iTunes. The player has gained and maintained a reputation as being easy to use while still providing many features for obtaining, organizing, and playing music. iTunes is also the principal way to manage the music on Apple's popular iPod digital audio player. The program is freely downloadable and is also supplied with Mac OS X as well as Apple's iLife home-application suite.

Media Support

iTunes can currently encode to MP3, AIFF, WAV, MPEG-4 AAC, and Apple Lossless, and can play anything QuickTime can play (even video formats, as long as they have audio). In order to play other formats such as the Ogg Vorbis audio format iTunes requires addition of QuickTime components. However, the extensions for Ogg Vorbis does not work with QuickTime version 7 and Mac OS Tiger installed. In May 2005, video support was introduced to iTunes with the release of iTunes 4.8. But Video support in iTunes is still limited at this point iTunes is so far still incompatible with the most common video formats such as .MPEG and WMV.

System support

iTunes supports Windows and MacOS. It cannot support the UNIX, Linux and BSD system.

Authoring and production

iTunes was developed from SoundJam MP, a popular commercial MP3 application distributed by the Macintosh software company Casady & Greene.

Openness and extensibility

On the Macintosh, iTunes is tightly integrated with Apple's iWork suite of applications and the rest of the applications in iLife.

Security and privacy: tbd

12.1.6 Winamp

Winamp [27] is an audio player made by Nullsoft, part of Time Warner. It is skinnable, multi-format freeware. Originally, MP3 playback was based on the AMP decoding engine by Tomislav Uzelac et al. In later versions this was replaced with Nitrane, a proprietary decoder created by Nullsoft and subject of a lawsuit from Playmedia Systems, Ltd. After an out of court settlement and licensing agreement, Nullsoft switched to an ISO decoder from Fraunhofer Gesellschaft, the developers of the MP3 format

Media Support

Winamp support a wide range of audio format file including MIDI, MOD, MP3, Ogg Vorbis, WAV, WMA and many other audio formats.

System support

Winamp only supports Windows. It cannot support MacOS, UNIX, Linux, and BSD system.

Authoring and production

Winamp was first released by Justin Frankel in 1997.

Security and privacy: tbd

The features of these above media players are summarized in the following tables.

General Features: Basic general information about the players: creator/company, license/price etc.

	Creator	First public release date	Stable version	Software license	Proprietary format
Flash	Macromedia	December 1996	8.0	Proprietary	Flash
Director	Macromedia	1988	MX 2004	Proprietary	DCR
iTunes	Apple Computer	January, 2001	4.9	Proprietary	Apple Lossless
QuickTime	Apple Computer	December, 1991	7.0.1	Proprietary	QuickTime
RealPlayer	RealNetworks	1995	10	Proprietary	RealAudio, RealVideo
Winamp	Nullsoft	June, 1997	5.094	Proprietary	NSV
Windows Media Player	Microsoft	November, 1992	10	Proprietary	WMA, WMV

Operating Systems Support: The operating systems on which the players can run natively (without emulation).

	Windows	Mac OS X	Linux	BSD	Unix

Flash	Yes	Yes	No	No	No
Director	Yes	Yes	No	No	No
iTunes	2000/XP/2003 only	Yes	No	No	No
QuickTime	Yes	Yes	No	No	No
RealPlayer	Yes	Yes	Yes	No	Yes
Winamp	Yes	No	No	No	No
Windows Media Player	Yes	Yes	Planned	No	No

Main Features:

	Audio playback	**Video playback**	**Outbound streaming**	**Skinnable**	**Media Database**
Flash	Yes	Yes	Yes	No	Yes
Director	Yes	Yes	Yes	No	Yes
iTunes	Yes	Yes	Yes	No	Yes
QuickTime	Yes	Yes	No	Partial	No
RealPlayer	Yes	Yes	No	Yes	Yes
Winamp	Yes	Yes	No	Yes	Yes
Windows Media Player	Yes	Yes	No	Yes	Yes

Audio Formats Support: Information about what audio file formats the players support.

	Lossy compression						**Lossless compression**		
	MP3	**WMA**	**RealAudio**	**Vorbis**	**AAC**	**AC3**	**APE**	**FLAC**	**ALAC**
Flash	Yes	Yes	No	No	No	No	No	No	No
Director	Yes	Yes	Yes	Yes	Yes	?	?	?	?
iTunes	Yes	Partial	No	No	Yes	No	No	No	Yes
QuickTime	Yes	No	No	No	Yes	No	?	No	Yes
RealPlayer	Yes	Yes	Yes	Yes	Yes	?	?	?	?
Winamp	Yes	Yes	No	Yes	Yes	No	No	No	No
Windows Media Player	Yes	Yes	No	No	No	No	No	No	No

Video Formats Support: Information about what video file formats the players support.

	MPEG-1	**MPEG-2**	**MPEG-4**	**WMV**	**RealVideo**	**Theora**	**Flash**
Flash	Yes	Yes	No	Yes	No	No	Yes
Director	Yes	?	?	No	Yes	?	Yes
iTunes	Yes	No	Yes	No	No	No	Yes
QuickTime	Yes	No	Yes	No	No	No	Yes
RealPlayer	Yes	?	?	No	Yes	?	Yes
Winamp	Yes	No	No	Yes	No	No	No
Windows Media Player	Yes	Yes	No	Yes	No	No	Yes

Container Formats Support: Information about what container formats the players support.

	AVI	**ASF**	**QuickTime**	**OGM**	**Matroska**	**MP4**
Flash	Yes	Yes	No	No	No	No
Director	Yes	?	Yes	?	?	Yes
iTunes	Yes	?	Yes	?	?	Yes
QuickTime	Yes	No	Yes	No	?	Yes
RealPlayer	Yes	No	No	?	?	?
Winamp	Yes	No	No	No	No	Yes
Windows Media Player	Yes	Yes	No	No	No	No

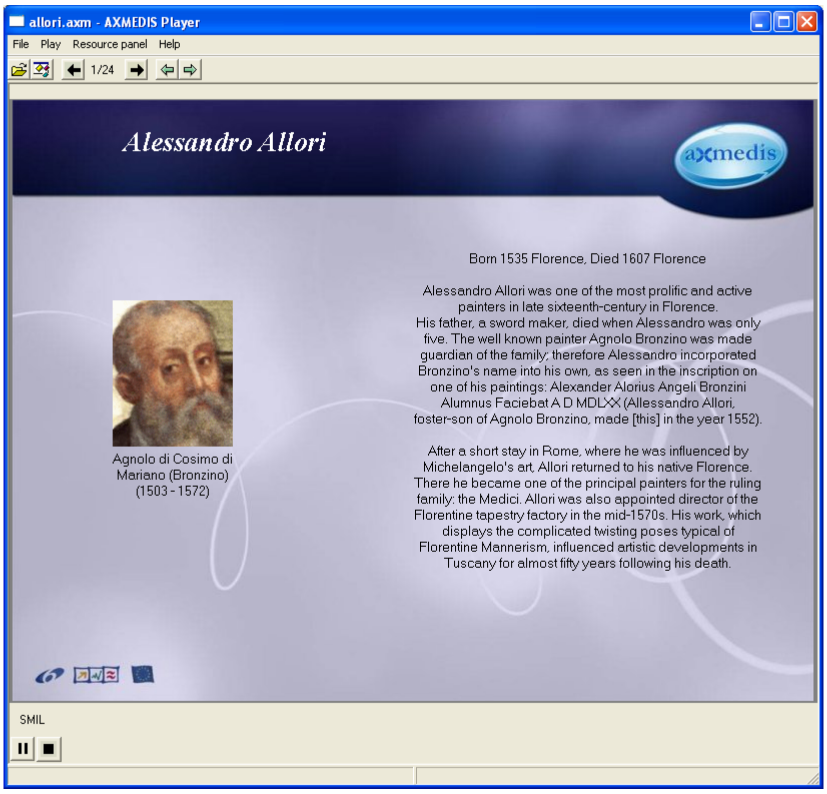
13 AXMEDIS PC Player: Work performed (DSI)

Two prototype players have been developed. The first one exploited the features of Microsoft Foundation Classes and ActiveX in order to rapidly develop a user friendly graphical interface able to open and display objects and the associated AXMEDIS hierarchy.
The second player is directly based on the AXMEDIS components not using the ActiveX.

The AXMEDIS PC Player allows to:

- load an AXMEDIS object as AXM and MP21 file
- browse the content in the object (only resources)
- hide/show the AXMEDIS hierarchy with the tree view showing the resources in the object
- get the content count (how many resources/objects are present in the object)
- play/stop the content fruition for audio, video, document and external resources
- view of metadata
- launch Internet Exploer to register the user
- import the user certificate received frim AXCS via e-mail
- certify the tool to allow using protected content

The following are snapshots of the AXMEDIS PC Player.



