Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE3.1.2.3.4 Specification of AXMEDIS Editors and Viewers, update of DE3.1.2.2.4

Version: 2.7
Date: 22-07-2007
Responsible: DSI (revised and approved by coordinator)

Project Number: IST-2-511299
Project Title: AXMEDIS
Deliverable Type: report
Visible to User Groups: yes
Visible to Affiliated: yes
Visible to the Public: yes

Deliverable Number: DE3.1.2.3.4
Contractual Date of Delivery: M34
Actual Date of Delivery: 27/09/2007
Title of Deliverable: Specification of AXMEDIS Editors and Viewers
Work-Package contributing to the Deliverable: WP3.1
Task contributing to the Deliverable: WP3, WP2
Nature of the Deliverable: report
Author(s): DSI, FUPF, EPFL, UNIVLEEDS, FHGIGD, SEJER

Abstract: this part includes the specification of components, formats, databases and protocol related to the AXMEDIS Framework area including AXMEDIS Editors and viewers.

Keyword List: AXMEDIS Editor, authoring, MPEG-21, AXMEDIS viewers, AXMEDIS player, Active X, plug in.

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

Table of Content

1 Executive Summary and Report Scope

The full AXMEDIS specification document has been decomposed in the following parts:

DE number	Deliverable title	responsible
DE3.1.2.3.1	Specification of General Aspects of AXMEDIS framework AXMEDIS-DE3-1-2-3-1-Spec-of-AX-Gen-Asp-of-AXMEDIS-framework	DSI
DE3.1.2.3.2	Specification of AXMEDIS Command Manager AXMEDIS- DE3-1-2-3-2-Spec-of-AX-Cmd-Man	DSI
DE3.1.2.3.3	Specification of AXMEDIS Object Manager and Protection Processor AXMEDIS-DE3-1-2-3-3-Spec-of-AXOM-and-ProtProc	DSI
DE3.1.2.3.4	Specification of AXMEDIS Editors and Viewers AXMEDIS-DE3-1-2-3-4-Spec-of-AX-Editors-and-Viewers	DSI
DE3.1.2.3.5	Specification of External AXMEDIS Editors/Viewers and Players AXMEDIS-DE3-1-2-3-5-Spec-of-External-Editors-Viewers-Players	DSI
DE3.1.2.3.6	Specification of AXMEDIS Content Processing AXMEDIS-DE3-1-2-3-6-Spec-of-AX-Content-Processing	DSI
DE3.1.2.3.7	Specification of AXMEDIS External Processing Algorithms AXMEDIS-DE3-1-2-3-7-Spec-of-AX-External-Processing-Algorithms	FHGIGD
DE3.1.2.3.8	Specification of AXMEDIS CMS Crawling Capabilities AXMEDIS-DE3-1-2-3-8-Spec-of-AX-CMS-Crawling-Capab	DSI
DE3.1.2.3.9	Specification of AXMEDIS database and query support AXMEDIS-DE3-1-2-3-9-Spec-of-AX-database-and-query-support	EXITECH
DE3.1.2.3.10	Specification of AXMEDIS P2P tools, AXEPTool and AXMEDIS AXMEDIS-DE3-1-2-3-10-Spec-of-AXEPTool-and-AXMEDIA-tools	DSI
DE3.1.2.3.11	Specification of AXMEDIS Programme and Publication tools AXMEDIS-DE3-1-2-3-11-Spec-of-AX-Progr-and-Pub-tool	UNIVLEEDS
DE3.1.2.3.12	Specification of AXMEDIS Workflow Tools AXMEDIS-DE3-1-2-3-12-Spec-of-AX-Workflow-Tools	UR
DE3.1.2.3.13	Specification of AXMEDIS Certifier and Supervisor and networks of AXCS AXMEDIS-DE3-1-2-3-13-Spec-of-AXCS-and-networks	DSI
DE3.1.2.3.14	Specification of AXMEDIS Protection Support AXMEDIS-DE3-1-2-3-14-Spec-of-AX-Protection-Support	UPC
DE3.1.2.3.15	Specification of AXMEDIS accounting and reporting AXMEDIS-DE3-1-2-3-15-Spec-of-AX-Accounting-and-Reporting	EXITECH

1.1 This document concerns

Specification of AXMEDIS Editors and Viewers, first update of DE3.1.2 part B of the above list

1.2 List of Modules or Executable Tools Specified in this document

A module is a component that can be or it is reused in other cases or points of the AXMEDIS framework or of other AXMEDIS based solutions.
The modules/tools have to include effective components and/or tools and also testing components and tools.

Module/tool Name	Module/Tool Description and purpose, state also in which other AXMEDIS area is used	Standards exploited if any
AXMEDIS Editor	allows creating and manipulating AXMEDIS objects	
Hierachy Editor and Viewer	allows viewing and editing the AXMEDIS Object structure as AXMEDIS structure or as MPEG 21 structure	MPEG21 DIDL
DRM Editor and Viewer	allows viewing and updating DRM information	
Visual Editor and Viewer	allows editing the visual presentation of content in SMIL	W3C SMIL 2.0
Behaviour and Functional Editor and Viewer	allows editing the behavioural presentation of content in SMIL	W3C SMIL 2.0
Metadata Editor and Viewer	allows viewing and updating generica XML metadata	W3C XML
Metadata Mapper Editor and Viewer	allows mapping XML metadata	
Workflow Editor and Viewer	allows viewing and updating workflow information of AXMEDIS object	
AXMEDIS Content Tool Error Manager	allows to store errors happened in AXMEDIS Tools	
AMEDIS Editor Configuration Manager	allows to manage configuration information related to AXMEDIS Tools	
AXMEDIS Editor Plug-in Manager	allows generic management of plug-ins	
AXOM Content Processing	allows to manage content processing plug-ins	
AXOM Command and Reporting	allows to manage command and reporting plug-ins	
Internal Audio Player	allows reproduction of audio resources	
Internal Image Viewer	allows rendering of image resources	
Internal Video Player	allows rendering of video resources	
Internal MPEG-4 Player	allows the decoding of an MPEG-4 Systems compliant resource	MPEG-4 Systems, MPEG-4 IPMPX
Internal SMIL Player	allows the decoding of a SMIL presentation composed by synchronized audiovisual resources	W3C SMIL 2.0
Internal Document Viewer	allows to view document resources	
Method Player	allows to run MPEG21 DIP methods	

1.3 List of Formats Specified in this document

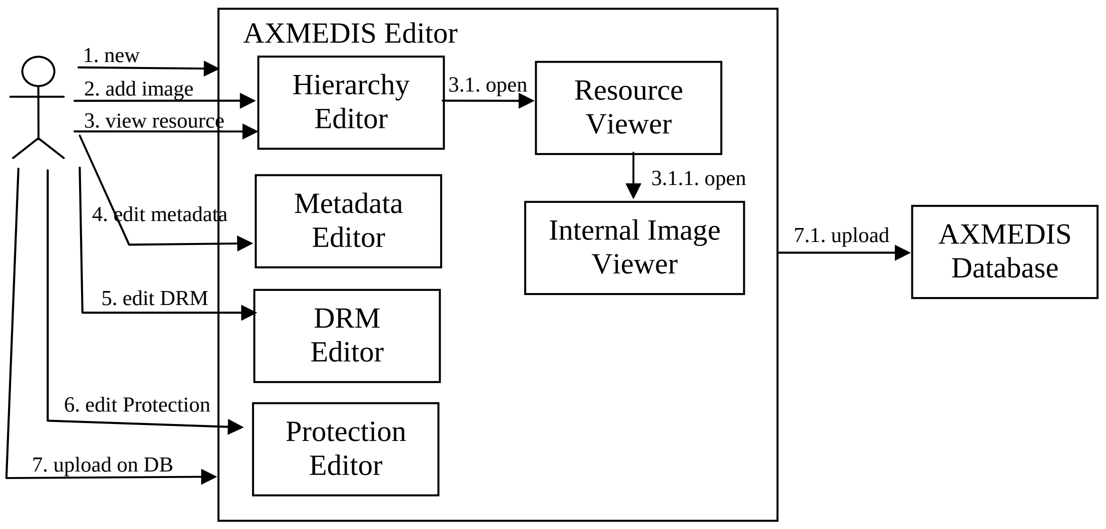
A format can be (i) an XML content file for modeling some information, (ii) a file format for storing information, (iii) a format that is manipulated by the tools described in this document, etc...