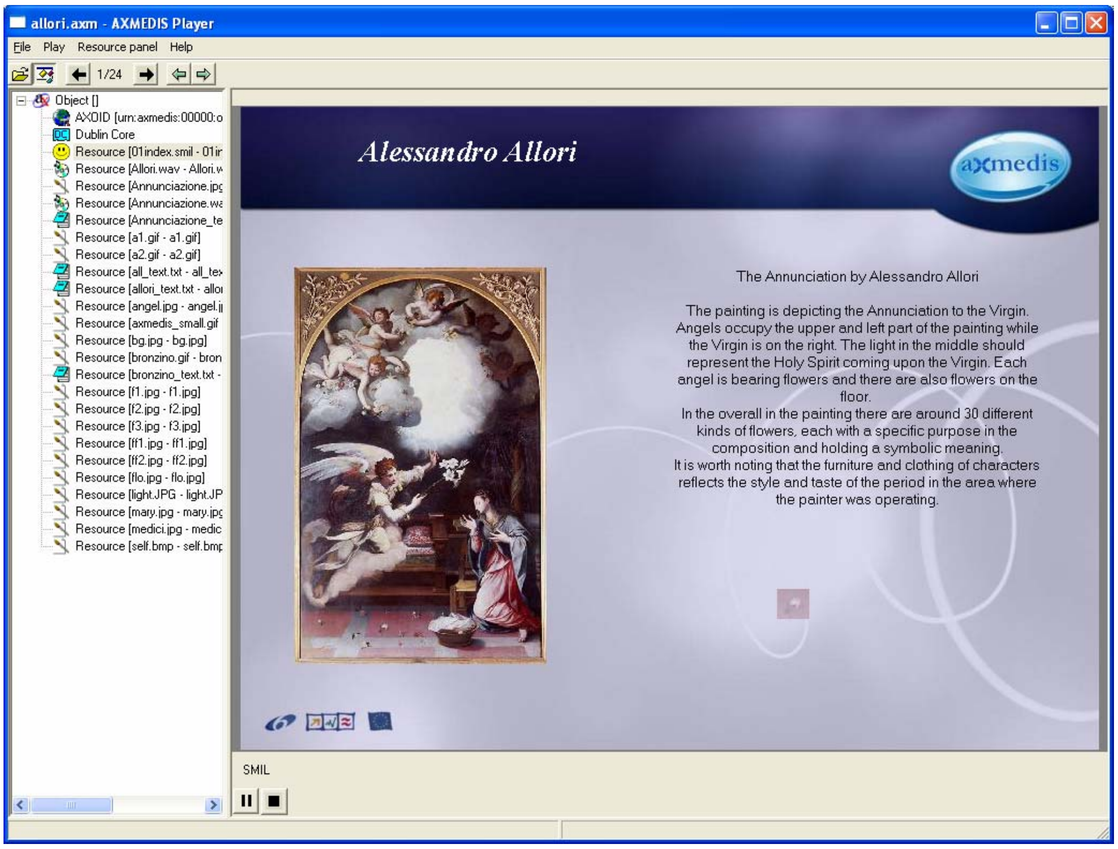
Agnolo di Cosimo di Mariano (Bronzino) (1503 - 1572)

Born 1535 Florence, Died 1607 Florence

Alessandro Allori was one of the most prolific and active painters in late sixteenth-century in Florence.
His father, a sword maker, died when Alessandro was only five. The well known painter Agnolo Bronzino was made guardian of the family; therefore Alessandro incorporated Bronzino's name into his own, as seen in the inscription on one of his paintings: Alexander Alorius Angeli Bronzini Alumnus Faciebat A D MDLXX (Allessandro Allori, foster-son of Agnolo Bronzino, made [this] in the year 1552).

After a short stay in Rome, where he was influenced by Michelangelo's art, Allori returned to his native Florence. There he became one of the principal painters for the ruling family: the Medici. Allori was also appointed director of the Florentine tapestry factory in the mid-1570s. His work, which displays the complicated twisting poses typical of Florentine Mannerism, influenced artistic developments in Tuscany for almost fifty years following his death.

The content of the AXMEDIS object may be browsed using the arrows in the toolbar or using the hierarchy view (opened using the button in toolbar)


allori.axm - AXMEDIS Player
File
Play
Resource panel
Help
1/24
Object []
AXOID [urn:axmedis:00000:o
Dublin Core
Resource [01index.smil - 01ir
Resource [Allori.wav - Allori.w
Resource [Annunciazione.jpg
Resource [Annunciazione.wa
Resource [Annunciazione_te
Resource [a1.gif - a1.gif]
Resource [a2.gif - a2.gif]
Resource [all_text.txt - all_tex
Resource [allori_text.txt - allo
Resource [angel.jpg - angel.j
Resource [axmedis_small.gif
Resource [bg.jpg - bg.jpg]
Resource [bronzino.gif - bron
Resource [bronzino_text.txt -
Resource [f1.jpg - f1.jpg]
Resource [f2.jpg - f2.jpg]
Resource [f3.jpg - f3.jpg]
Resource [ff1.jpg - ff1.jpg]
Resource [ff2.jpg - ff2.jpg]
Resource [flo.jpg - flo.jpg]
Resource [light.JPG - light.JP
Resource [mary.jpg - mary.jpg
Resource [medici.jpg - medic
Resource [self.bmp - self.bmp
Alessandro Allori
axmedis
SMIL

The Annunciation by Alessandro Allori

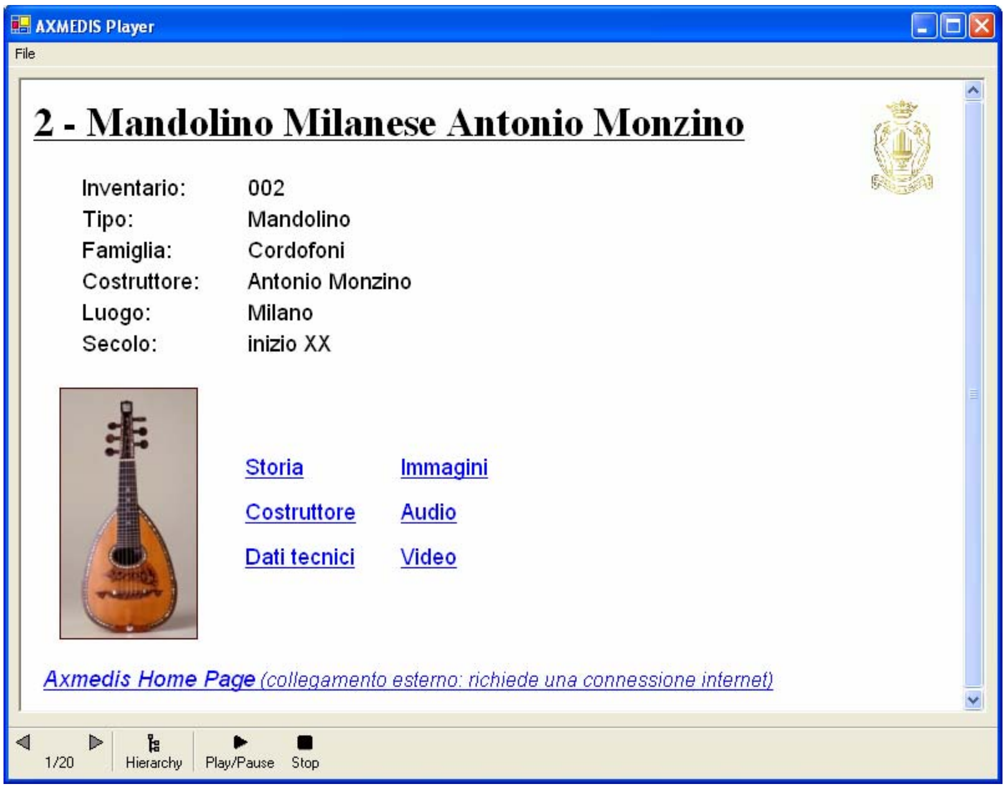
The painting is depicting the Annunciation to the Virgin. Angels occupy the upper and left part of the painting while the Virgin is on the right. The light in the middle should represent the Holy Spirit coming upon the Virgin. Each angel is bearing flowers and there are also flowers on the floor.
In the overall in the painting there are around 30 different kinds of flowers, each with a specific purpose in the composition and holding a symbolic meaning.
It is worth noting that the furniture and clothing of characters reflects the style and taste of the period in the area where the painter was operating.

14 AXMEDIS PC Player ActiveX based (DSI)

The preliminary prototype of the AXMEDIS PC Player allows to:

- load an AXMEDIS object
- browse the content in the object (only resources)
- hide/show the AXMEDIS hierarchy with the tree view showing the resources in the object
- get the content count (how many resources/objects are present in the object)
- play/stop the content fruition for audio, video, document and external resources
- view/load/edit/save of metadata in XML
- import user certificate
- certify the tool

Selecting the **File/Load…** in the menu it is possible to load an object. When an object is loaded his first resource in the hierarchy is automatically showed or played.



2 - Mandolino Milanese Antonio Monzino

Inventario:	002
Tipo:	Mandolino
Famiglia:	Cordofoni
Costruttore:	Antonio Monzino
Luogo:	Milano
Secolo:	inizio XX

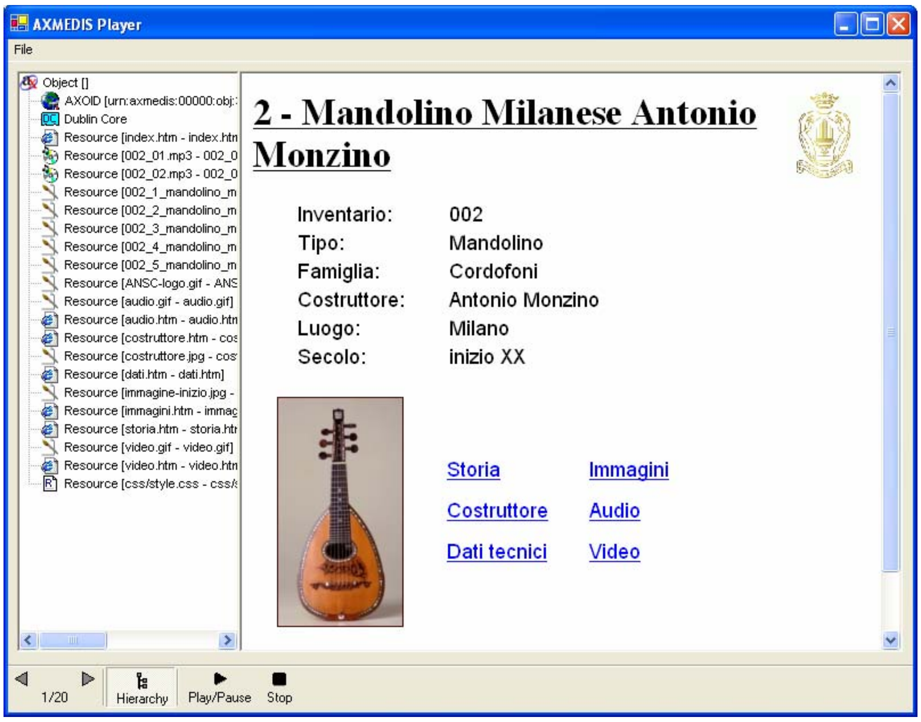
Storia Immagini
Costruttore Audio
Dati tecnici Video

Axmedis Home Page (collegamento esterno: richiede una connessione internet)

Figure 13 – AXMEDIS PC player rendering an ActiveX image

Note: in the figure above the ActiveX is only the image and not the buttons

When the Hierarchy button is pressed, the AXMEDIS Hierarchy is shown:



Object []
AXOID [urn:axmedis:00000:obj:
Dublin Core
Resource [index.htm - index.htn
Resource [002_01.mp3 - 002_0
Resource [002_02.mp3 - 002_0
Resource [002_1_mandolino_m
Resource [002_2_mandolino_m
Resource [002_3_mandolino_m
Resource [002_4_mandolino_m
Resource [002_5_mandolino_m
Resource [ANSC-logo.gif - ANS
Resource [audio.gif - audio.gif]
Resource [audio.htm - audio.htn
Resource [costruttore.htm - cos
Resource [costruttore.jpg - cos
Resource [dati.htm - dati.htm]
Resource [immagine-inizio.jpg -
Resource [immagini.htm - immag
Resource [storia.htm - storia.htr
Resource [video.gif - video.gif]
Resource [video.htm - video.htn
Resource [css/style.css - css/s

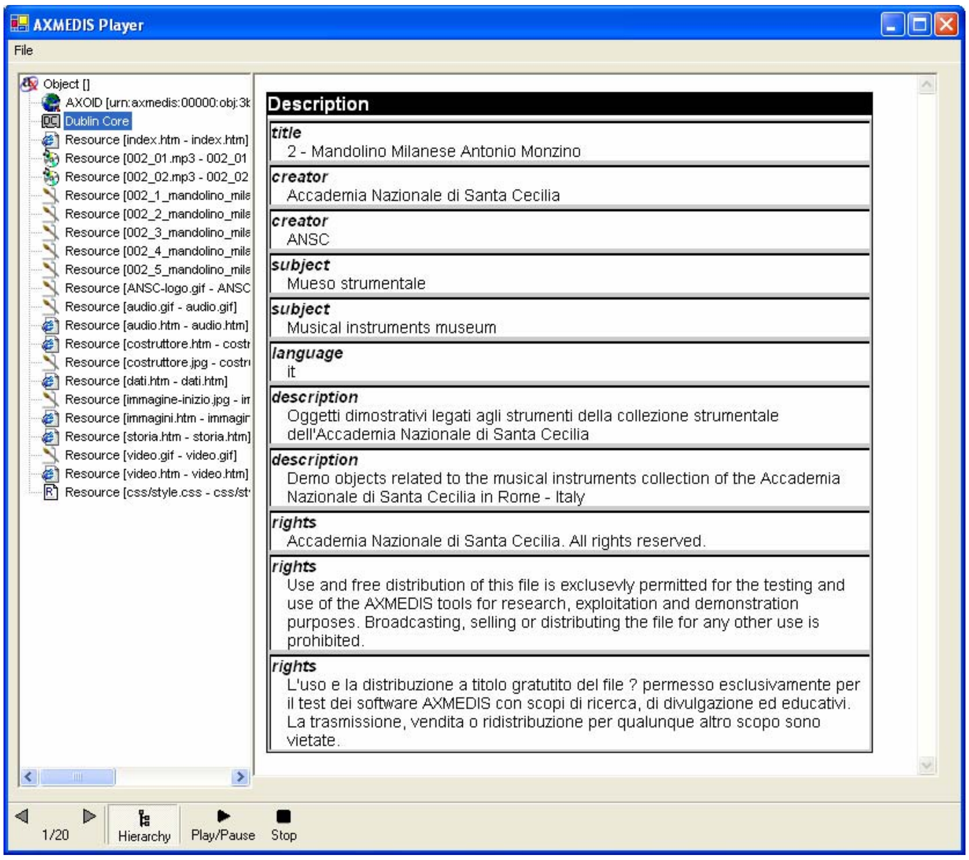
2 - Mandolino Milanese Antonio Monzino

Inventario:	002
Tipo:	Mandolino
Famiglia:	Cordofoni
Costruttore:	Antonio Monzino
Luogo:	Milano
Secolo:	inizio XX

Storia Immagini
Costruttore Audio
Dati tecnici Video

Figure 14 – AXEDIS hierarchy shown with the resource

When the Dublin Core element in the tree view is selected it is possible to edit, view, load and save metadata in XML format.


AXMEDIS Player
File
Object []
AXOID [urn:axmedis:00000:obj:3k
Dublin Core
Resource [index.htm - index.htm]
Resource [002_01.mp3 - 002_01
Resource [002_02.mp3 - 002_02
Resource [002_1_mandolino_mila
Resource [002_2_mandolino_mila
Resource [002_3_mandolino_mila
Resource [002_4_mandolino_mila
Resource [002_5_mandolino_mila
Resource [ANSC-logo.gif - ANSC
Resource [audio.gif - audio.gif]
Resource [audio.htm - audio.htm]
Resource [costruttore.htm - costr
Resource [costruttore.jpg - costr
Resource [dati.htm - dati.htm]
Resource [immagine-inizio.jpg - im
Resource [immagini.htm - immagir
Resource [storia.htm - storia.htm]
Resource [video.gif - video.gif]
Resource [video.htm - video.htm]
Resource [css/style.css - css/st
1/20
Hierarchy
Play/Pause
Stop

Description

title
2 - Mandolino Milanese Antonio Monzino

creator
Accademia Nazionale di Santa Cecilia

creator
ANSC

subject
Mueso strumentale

subject
Musical instruments museum

language
it

description
Oggetti dimostrativi legati agli strumenti della collezione strumentale dell'Accademia Nazionale di Santa Cecilia

description
Demo objects related to the musical instruments collection of the Accademia Nazionale di Santa Cecilia in Rome - Italy

rights
Accademia Nazionale di Santa Cecilia. All rights reserved.

rights
Use and free distribution of this file is exclusevly permitted for the testing and use of the AXMEDIS tools for research, exploitation and demonstration purposes. Broadcasting, selling or distributing the file for any other use is prohibited.

rights
L'uso e la distribuzione a titolo gratutito del file ? permesso esclusivamente per il test dei software AXMEDIS con scopi di ricerca, di divulgazione ed educativi. La trasmissione, vendita o ridistribuzione per qualunque altro scopo sono vietate.