Format Name	Format Description and purpose, state also in which other modules is used	Standards exploited if any

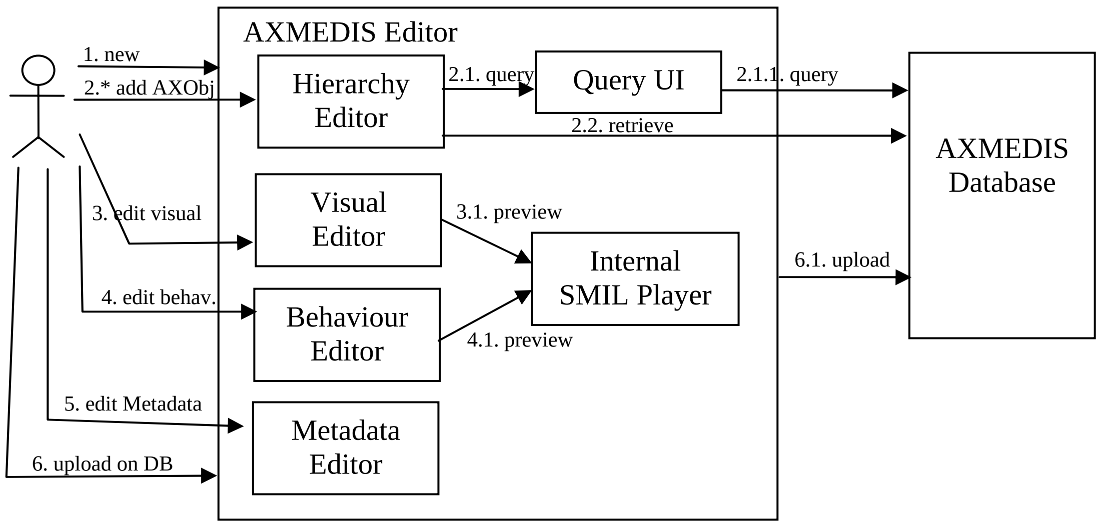
Error Coding	**XML based format to code errors information**	
Error Log	**XML based format to represent the errors happened**	
Configuration	**XML based format to represent the configuration information of AXMEDIS Tools**	
Plug-in description	**XML based format to represent functionalities provided by plug-ins**	
Content Processing Plug-in specific description	**XML based format for content-processing plug-ins functionalities**	
Parameter description	**XML based format for parameters of plug-ins**	

2 General Use Cases and scenarios

2.1 Use Case – Creation of a New AXMEDIS Object

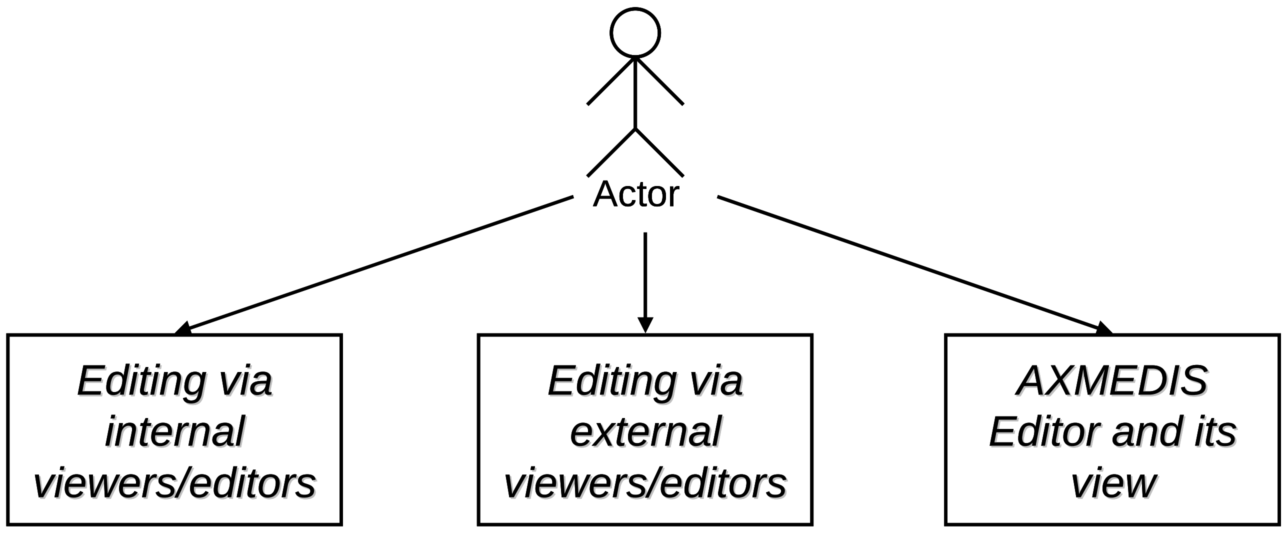


2.2 Use Case - Creation of Composite AXMEDIS Object



3 General architecture and relationships among the modules produced (DSI)

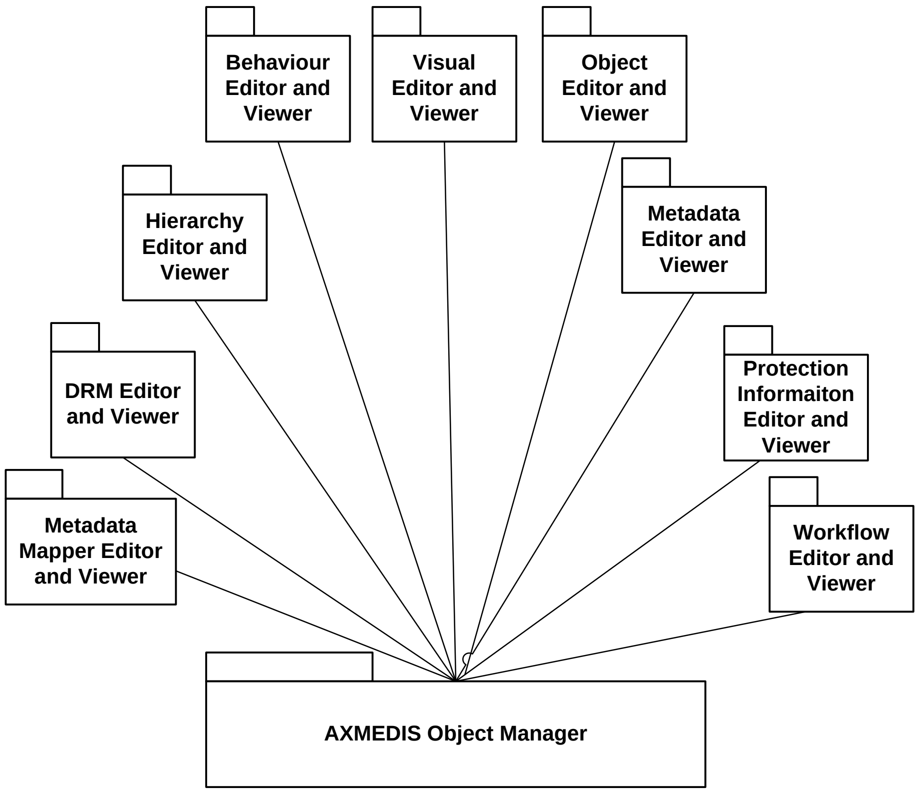
In this document everything concerning how to handle (play/edit) AXMEDIS content is detailed. Different scenarios of utilization are considered in the following sections. The beginning part put the basis to the construction of the AXMEDIS editor: all the editor/viewers that show different features of the structured content inside an AXMEDIS object are specified. These AXMEDIS Viewer can browse structure (hierarchy), multimedia feature (metadata, behaviour, visual, object), protection and DRM. Even workflow aspects are manageable in a suitable user interface integrated with the AXMEDIS Editor.
Errors and configuration of the AXMEDIS Editor are considered in the specification of the Content Tool Error Manager and Configuration Manager. The AXMEDIS Editor has to be extendable in order to add new content processing capabilities and to interoperate with different already established workflow management services. The AXMEDIS Editor Plug-in Manager has been specified in this document, providing methods to plug-in new functions (profile description and dynamic linking).
All the needed modules to render the multimedia resources included in the AXMEDIS objects (audio, video, images, documents, mpeg4, smil) are specified


Actor
Editing via internal viewers/editors
Editing via external viewers/editors
AXMEDIS Editor and its view

3.1 View Modules (DRM Edit/View, Hierarchy Edit/View, Metadata Edit/View, etc…)

View modules are those parts of AXMEDIS Editor GUI which show some aspects of the actual AXMEDIS object and parts thereof. In the following subsections will be analyzed the most important aspects (and the corresponding views) thought about till this moment, i.e.:

- Hierarchy;
- DRM;
- Visual;
- Behavioural and Functional;
- Descriptions and comments;
- Metadata;
- Etc.