Figure 153 – AXEDIS Dublin Core metadata view

When a resource is selected in the tree view, the resource is played or visualized.

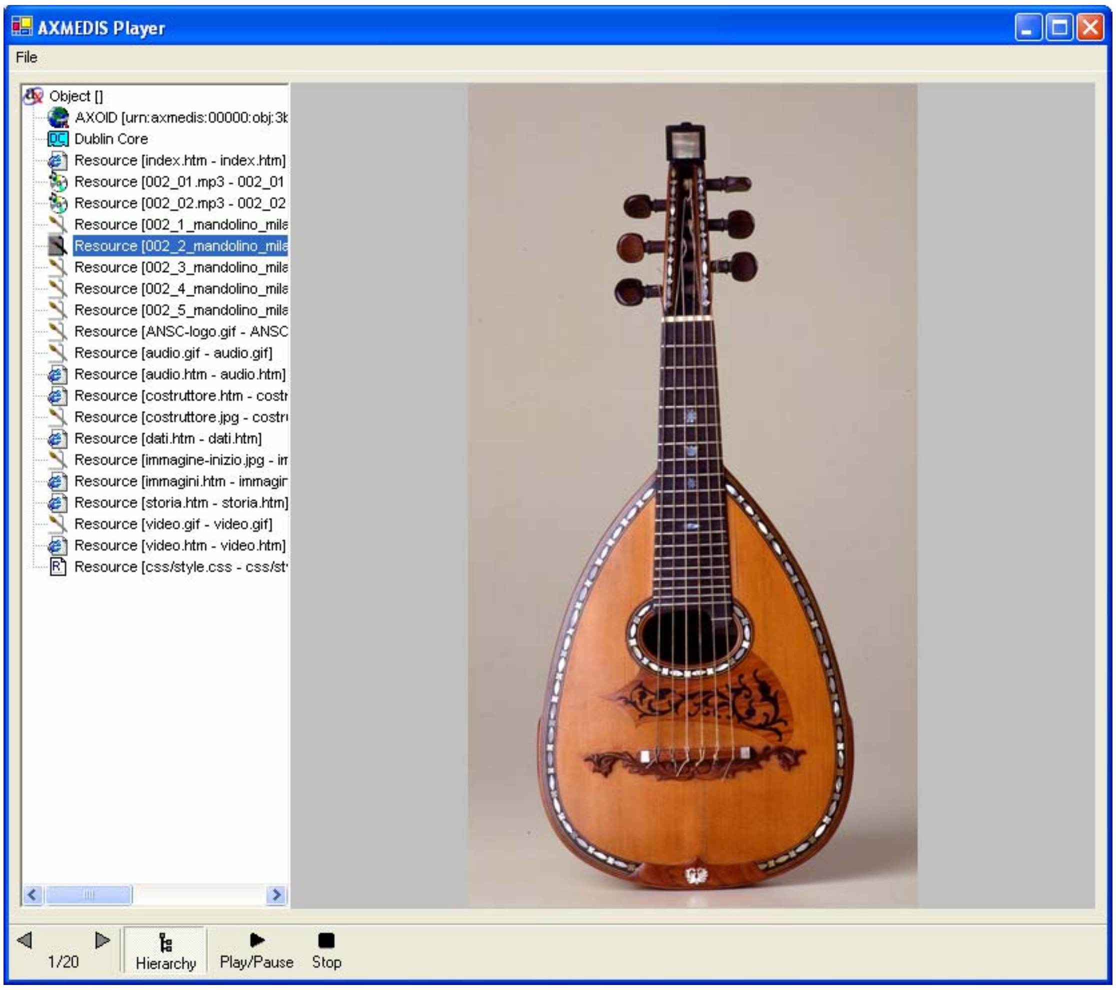

AXMEDIS Player
File
Object []
AXOID [urn:axmedis:00000:obj:3k
Dublin Core
Resource [index.htm - index.htm]
Resource [002_01.mp3 - 002_01
Resource [002_02.mp3 - 002_02
Resource [002_1_mandolino_mila
Resource [002_2_mandolino_mila
Resource [002_3_mandolino_mila
Resource [002_4_mandolino_mila
Resource [002_5_mandolino_mila
Resource [ANSC-logo.gif - ANSC
Resource [audio.gif - audio.gif]
Resource [audio.htm - audio.htm]
Resource [costruttore.htm - costr
Resource [costruttore.jpg - costr
Resource [dati.htm - dati.htm]
Resource [immagine-inizio.jpg - im
Resource [immagini.htm - immagir
Resource [storia.htm - storia.htm]
Resource [video.gif - video.gif]
Resource [video.htm - video.htm]
Resource [css/style.css - css/st
1/20
Hierarchy
Play/Pause
Stop

Figure 164 – Example of image resource view

Through the navigation buttons it is possible to view of play the next or the previous resource. Also the content count with the total number of resource available in the object is showed.

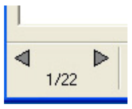


Figure 175 – The navigation buttons

Through the Play/Pause and Stop buttons it is possible to control an audio or a video resource playing.

Figure 186 – The Play/Pause and the Stop buttons

All types of resources managed by the AXMEDIS Editor are also visualized or played in the Pc Player ActiveX based.

15 SMIL Player in AXMEDIS player (EPFL)

The SMIL player used in AXMEDIS will be based on the AMBULANT SMIL Player. The developed prototype of the AMBULANT SMIL player running embedded in a wxWidgets application can be used by users to perform the standard operations of a player: play, stop, pause, close and open a SMI L resource from an AXMEDIS object.

15.1 SMIL Player in AXMEDIS player: Work performed (EPFL)

The SMIL player used in AXMEDIS will be based on the AMBULANT Player. The AMBULANT Open SMIL Player is an open-source, full SMIL 2.0 media player. It is intended for researchers and developers who want a source-code player upon which they can build higher-level systems solutions for authoring and content integration, or within which they can add new or extended support for networking and media transport components. The AMBULANT player may also be used as a complete, multi-platform media player for applications that do not need support for closed, proprietary media formats. The AMBULANT player written in C++, is distributed under a modified GPL license, and it is available for Windows, Linux, and Macintosh.

When the user wants to display the SMIL component, the Player will extract it from the AXMEDIS Object, uncompress it from binary to a media file and store it as a temporary file on the disk. The SMIL file is described in section 8.1.2 as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<smil>

  <head>
    <layout>
      <root-layout background-color="red" height="600" id="rootlayout"
title="AXMEDIS" width="800"/>
      <region height="143" id="RegionName1" left="105" top="124" width="104"/>
      <region height="145" id="RegionName2" left="295" top="125" width="235"/>
      <region height="147" id="RegionName3" left="234" top="287"
width="357"/></layout>
  </head>

  <body>
<par>
    <audio begin="2" end="16" region="RegionName1" src="Reference1"/>
    <video begin="6" end="21" region="RegionName2" src="Reference7"/>
    <video begin="2" end="14" region="RegionName3" src="Reference5"/>
</par>
  </body>

</smil>
```

The src value refers to the resources contained in the AXMEDIS Object, the player extracts relevant media resources from the AXMEDIS Object, stores them as temporary media files on the disk. After this, the Player will replace the <body> part of the above file with these directories of temporary media files and returns them to a new temporary SMIL file with the <body> part as follows:

```
  <body>
    <audio begin="2" end="16" region="RegionName1" src="C:\Temp\Birthday.mp3"/>
    <video begin="6" end="21" region="RegionName2" src="C:\Temp\BPz3-g.mpg"/>
    <video begin="2" end="14" region="RegionName3" src="C:\Temp\Fz3s.mpg"/>
  </body>
```

However this is a temporary solution a way to get resources without saving them to disk have to be studied.

It works but it consumes a lot of memory space. Considering the limited space in different clients, such as PDA or mobile phone, this may does not work.

How to play a SMIL component directly from AXMEDIS object?

This section attempts to explain the basic structure of the internal SMIL Player by loosely explaining what happens when you run it and play an AXMEDIS object. The main mechanism and process are based on those of AMBULANT SMIL player. We modify the section of generating data source to get it integrated with AXMEDIS editor. If you want to have a better knowledge of this mechanism, please check AMBULANT website and it will definitely help you to understand our method here.

1. Opening a SMIL component inside an AXMEDIS object

When the user selects Open (or double-clicks) we need to get the data of the SMIL component from AXMEDIS object. It generates the decompressed stream and gives it to the Player. There is an XML parser as a third party library to parse the stream into a DOM tree and create a player to play that DOM tree. In addition, we need to tell the player how it can obtain media data, create windows and more, because the media data are described as "AxIndex" and the content of media data are stored in the AXMEDIS object.

Most player implementations (the Windows player is an exception) have a class with a name like mainloop to handle this. Such a mainloop is created per SMIL component. This mainloop object will first create the various factories and populate them:

- A **window factory** is usually implemented by the main program itself. The player will call this when it needs a window. Usually the first request to create a window will actually return the document window (after resizing to the appropriate size).
- A **global playable factory** is created. This is the object the player will use to create renderers for the various media types. The global factory is filled with the various renderers this ambulant player supports. In effect, this is the step where you get to decide how various media are rendered.
- A **datasource factory** is created and filled with the factory functions that will create datasources for audio, video or other, raw, data such as text. The factory functions that are added to the datasource factory partially determine how data is retrieved over the net, which protocols and formats are supported and such. Partially, because some media items (audio and video, notably) may be rendered by simply passing the URL to some underlying media infrastructure such as DirectX or QuickTime. Note, **since we will directly retrieve the resource through "AxIndex" instead of using original URL, we will modify the code here. i.e., to generate the data source or data stream from "AxIndex".**

Next the factories are put together in a factories struct, and if the architecture supports dynamically loadable plugins we get the plugin engine singleton object and ask it to load the plugins. This will search the plugin directories for dynamic objects with the correct naming convention, load them, and call their initialize routine. The factories object is passed to the initialize routine, so the plugin itself can register any factories it wants.

The next step is to create the DOM tree. One way to do this is to use to "axom::getAxObjectElement(*resourceIndex)" to read the SMIL component from the AXMEDIS object, and then pass this data to document::create from string. This will return a document object. This object contains the DOM tree itself (implemented by the node object) and some context information. **Note: In AMBULANT SMIL Player, The context information are XML namespace information, original URL for resolving relative URLs used in the document, a mapping from XML IDs to node objects. Since we will directly retrieve the resource through "AxIndex" instead of using original URL and resolved relative URLs, we have to modify the code here.**

The final step is to create a player object. This is done through create smil2 player, passing the document, the factories and one final object, embedder. This object is again implemented by the main program, and implements a small number of auxiliary functions, such as opening an external webbrowser or opening a new SMIL component.

2. Creating the player (same as AMBULANT)

When the smil player object is created it gets the document, factories and embedder arguments. It now needs to create its internal data structures to facilitate playback later on:

- A timer and event processor are created. The timer is the master clock for the presentation, and the event processor is a runqueue object that is used for low-level scheduling. Whenever the high-level scheduler wants some code to be executed it will add an event to the event processor, possibly with a timeout and a priority. The event processor runs in a separate thread, waits for events in it runqueue to become elegible and then runs them. This mechanism is the underlying engine that makes the whole player work, anything that needs to wait doesn't do so inline but uses the event processor to get a callback at a later stage: the scheduler, renderers needing input data, etc.

- A layout manager is created, which will be used to find where media items should be displayed. The smil layout manager reads the <layout> section of the DOM tree and builds a parallel layout tree (which also contains information on some of the body media nodes, the ones that have layout information themselves) of region node objects. Then this tree of region node objects is converted into a tree of surface template objects. To create toplevel windows the new topsurface method of the window factory is used, and subregions are created using the new subsurface method of their parent surface template. The layout manager also contains mappings to be able to get from a node to the corresponding region node to the surface template, and this will be used during playback to play media items in the correct location.
- A timegraph is created. This is the internal representation of the <body> part of the DOM tree that will be used to play back the document. In addition a scheduler is created, which will interpret the data in the timegraph.

3. Starting playback