Behaviour Editor and Viewer
Visual Editor and Viewer
Object Editor and Viewer
Hierarchy Editor and Viewer
Metadata Editor and Viewer
DRM Editor and Viewer
Protection Informaiton Editor and Viewer
Metadata Mapper Editor and Viewer
Workflow Editor and Viewer
AXMEDIS Object Manager

4 Executable Tool - AXMEDIS Editor (DSI)

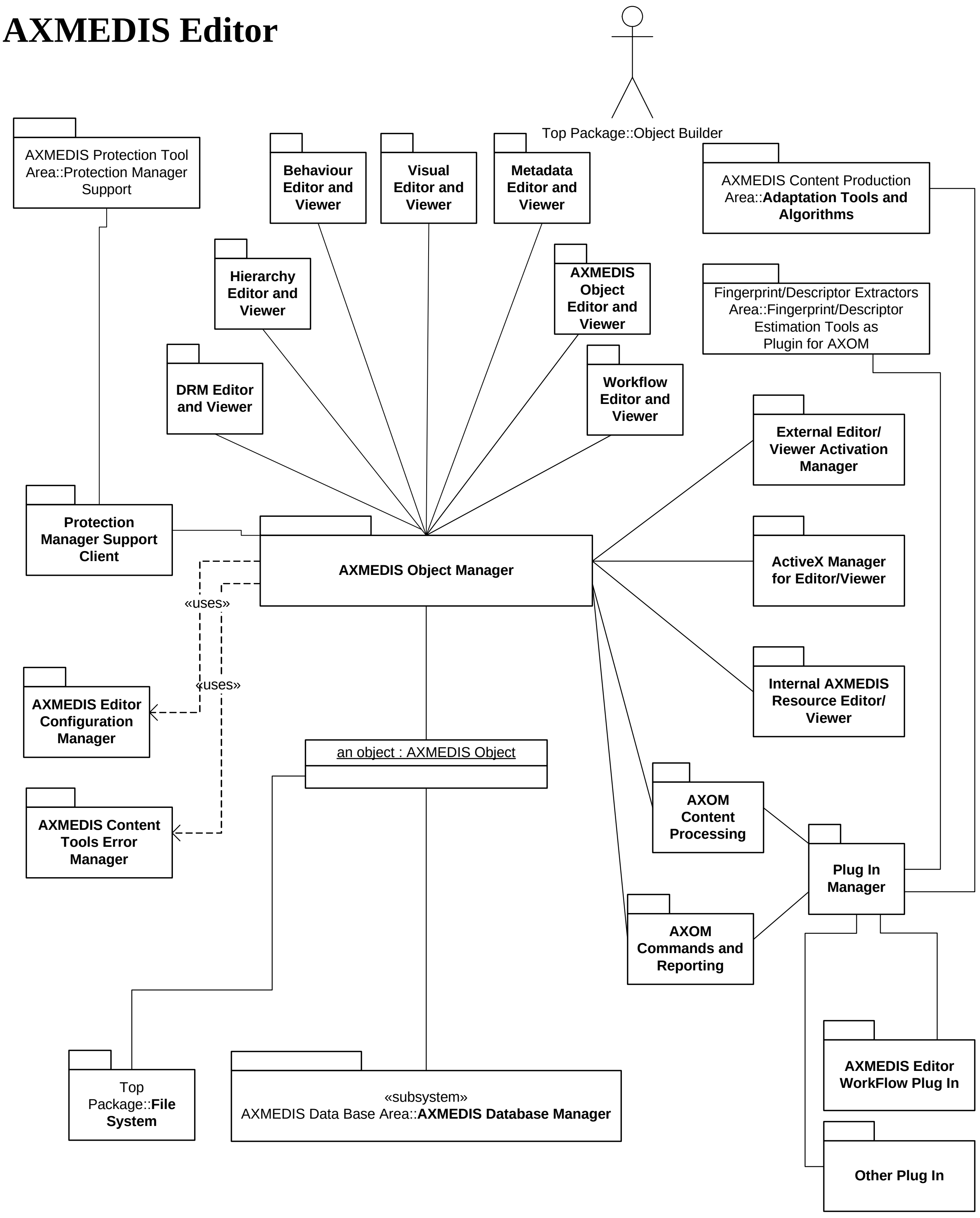
Module/Tool Profile		
AXMEDIS Editor		
Responsible Name	Bellini	
Responsible Partner	DSI	
Status (proposed/approved)		
Implemented/not implemented	Implemented	
Status of the implementation	80%	
Executable or Library/module (Support)	Executable	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Microsoft Windows, Linux, MACOS X	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Applications/axeditor/source https://cvs.axmedis.org/newrepos/Framework/source/axeditorlib	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/axeditor/bin	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any	NA	
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	yes	
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved	-- --	
Major pending requirements		
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section

Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

4.1 General Description of the Module

AXMEDIS Editor will support MPEG-21 with related composition and the nesting of levels; it will support the navigation in MPEG-21 objects. Drag and drop mechanisms will be used to create objects. Moreover, plug-ins will be developed for enabling the MPEG-21 approach in other content editors. Vice versa, AXMEDIS Editor shall use ActiveX to call and use players for proprietary media formats that will be possible through MIME type interpretation.

AXMEDIS Editor


Top Package::Object Builder
AXMEDIS Protection Tool Area::Protection Manager Support
Behaviour Editor and Viewer
Visual Editor and Viewer
Metadata Editor and Viewer
AXMEDIS Content Production Area::Adaptation Tools and Algorithms
Hierarchy Editor and Viewer
AXMEDIS Object Editor and Viewer
Fingerprint/Descriptor Extractors Area::Fingerprint/Descriptor Estimation Tools as Plugin for AXOM
DRM Editor and Viewer
Workflow Editor and Viewer
External Editor/ Viewer Activation Manager
Protection Manager Support Client
AXMEDIS Object Manager
ActiveX Manager for Editor/Viewer
«uses»
«uses»
AXMEDIS Editor Configuration Manager
Internal AXMEDIS Resource Editor/ Viewer
an object : AXMEDIS Object
AXMEDIS Content Tools Error Manager
AXOM Content Processing
Plug In Manager
AXOM Commands and Reporting
AXMEDIS Editor WorkFlow Plug In
Top Package::File System
«subsystem» AXMEDIS Data Base Area::AXMEDIS Database Manager
Other Plug In

AXMEDIS Editor shall manipulate AXMEDIS object in respect of DRM which has been granted to the user on that object, e.g. a user could hold an object on which he hold play/view grants while he has not copy, move or other (manipulation) grants; in such a situation AXMEDIS Editor should stop every attempts of modifying the object by the user.