When the user selects Play we call the start method of the player object. This will invoke start on the scheduler. This will start playing the root node of the tree. The scheduler will now do all the SMIL 2 magic, whereby events such as the root node being played causes other nodes to become playable, etc. At some point a media item needs to be rendered. The scheduler calls the new playable method from the global playable factory. This will pass the DOM node to the various factories until one signals that it can create a playable for the object. In addition, if the playable has a renderer (which is true for most media objects, but not for things like SMIL animations) we also obtain the surface on which the media item should be renderered, through the layout manager. We then tell the renderer which surface to use. Soon afterwards the start method of the playable is called to start playback. **An average renderer will need to obtain data from AxIndex (note, originally, AMBULANT obtains from URL). It will do this by creating a datasource for the document through the datasource factory object.** Every time the renderer wants more data it calls the start method of the datasource passing a callback routine. Whenver data is available the datasource will schedule a call to the callback routine, through the event processor. When the renderer has enough information to start drawing it will not actually draw immedeately, but it will send a need redraw call to its surface. This will percolate up the surface hierarchy, to the GUI code, and eventually come back down as a redraw call all the way to the renderer (assuming it is not obscured by other media items, etc). At this point the bits finally get drawn on the screen. Whenever anything “interesting” happens in the renderer (the media item stopped playing, the user clicked the mouse, etc) it invokes a corresponding method on its playable notification. This interface is implemented by the scheduler, and these notifications are how the scheduler gets informed that it can start scheduling new things, etc.

4. Steps for real integration

Steps for ambulant player integration:

- Modifying the SMIL file input to direct access
- Modifying text/image/audio/video renders to accept a link to an AxObject
- Modifying memfiles for direct playing text and images
- Creating a DirectShow input with direct play. (In common with DSI)
- Adapting the output to fit the AxEditor frames

Modifying the SMIL file input to direct access:
This is done by defining new methods for DOM Tree/document generation and by using these new methods in a new player constructor with stream interface.

Modifying text/image/audio/video renders to accept a link to an AxObject:
There we have to modify all functions that call loading(render) function to accept an AxObjectManager pointer to access the played data. In the renders functions, we have to solve the “url” to AxIndex to get the datas (done by FindResourceIndex in memfile).

Modifying memfiles for direct playing text and images:
Memfiles are used in text and image playing. By modifying them we allow direct access for image and text. For audio and video, we add all necessary static method for getting istream* from “url”.

Creating a DirectShow input with direct play. (In common with DSI):

We have to modify the way Ambulant calls DirectShow to be compliant to istream*. For this we have to define a new DirectShow SourceFilter. And connect this new SourceFilter to the Renders.

Adapting the output to fit the AxEditor frames:
This consists in several little modifications for frames.

•

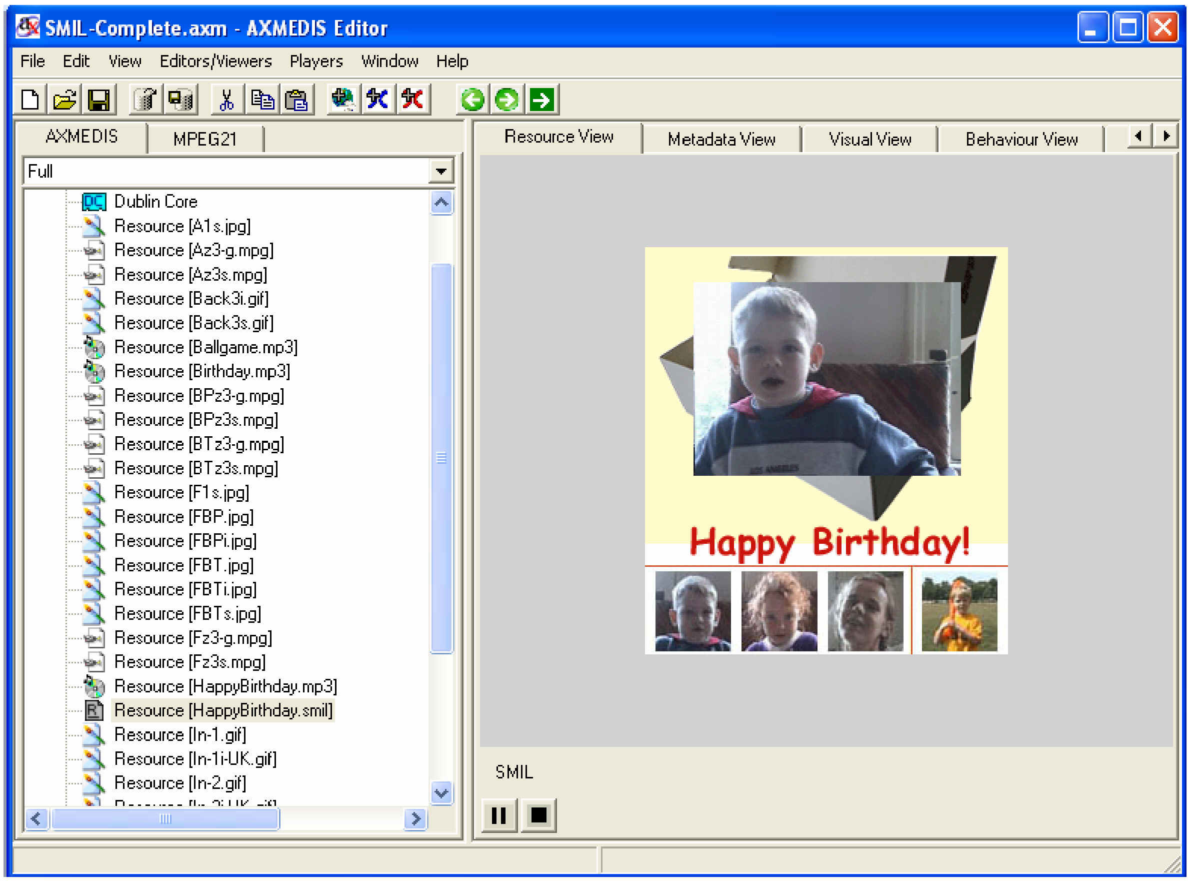


Figure 19 – Sample SMIL presentation rendered by the internal AXMEDIS SMIL player

15.2 SMIL Player in AXMEDIS player: problems and actions (EPFL)

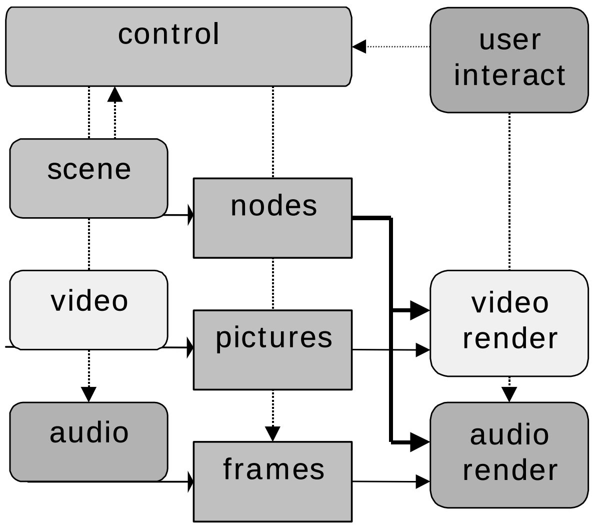
How to use DirectShow to play Audio and Video resources

1. DirectShow has no default interface for (stream/buffer)
2. DirectShow moved from DirectX SDK to Platform SDK – some problems with missing files.

We planned to solve the above problems with the following methods:
1. Creating a new Source Filter (Output Pin) for DirectShow
2. Connecting the new Source Filter to the render

16 MPEG-4 OSMO Player in AXMEDIS player (EPFL)

The MPEG-4 internal player constitutes a slightly different case of Media Player for AXMEDIS. In fact MPEG-4 itself not only supports media content in terms of different media files or streams, but it satisfies a much more relevant number of requirements providing tools to multiplex and synchronize all the elementary media streams even in the wider context of a rich multimedia scene (including user navigation, user interaction, inherent behavior of the scene and presentation of natural and synthetic sounds and media). All of these are included in a compliant MPEG-4 Player, so that any kind of control description or rule is normally coded inside the mp4 file or systems specific stream. The overall architecture of the Player in accordance to the MPEG-4 specification is reported in the following picture (control flow in dotted lines):



For all these reasons including the MPEG-4 Player into the internal AXMEDIS resources may result rather straightforward as only a very reduces number of commands are transmitted from the current *Player* user interface to the underlying architecture (executive control).

The overall player interface can be based on the abstract class **axMediaPlayer** (see previous sections above), through the specialized class **axMPEG4Player**. The functionality that is implemented by this class is rather reduced in terms of operations, given the complex architecture of the player itself and associated content described above.
Currently the MPEG-4 Player can allow two working modes:

- Network Channel (DMIF): in this modality the only possible command is **open** of a network address After this is done by validation of the rights through the AXOM, all the streaming content is received and rendered including audiovisual objects and scene/interaction. Connection is closed when a new one is open or the Player is closed.
- File (MP4): in this modality and under the AXOM control **load** of a file is possible and content is available as for the network modality. In any case this mode may allow the implementation of simple axMedia functions like **start/stop/pause** since file is available and no indeterminate buffering is necessary. All more than this may be really complex as it will interact with the decoding process of all built-in MPEG-4 decoders. More complex behavior for multiple media in AXMEDIS can be implemented in single objects linked through SMIL in the main AXMEDIS Player (and Editor).

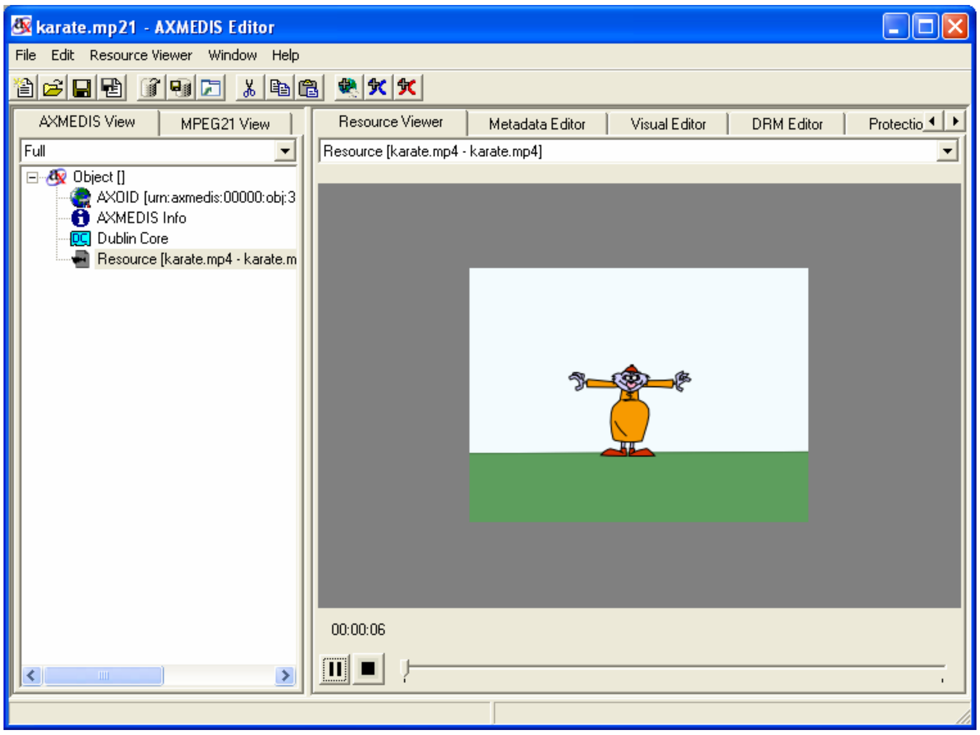
Once open or load are allowed, user activity can be monitored by built-in tracing capabilities and possibly reported: it is in any case *MPEG-4 activity* in terms of operation on the MPEG-4 content by built-in sensors and controls.

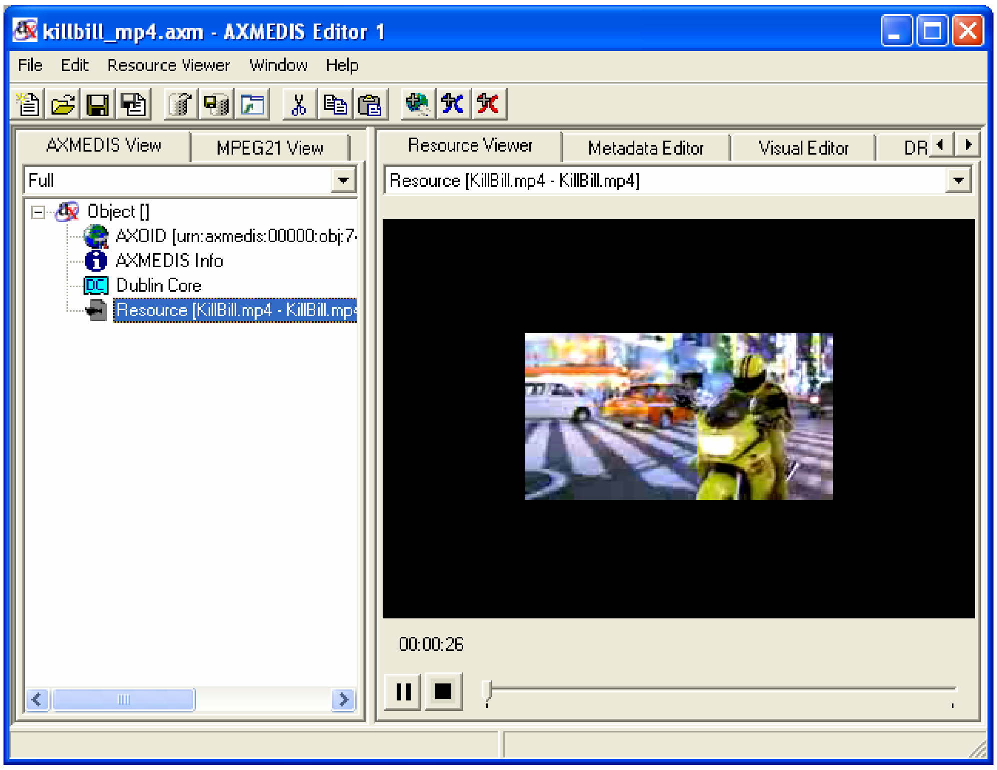
16.1 MPEG-4 OSMO Player in AXMEDIS player: Work performed (EPFL)

- **Directly display mp4 resource without any temporary files.**

How to directly access the MPEG-4 resource from AXMEDIS object without saving a temporary file ? The OSMO player is based on the GPAC library. GPAC is written in C for portability reasons (embedded platforms and DSPs), attempting to keep the memory footprint as low as possible. GPAC provide a possible work mode on the MPEG-4 file, however, AXMEDIS editor only provide a C++ istream class interface to access the MPEG-4 resource in AXMEDIS object. A simple solution for this problem is to extract the MPEG-4 resource and save them into a temporary MPEG-4 file. But this solution is not optimal because it requires an addition disk space for storing the temporary file.

A better solution is that the player directly accesses the MPEG-4 resource from AXMEDIS object without saving a temporary file. To implement this solution, GPAC interface has to be changed. A new interface has been developed and it consists of several interface functions, which make it possible for GPAC C code to operate on a C++ istream class instead of a file. Then some interface source code in GPAC has to be modified, and all the source code relative to file operation has to be replaced by the new interface, which is used for player to access the MPEG4 resource from AXMEDIS object using C++ istream class.


karate.mp21 - AXMEDIS Editor
File Edit Resource Viewer Window Help
AXMEDIS View
MPEG21 View
Full
Object []
AXOID [urn:axmedis:00000:obj:3
AXMEDIS Info
Dublin Core
Resource [karate.mp4 - karate.m
Resource Viewer
Metadata Editor
Visual Editor
DRM Editor
Protectio
Resource [karate.mp4 - karate.mp4]
00:00:06


killbill_mp4.axm - AXMEDIS Editor 1
File Edit Resource Viewer Window Help
AXMEDIS View
MPEG21 View
Full
Object []
AXOID [urn:axmedis:00000:obj:7
AXMEDIS Info
Dublin Core
Resource [KillBill.mp4 - KillBill.mp
Resource Viewer
Metadata Editor
Visual Editor
DR
Resource [KillBill.mp4 - KillBill.mp4]
00:00:26

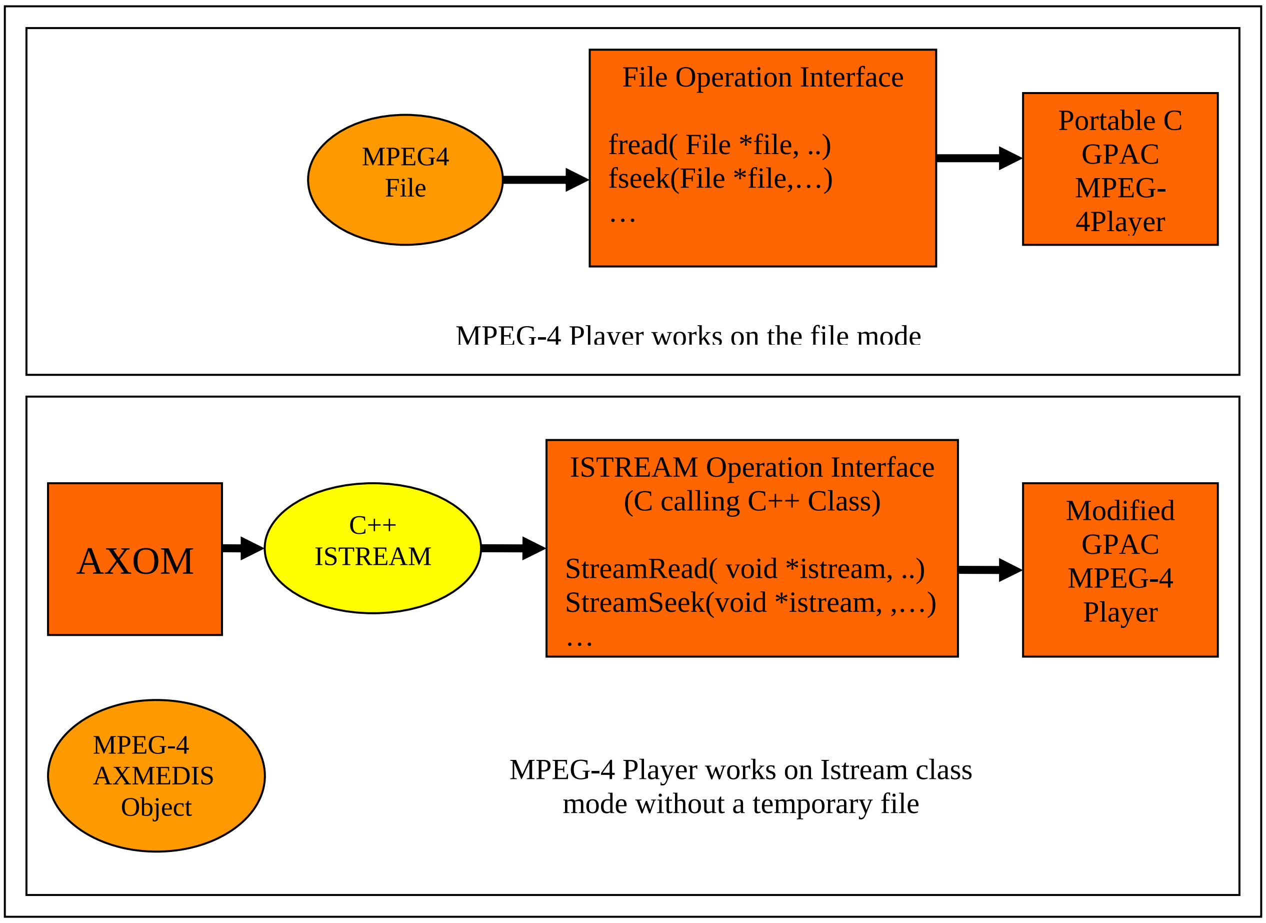

File Operation Interface
fread(File *file, ..)
fseek(File *file,…)
…
MPEG4 File
Portable C GPAC MPEG-4Player
MPEG-4 Player works on the file mode
AXOM
C++ ISTREAM
ISTREAM Operation Interface (C calling C++ Class)
StreamRead(void *istream, ..)
StreamSeek(void *istream, ,…)
…
Modified GPAC MPEG-4 Player
MPEG-4 AXMEDIS Object
MPEG-4 Player works on Istream class mode without a temporary file

Figure: MPEG-4 Player accesses MPEG4-resource from an AXMEDIS object using an ISTREAM operation interface without a temporary file

16.2 MPEG-4 OSMO Player in AXMEDIS player: problems and actions (EPFL)

17 Comments on the full round of AXMEDIS process

- **User registration and certification**
  - Now it is online since the CA and the AXCS are connected
  - Certificates are produced using the Registration Portal
- **Tools registration**
  - Ok
  - Enabling code, is done automatically
- **Production of a new objects**
  - **What you can do:**
    - Putting Single resource, multiple resources, etc.
      - Using Drag and drop, etc…
    - Definition of metadata, etc…
    - To be improved: Definition of internal PAR, loading them from some db
    - To be improved: Definition of the par a subset of IPAR
- **Request of AXOID**
  - **What you can do:**
    - Direct connection with the AXCS from the AXMEDIS editor to request the final AXOID
- **Object protection**
  - **What you can do:**
    - We can protect the object with custom parameters
- **Registration of the object on AXCS, posting of protection information on AXCS**
  - **What you can do:**
    - registration of the object from AXMEDIS editor and AXCP tools
  - **What is missing:**
    - Generation of object signature and storing of the signature in the object
    - Storing of AXMEDIS object DC metadata
- **Object save, download, distribution**
  - **What you can do:**
    - Save as .axm (XML) or .mp21 (MPEG21 File Format)
    - Download from http
    - Upload/Download on P2P network with AXEPTOOL
- **Production of the license**
  - License can be produced from
    - AXMEDIS Editor internal License Editor
    - Independent License editor
    - It possible to create a license and post on the PMS
  - FUPF Reporting
    - What is accessible: PMS Client – PMS Server allow creation and management of licenses
    - What is missing: Pending integration issues from DSI. Pending interface improvements.
    - Which problems are pending: PAR discussion → Reference to PAR from AXMEDIS object
  - **Problems and work to be done**
    - The present version works only if the license is produced manually
    - The DRM editor talks only with one PMS Server, the Server URL has to be moved into the Configuration.
    - No ask of AXCID and AXOID, implicit values
    - First case: Distributor producing, protecting and distributing
      - Verification: PAR license verification
    - SECOND case: Content producers, protecting and distributing to distributor
      - Verification:
        - Parent license
        - PAR license verification
    - Posting of the license on the PMS automatically

- **Object usage on a player**
  - **What you can do:**
    - Register the user
    - Install user certification
    - Certify the tool
    - It does not work if you do not have the license
    - Allow you playing when you have the right license
    - …..
  - **What is missing:**
    - ……nothing up to now…….
  - **Problems:**
    - ……testing pending…….

18 AXMEDIS PDA Player (TISCALI, DSI)

18.1 AXMEDIS PDA Player: Work performed

- The porting of AXOM libraries over PDA platforms and that of following well-known libraries:
  - STLPort version 5.0.1 or greater. This is an open source implementation of the C++ Standard Template Library (STL). It has already been ported for eVC4 and WCE platforms. In that way, AXOM code, which relies on STL classes, should not be changed during the porting.
  - Xerces-C++ version 2.7.0. Some effort has to be done to port this library on PDA platform since Xerces community does not provide any hint for doing that port. Only general guidelines are available.
  - OpenSSL version 0.9.8a or greater. It already provides means to be ported on PDAs. The main issue in doing that is the introduction of a support library (WCECompat version 1.2) which provides to OpenSSL some feature of the C Runtime Library which are missed in the eVC4 libraries. In order to avoid complicated integration among STLPort, WCECompat and standard eVC4 CRT it is suitable to use OpenSSL as dynamic library on PDA.
  - Missing of: libcurl.lib for http, ftp protocol implementation
  - Missing: estimation of fingerprint
  - Missing: management of the current directory, use as current directory the path of the executable file.
- Porting of the OSMO MPEG-4 player
  - MS Windows Pocket PC 2003
  - MS Windows Mobile 2005
  - Visual Studio 2005
  - Moving its user interface to AXMEDIS player one
  - The migration of the interface to go into that of AXMEDIS is needed
- Porting of the SMIL ambulant player
  - SMIL ambulant engine porting done: libAmbulant is the main engine to play SMIL contents in the PDA player
- The integration in the PDA player: OSMO and AMBULANT