AXMEDIS Editor includes (or use) the following modules:

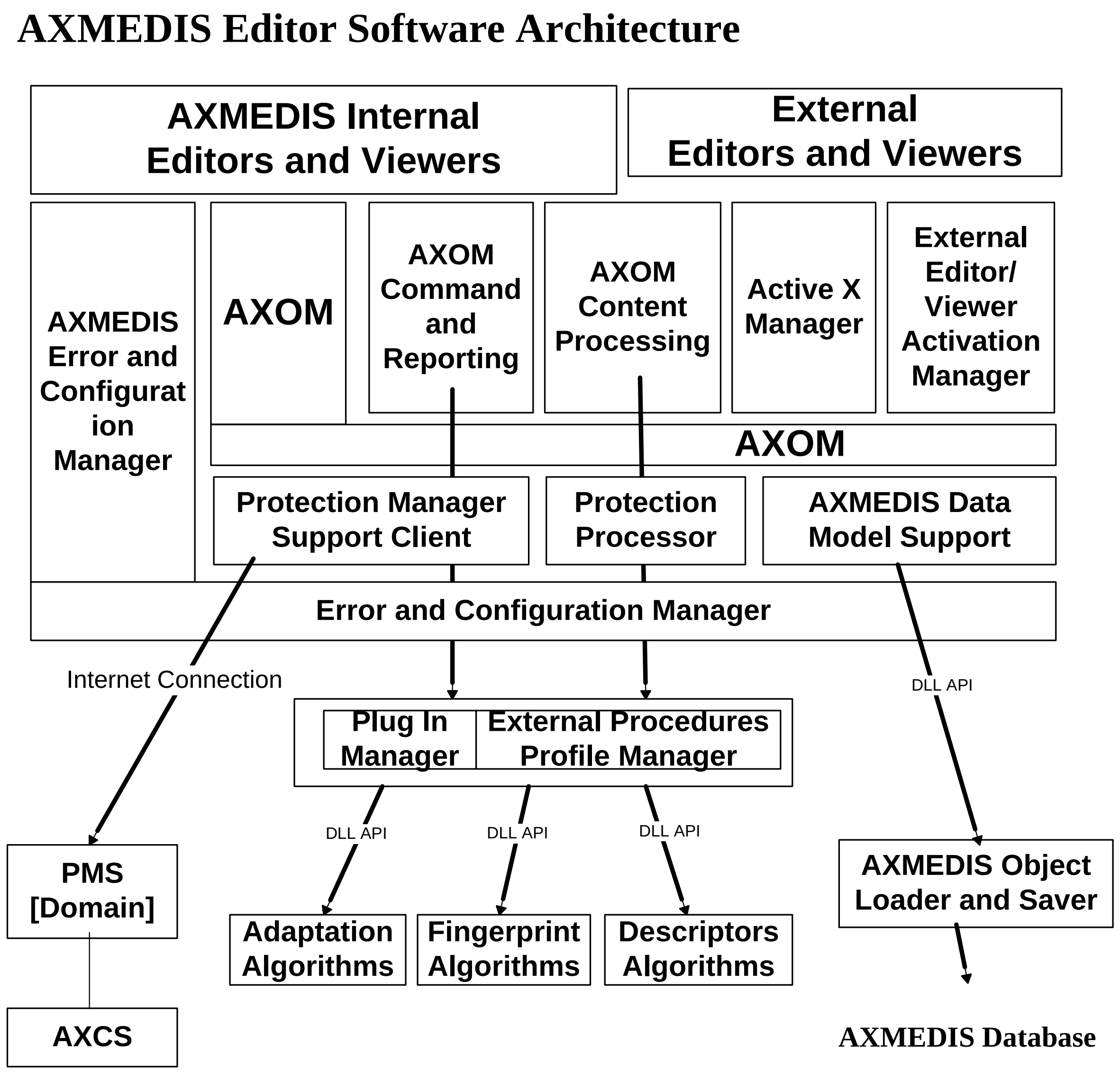
- **AXMEDIS Object Manager** – an AXMEDIS object model container wrapped for secure AXMEDIS object content manipulation. See Section 9;
- A set of viewer/editor for rendering or manipulating the information contained into the AXMEDIS Object Manager and model;
    - **Hierarchy Editor and Viewer** – a view for visualizing and modifying the structure of a document in terms of elements and their parent-child relationship. Hierarchy Editor/Viewer is the main entry-point to interact with the object and parts thereof, to open other kind of editor/viewer, etc… ;
    - **Metadata Editor and Viewer -** a view to edit and view metadata associated with the object..
    - **Visual Editor and Viewer -** a view for editing the visual rendering of the object as SMIL content.
    - **Behaviour Editor and Viewer** – a view for editing the behaviour of the object, allowing to create MPEG21-DIP Methods that will be stored in the AXMEDIS Object.
    - **Protection Editor and Viewer** – a view to edit/view the protection tools (e.g. encryption algorithm) used to protect the object and define the protection parameters (e.g. encryption keys) used for the content protection.
    - **DRM Editor and Viewer,** a view for editing the PAR (Potential Available Rights) associated with the object. From the DRM editor the license editor can be launched to create the licenses that will be stored on the PMS.
    - **Workflow Editor and Viewer -** a view to see the status of the object with respect to workflow management.
- **Protection Manager Support Client** – a set of functionalities to verify the protection and the rights (DRM), for the content contained into the AXMEDIS Object Manager. It contacts Protection Manager Support that in turn contacts AXCS, for certification, registration, accounting, etc.
- **External Editor/Viewer Activation Manager** and relative external application plug-ins – gives, to AXMEDIS Editor, the capability of viewing/modifying resources by using external application which have not ActiveX/COM interface. It has a list of possible external applications which are compliant with AXMEDIS protection model.
- **ActiveX Manager for Editor/Viewer** and relative ActiveX application plug-ins – gives, to AXMEDIS Editor, the capability of viewing/modifying resources by using external application which have ActiveX/COM interface; See Part B.
- **Internal AXMEDIS Resource Editor/Viewer** – a set of editor/viewer built-in AXMEDIS Editor which guarantees a range of basic behaviors, for example, audio player, video player, doc viewer, etc.
- **AXMEDIS Editor Configuration Manager** – is the responsible for configuration storage and maintaining of any AXMEDIS Editor or AXMEDIS client.
- **AXMEDIS Content Tools Error Manager** – is a support for managing errors in the area of content processing, editing, formatting, etc.
- **AXOM Content processing --** This plug in interface allows to demand content processing to external algorithms. It has to allow the presentation of the some functionalities of the algorithms also to the user, e.g. fingerprint extractors, for Digital Item Adaptation, etc.; This interface for plug in is mainly usable for demanding content processing from out side.
- **AXOM Commands and Reporting –** This plug-in interface allows to control the action of the AXMEDIS Object Manager and to send messages and controls outside.
- **Plug-in Manager** – allows the use of external plug-in which can be used for accomplishing various tasks (e.g., workflow plug-ins). These plug ins have to be certified in some manner to guarantee the safeness of their environment. The communication on these plug-in has to be performed in some protected manner since the content if going to be processed by them.

Further AXMEDIS Editor specifications are certainly:

- Each configurable AXMEDIS Editor modules shall conform to AXMEDIS Editor Configuration Manager requirements. Moreover, each configurable AXMEDIS Editor modules shall own a default settings set in order to work also on AXMEDIS Editor Configuration Manager fault or absence;

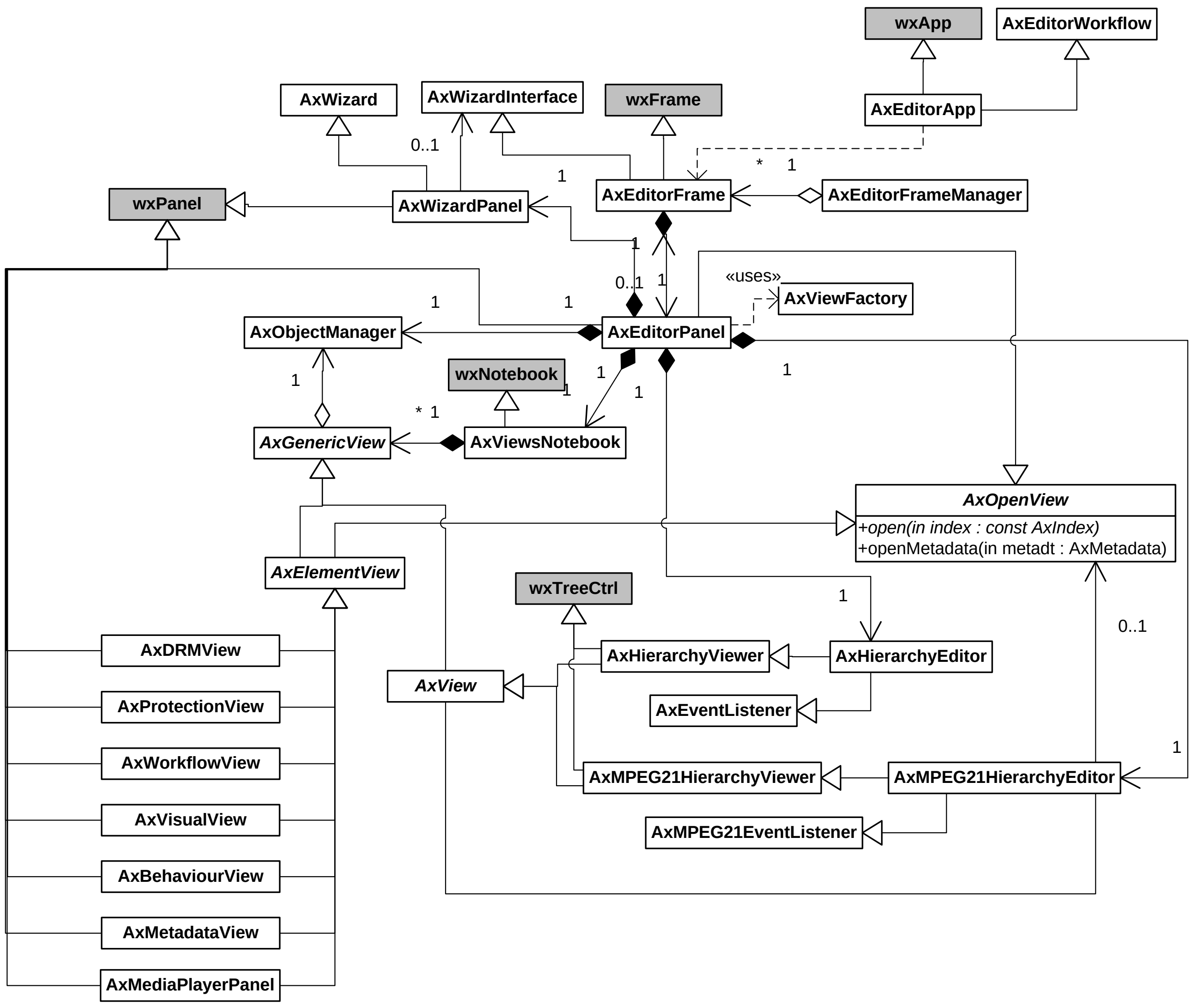
In this section will be analyzed only those modules which belong to AXMEDIS Editor area, the others will be analyzed in the respective sections.

4.1.1 Software Architecture


AXMEDIS Editor Software Architecture
AXMEDIS Internal Editors and Viewers
External Editors and Viewers
AXMEDIS Error and Configuration Manager
AXOM
AXOM Command and Reporting
AXOM Content Processing
Active X Manager
External Editor/ Viewer Activation Manager
AXOM
Protection Manager Support Client
Protection Processor
AXMEDIS Data Model Support
Error and Configuration Manager
Internet Connection
DLL API
Plug In Manager
External Procedures Profile Manager
DLL API
DLL API
DLL API
PMS [Domain]
AXCS
Adaptation Algorithms
Fingerprint Algorithms
Descriptors Algorithms
AXMEDIS Object Loader and Saver
AXMEDIS Database

4.2 Module Design in terms of Classes

The following is the class diagram for the AXMEDIS Editor:



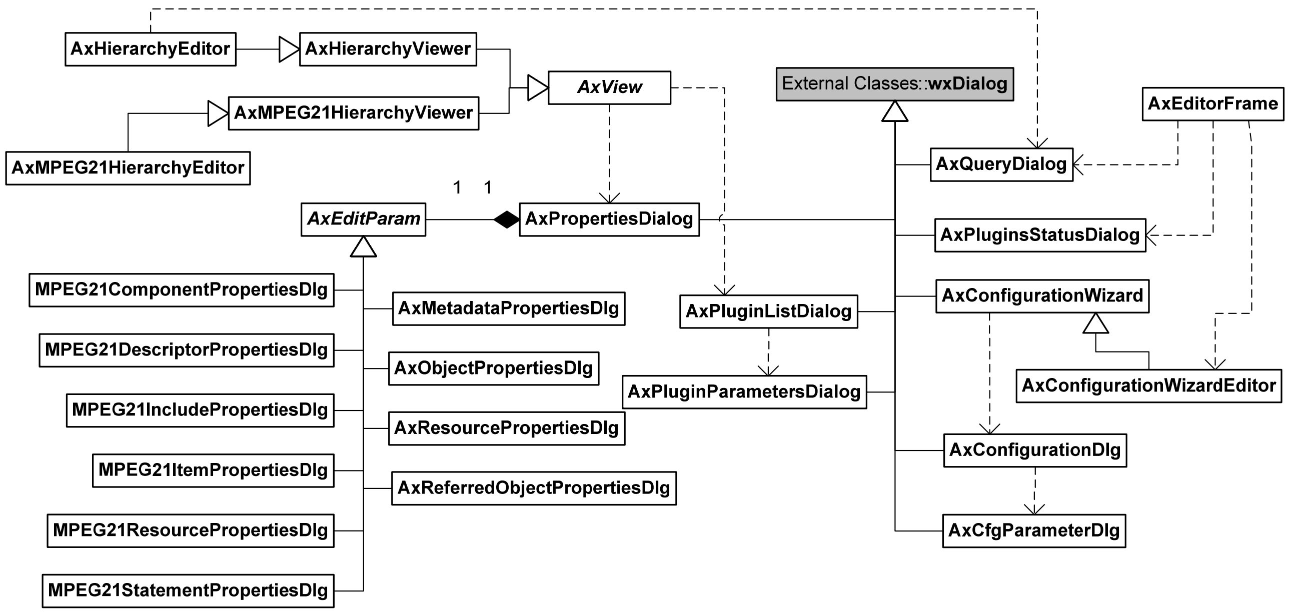
where:

- class *AxEditorApp* is the AXMEDIS Editor application, it creates the first *AxEditorFrame* and it receives the messages coming from the Workflow Manger using the workflow plugin.
- class *AxEditorFrame* is the main frame, it contains an *AxEditorPanel*
- class *AxEditorFrameManager* is a singleton class where each *AxEditorFrame* registers itself on creation. It allows to know which frames are currently open.
- class *AxOpenView* is an interface called when an element of an AXMEDIS Object have to be opened
- class *AxEditorPanel* contains:
  - o an *AxObjectManager* used to manipulate the AXMEDIS object;
  - o an *AxHierarchyEditor* and an *AxMPEG21HierarchyEditor*, tree controls used to manipulate the object structure
  - o a set of views open for the object contained in *AxViewsNotebook*
  - o an *AxWizardPanel* allowing the user to speed up some common operations

  AxEditorPanel uses the *AxViewFactory* object to create the Views, and it is derived from *AxOpenView* to open the view requested from the two hierarchy editors;
- class *AxViewFactory* is a singleton class used to create the different views
- class *AxViewsNotebook* is used to contain different views of the object that can be hosted inside a Notebook, they can be: *AxDRMView*, *AxProtectionView*, *AxMetadataView*, *AxWorkflowView*, *AxVisualView*, or *AxBehaviourView*.
- Abstract class *AxGenericView* represents a view on an *AxObjectManager*, it is derived from Ax*MPEG21EventListener*, *AxEventListener* to receive events when the object is manipulated.