  - The PDA player is a single MS Windows Application shell containing both technologies
  - The OSMO package is based on a single engine library called GPAC.
  - GPAC has a FILE* based streaming mechanism (a mini shim was written to deal with FILE as C++ streams)
  - Ambulant has c++ stream so the access to the resource was easier
- Stream from AXOM passed to GPAC and AMBULANT

  - AXOM Presents c++ std::streams. When openend, after discriminating the mimetype, the stream are passed to the correct engine (GPAC or AMBULANT)
  - When the AXPDA Player opens a content, AXOM methods are used to parse contents, and if Playble contents are found, the play starts (with correct player)
- The User Interface
  - A Standard Windows Mobile 2005 application has been created. With:
    - Splash screen
    - Main Menu
      - File Menu
        - Certification Menu
      - View Menu
    - Main GPAC window
    - Main AMBULANT window
    - Certification Dialog
  - No hierarchy view viewer
- PMS Client has been ported on PDA
  - The AXPDA Player can play protect contents using the Protection Processor from AXOM
- Porting on HTML and document view, performed by DSI
- GPAC
  - Renamed configuration as AXPDA.cfg
  - The player from AXMEDIS editor is not in the frame on right but in a different window
- Other Issues Solved
  - The AXPDAPlayer has a reasonable size. Excluding all the external libraries, the executable is only 2.7 MBytes
  - The player is a fully Windows Mobile 2005 applications

18.2 AXMEDIS PDA Player: problems and actions

- PMS Client on PDA
  - A simple version of secure cache will be needed later
- Other Issues:
  - A better XVID code should be tested. Something ffmepeg based should have the correct optimization to play high quality video streams.

19 AXMEDIS Mobile Player (DSI)

19.1 AXMEDIS Mobile Player: Work performed

- **Example and experiments done and current:**
  - DSI: done, Player Symbian in C++ with progressive download
  - DSI: started, player pure java, with MPEG-2 streaming, single resources, few metadata
  - DSI: done, p2p among mobiles
  - DSI: started, scenarios editor for SMIL scenes
- **Mobile player hyp:**
  - Take up ELTEO is requesting the mobile player distributing video and audio
  - Take up DELTA is requesting the mobile player distributing video and audio, menu' flash or SMIL,
  - Take up 4HOME, OMA players
  - TISCALI: creation of a JavaME SMIL Player (subset of 2.0 SMIL documents) named SMILZO
- **Possible Solutions:**

    - Symbian in C++, too many cross compiling
    - Pure java, simpler, remake from C++
    - Windows Mobile 5, pratically derived from PDA version C++
- **Additional features**
    - Protection of the HTML, SMIL, Flash
    - A SMIL player in Java, for mobile is accessible
    - At least single resources
    - Streaming on MPEG-2 of the MPEG-21 files
    - TISCALI SMILZO plays streamed from server SMIL documents

19.2 AXMEDIS Mobile Player: problems and actions

- **Possible Decision:**
    - Pure Java implementation of simple player MPEG-21/AXMEDIS
        - Probably JAVA profile with Threads will be needed ?? for the unprotection
    - Usage of the local video-audio renderer
    - Implementation or reuse of a simple SMIL player for mobile in java at least for managing menu, images, etc., execution of URI and links
    - Production of the MOBILE version of AXMEDIS objects from AXMEDIS editor, with decomposition of AXMEDIS Object Model and SMIL if needed
        - The same features will be accessible on AXCP
        - Scripts have to be produced for testing and as examples
    - Possible support of AXOM with hierarchies into the java player,
        - first implementation with simple one layer support, with reduced metadata, etc.
        - then addition of hierarchy
    - protection processor:
        - only one level of protection
        - only one algorithm in the first version, more later
        - small parameters
        - fingerprint based on IMEI
    - Direct connection with PMS Server via protected channel, with SSL,
        - Certificate download for the SSL
        - no usage of the secure cache (at least for the first version)
        - simple secure cache later
- **Decisions:**
    - **Develop a pure java version of AXMEDIS player, as described above**
- **Time schedule:**
    - First Usable version for November 2007
- **Proposed Partners committed, under verification:**
    - TISCALI: SMIL and play, user interface
    - DSI: AXOM, Protection processor, connection, certificate,
    - EXITECH: RX stream MPEG-2, for realizing the streaming effect/progressive download, usage of a Server MPEG-2 stream
    - UPC, secure communication from the mobile and the PMS

20 Bibliography

[1] MPEG-21 Multimedia Framework Part 1: Vision, Technologies and Strategy. ISO/IEC 21000-1
[2] ENTHRONE Web Site: http://www.enthrone.org/
[3] AXMEDIS Deliverable DE 2.1.1 “User Requirements and use cases”
[4] ENIKOS Web Site: http://www.enikos.com/home.shtml
[5] WEDELMUSIC Web Site: http://www.wedelmusic.org/
[6] MPEG-21 Multimedia Framework Part 2 – Digital Item Declaration (DID). ISO/IEC 21000-2

[7] MPEG-21 Multimedia Framework Part 4 – Intellectual Property Management and Protection (IPMP). ISO/IEC 21000-4.
[8] MPEG-21 Multimedia Framework Part 3 – Digital Item Identification (DII). ISO/IEC 21000-3.
[9] SMIL overview on the W3C consortium Web Site: http://www.w3.org/AudioVideo/
[10] Oratrix Web Site: http://www.oratrix.com/
[11] LimSee2 Web Page on the INRIA Web Site : http://wam.inrialpes.fr/software/limsee2/
[12] AXMEDIS deliverable DE 4.3.1 " Content Composition and Formatting"
[13] Dublin Core Metadata Initiative Web Site : http://dublincore.org/
[14] Xerces Web Page on the Apache Web Site : http://xml.apache.org/xerces-c/
[15] Mozilla Web Site: http://www.mozilla.org/
[16] AXMEDIS Deliverable DE 4.6.1 "Content Distribution via Internet"
[17] MPEG-7 Part 1: Systems - ISO/IEC 15938-1
[18] The AMBULANT Project Web Site: http://www.cwi.nl/projects/Ambulant/distPlayer.html
[19] The BIM Open Source Web Site: http://bimopensourcesolutions.freewebspace.com/
[20] EXPWAY Web Site: http://www.expway.com/bim.php
[21] XMill Web Site: http://sourceforge.net/projects/xmill
[22] Macromedia Inc. Web Site: http://www.macromedia.com/
[23] Northcode Company Web Site: http://www.northcode.com/
[24] QuickTime Web Site: http://www.apple.com/quicktime/pro/
[25] Real Player Web Site: http://www.real.com/
[26] iTunes Web Site: http://www.apple.com/itunes/
[27] Winamp Web Site: http://www.winamp.com/
[28] AXMEDIS Deliverable DE 5.1.1 "AXMEDIS Framework Infrastructure"

21 Other Bibliography

- 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.
- 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.
- Salehi S., “ImageMagick Tricks, Web Image Effects from the Command Line and PHP”, Packt Publishing, June 2006
- Still Michael, “The Definitive Guide to ImageMagick”, APress, December 2005
- 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.