- Abstract class *AxView* represents a view on the whole object (like a hierarchy view) and *AxElementView* is a view on a specific element of the object.
- classes *AxHierarchyViewer* and *AxMPEG21HierarchyViewer* (derived from *AxView*) allow to view the AXMEDIS object as AXMEDIS/MPEG21 tree structures, they reference an object implementing the *AxOpenView* interface to open the view for an element of the object (e.g. via double click)
- classes *AxHierarchyEditor* and *AxMPEG21HierarchyEditor* are extensions of the viewers allowing manipulation of the object structure
- classes *AxDRMView*, *AxProtectionView*, *AxMetadataView*, *AxWorkflowView*, *AxVisualView* and *AxBehaviourView* are specific views on an element
- class *AxMediaPlayerPanel* (detailed in another diagram) allow to view/manipulate digital resources

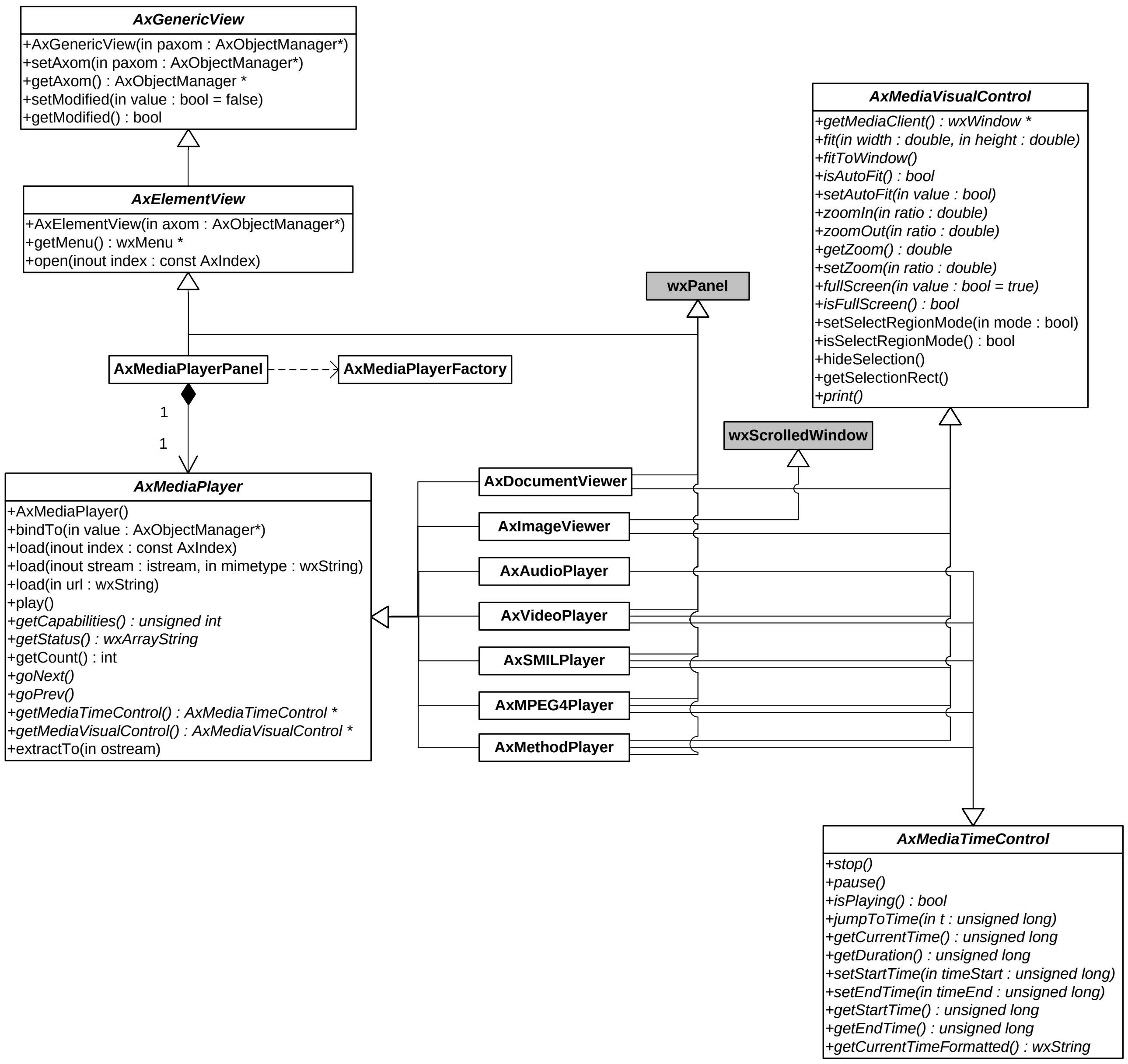
the following class diagram is on the Dialogs used:



where:

- class *AxQueryDialog* is used to make queries on the database, it is used from the *AxEditorFrame* to open an object from DB or from the *AxHierarchyEditor* to embed an AXMEDIS object coming from the DB.
- class *AxPluginStatusDialog* it is used from the *AxEditorFrame* to show the plugins installed
- class *AxPropertiesDialog* it is used by the *AxView* to show/edit the properties of the MPEG21/AXMEDIS elements, the AxPropertiesDialog is instantiated with the specific *MPEG21…PropertiesDlg* or *Ax…PropertiesDlg* object depending on the kind of object that have to be edited;
- class *AxPluginDialog* is used from the *AxView* to use a content processing plugin on a resource, the dialog shows the plugins that can be used and using the *AxParameterDialog* the plugin parameters can be set and the plugin called.
- class *AxConfigurationWizardEditor* is used form the AxEditorFrame to edit the configuration parameters, this configuration dialog helps in setting up the common parameters (AXMEDIS Database host name and port, AXCS host name, etc.), using the Advanced button the generic *AxConfigurationDlg* class is instantiated allowing the complete inspection/edit of all the configuration parameters.

The following diagram details the relationships among the classes used to display/manipulates digital resources:

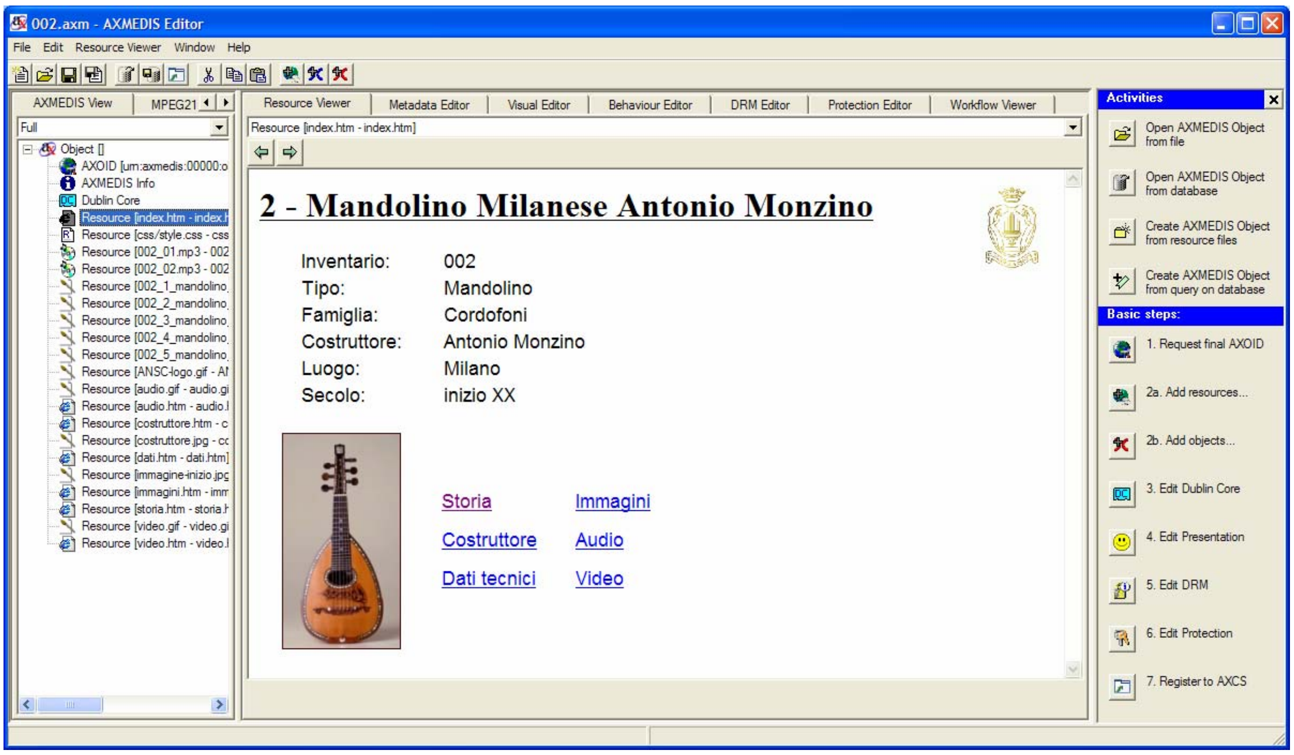

AxGenericView
+AxGenericView(in paxom : AxObjectManager*)
+setAxom(in paxom : AxObjectManager*)
+getAxom() : AxObjectManager *
+setModified(in value : bool = false)
+getModified() : bool
AxElementView
+AxElementView(in axom : AxObjectManager*)
+getMenu() : wxMenu *
+open(inout index : const AxIndex)
wxPanel
AxMediaVisualControl
+getMediaClient() : wxWindow *
+fit(in width : double, in height : double)
+fitToWindow()
+isAutoFit() : bool
+setAutoFit(in value : bool)
+zoomIn(in ratio : double)
+zoomOut(in ratio : double)
+getZoom() : double
+setZoom(in ratio : double)
+fullScreen(in value : bool = true)
+isFullScreen() : bool
+setSelectRegionMode(in mode : bool)
+isSelectRegionMode() : bool
+hideSelection()
+getSelectionRect()
+print()
AxMediaPlayerPanel
AxMediaPlayerFactory
1
1
wxScrolledWindow
AxMediaPlayer
+AxMediaPlayer()
+bindTo(in value : AxObjectManager*)
+load(inout index : const AxIndex)
+load(inout stream : istream, in mimetype : wxString)
+load(in url : wxString)
+play()
+getCapabilities() : unsigned int
+getStatus() : wxArrayString
+getCount() : int
+goNext()
+goPrev()
+getMediaTimeControl() : AxMediaTimeControl *
+getMediaVisualControl() : AxMediaVisualControl *
+extractTo(in ostream)
AxDocumentViewer
AxImageViewer
AxAudioPlayer
AxVideoPlayer
AxSMILPlayer
AxMPEG4Player
AxMethodPlayer
AxMediaTimeControl
+stop()
+pause()
+isPlaying() : bool
+jumpToTime(in t : unsigned long)
+getCurrentTime() : unsigned long
+getDuration() : unsigned long
+setStartTime(in timeStart : unsigned long)
+setEndTime(in timeEnd : unsigned long)
+getStartTime() : unsigned long
+getEndTime() : unsigned long
+getCurrentTimeFormatted() : wxString

where:

- abstract class *AxMediaPlayer* represents a media player, it is derived in the specific classes *AxDocumentViewer*, *AxImageViewer*, *AxAudioPlayer*, *AxVideoPlayer*, *AxSMILPlayer*, *AxMPEG4Player* and *AxMethodPlayer* used to show the specific resources.
- abstract class *AxMediaVisualControl* is used to control aspect related to the visual rendering
- abstract class *AxMediaTimeControl* is used to control the execution of resources dealing with time.
- class *AxMediaPlayerPanel* contains the UI controls (play/stop/pause buttons, status text, seek slider) to control the object (derived from *AxMediaPlayer*) used to display a resource
- class *AxMediaPlayerFactory* creates the specific class for displaying a resource on the basis of the mimetype

4.3 User interface description

The following is a possible user interface for the AXMEDIS Editor, a Frame is used for each object opened. Inside the frame, three area are present on the left a tree view representing the object structure is presented in the center the different views of the object are hosted and on the right the wizard allow to speddup some basic operations.
The AXMEDIS Hierarchy can be shown also as it will be visible from the End User by selecting the specific Combo Box.


002.axm - AXMEDIS Editor
File Edit Resource Viewer Window Help
AXMEDIS View
MPEG21
Resource Viewer
Metadata Editor
Visual Editor
Behaviour Editor
DRM Editor
Protection Editor
Workflow Viewer
Activities
Open AXMEDIS Object from file
Open AXMEDIS Object from database
Create AXMEDIS Object from resource files
Create AXMEDIS Object from query on database
Basic steps:
1. Request final AXOID
2a. Add resources...
2b. Add objects...
3. Edit Dublin Core
4. Edit Presentation
5. Edit DRM
6. Edit Protection
7. Register to AXCS
Full
Object []
AXOID [urn:axmedis:00000:o
AXMEDIS Info
Dublin Core
Resource [index.htm - index.h
Resource [css/style.css - css
Resource [002_01.mp3 - 002
Resource [002_02.mp3 - 002
Resource [002_1_mandolino.
Resource [002_2_mandolino.
Resource [002_3_mandolino.
Resource [002_4_mandolino.
Resource [002_5_mandolino.
Resource [ANSC-logo.gif - AI
Resource [audio.gif - audio.gi
Resource [audio.htm - audio.l
Resource [costruttore.htm - c
Resource [costruttore.jpg - cc
Resource [dati.htm - dati.htm]
Resource [immagine-inizio.jpg
Resource [immagini.htm - imm
Resource [storia.htm - storia.h
Resource [video.gif - video.gi
Resource [video.htm - video.l
Resource [index.htm - index.htm]

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 Editor

The top-level menus are:

File	Description
New	creates a new AXMEDIS object in a new window
Open…	loads an AXMEDIS object in a new window
Open from database…	opens an object from the database
Save	saves the object to disk
Save as…	saves the object as AXM format (XML)
Save as MPEG21…	saves the object with MPEG21 file format
Save on database	saves the object into the database
Close	closes the current window
Notify workflow activity completion	notifies an workflow activity completion
Configuration…	opens the configuration editor
Plugins…	shows which plugins have been found
Recent files	allows to select a recently opened file
Exit	closes the application

Edit	Description
… depends on the item selected…	operations that are allowed on the currently selected item in the hierarchy view

View	Description
… view specific menu…	a menu used for the view currently selected

Window	Description
…	contains the list of windows currently opened

Help	**Description**
Guide to AXMEDIS Editor…	Opens the guide to the AXMEDIS Editor
Show Activities	Hides/show the activity wizard
Register User…	Uses Internet Explorer to register the user on the AXCS
Import User Certificate…	Allows to import the user certificate sent from AXCS
Tool certification…	Allows to certify the tool for the user
Copy User ID	Copies to the clipboard the user ID
About AXMEDIS Editor…	Opens a dialog showing information on the AXMEDIS Editor

Contextual menus are used in the AXMEDIS Hierarchy to perform specific operations on each element of the hierarchy:

Open	open a default view for the element
Open with…	in case more choices are available let the user choose
Properties…	show properties of the element
Cut	cuts the element
Copy	copies the element
Paste	pastes an element after the selected one
Delete	removes the element
Move up	move the element up
Move down	move the element down
Insert	
metadata…	inserts a new metadata element
resource…	inserts a new resource element
object…	inserts a new object
Plugin…	let the user select the operation to be performed on the element

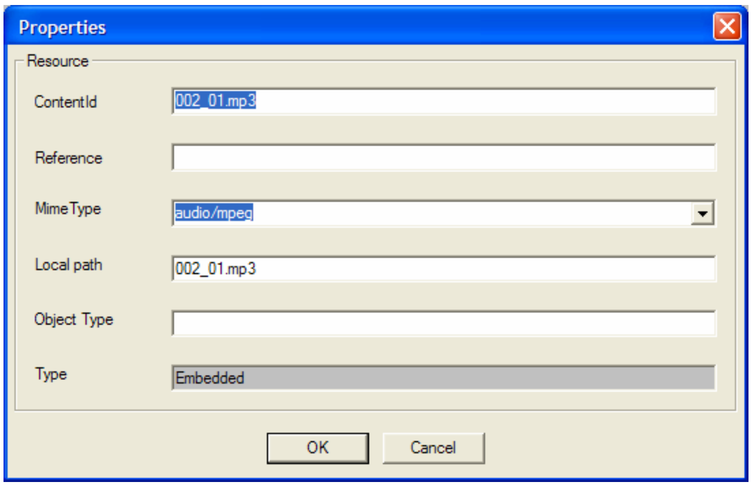
On the MPEG21 Hierarchy specific contextual menus are used:

Open	open a default view for the element
Open with…	in case more choices are available let the user choose
Properties…	show properties of the element
Cut	cuts the element
Copy	copies the element
Paste	pastes an element after the selected one
Delete	removes the element without putting it in the clipboard
Move up	move the element up
Move down	move the element down
Insert	
Descriptor	inserts a new descriptor
Statement	inserts a new statement
Item	inserts a new item
Component	inserts a new component
Resource	inserts a new resource
Container	inserts a new container
Reference	inserts a new reference
…	
Plugin…	let the user select the operation to be performed on the element

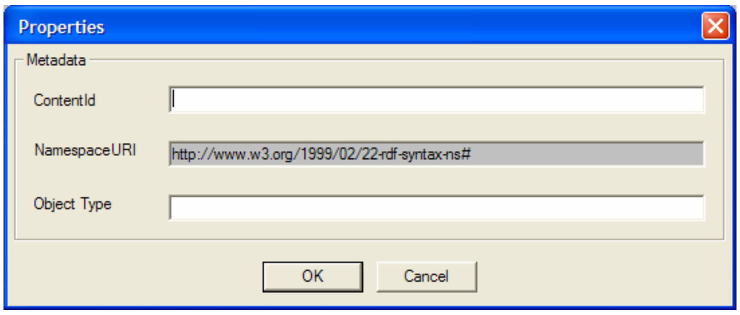
Drag and drop can be used to copy an element from one hierarchy to another hierarchy (of different objects) or to move an element from one point to another (in the same object).

Files (images, video, documents, …) can be dropped inside an object from the file system to become automatically resources belonging to the object.

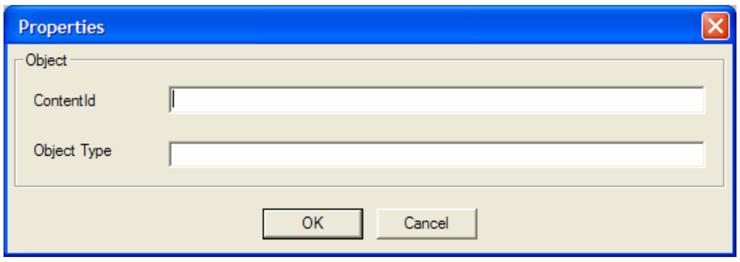
The following are the dialog to edit properties of a resource, a metadata and an AXMEDIS Object:



ContentId	002_01.mp3
Reference	
MimeType	audio/mpeg
Local path	002_01.mp3
Object Type	
Type	Embedded



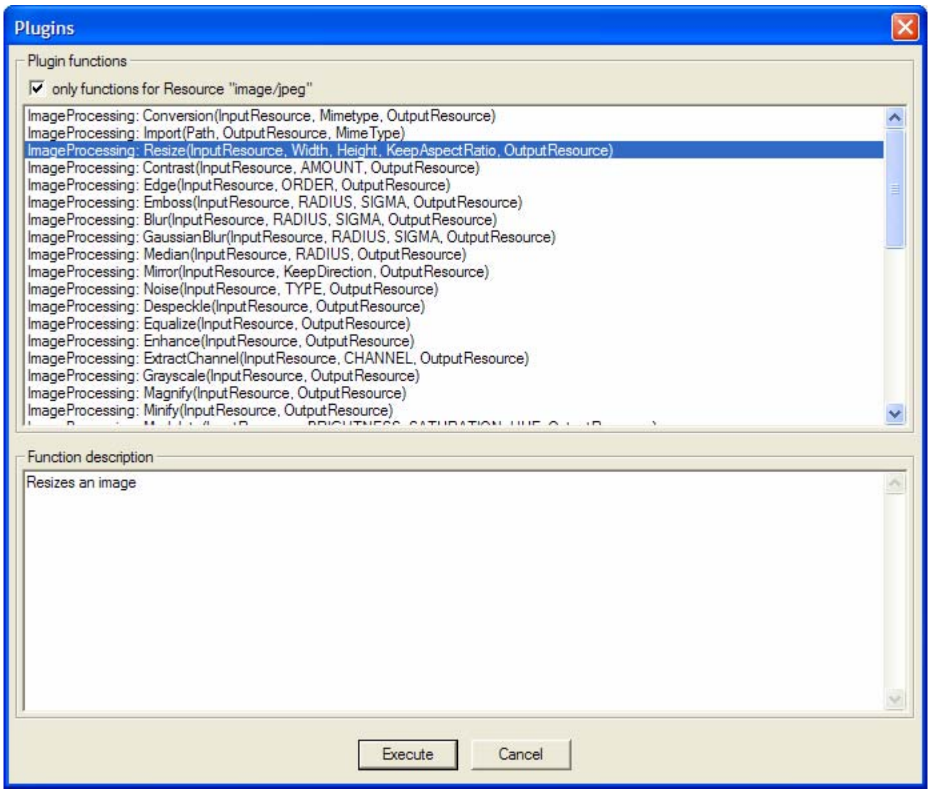
ContentId	
NamespaceURI	http://www.w3.org/1999/02/22-rdf-syntax-ns#
Object Type	



ContentId	
Object Type	

Similar dialogs are present to edit properties of MPEG21 elements (Resource, Component, Item, Description, Statement)

The following is the dialog to select the content processing plugin to be used on a resource:

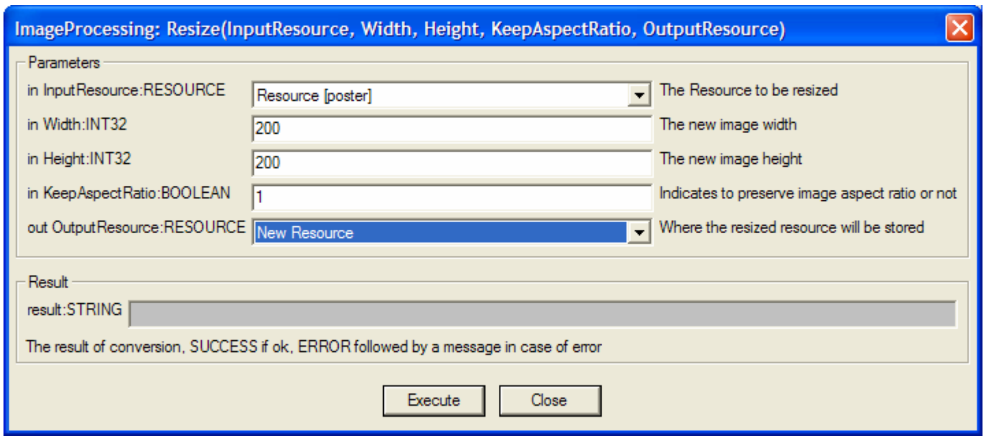


☑ only functions for Resource "image/jpeg"

ImageProcessing: Conversion(InputResource, Mimetype, OutputResource)
ImageProcessing: Import(Path, OutputResource, MimeType)
ImageProcessing: Resize(InputResource, Width, Height, KeepAspectRatio, OutputResource)
ImageProcessing: Contrast(InputResource, AMOUNT, OutputResource)
ImageProcessing: Edge(InputResource, ORDER, OutputResource)
ImageProcessing: Emboss(InputResource, RADIUS, SIGMA, OutputResource)
ImageProcessing: Blur(InputResource, RADIUS, SIGMA, OutputResource)
ImageProcessing: GaussianBlur(InputResource, RADIUS, SIGMA, OutputResource)
ImageProcessing: Median(InputResource, RADIUS, OutputResource)
ImageProcessing: Mirror(InputResource, KeepDirection, OutputResource)
ImageProcessing: Noise(InputResource, TYPE, OutputResource)
ImageProcessing: Despeckle(InputResource, OutputResource)
ImageProcessing: Equalize(InputResource, OutputResource)
ImageProcessing: Enhance(InputResource, OutputResource)
ImageProcessing: ExtractChannel(InputResource, CHANNEL, OutputResource)
ImageProcessing: Grayscale(InputResource, OutputResource)
ImageProcessing: Magnify(InputResource, OutputResource)
ImageProcessing: Minify(InputResource, OutputResource)

Resizes an image

The following is the dialog to provide parameters for the plugin:

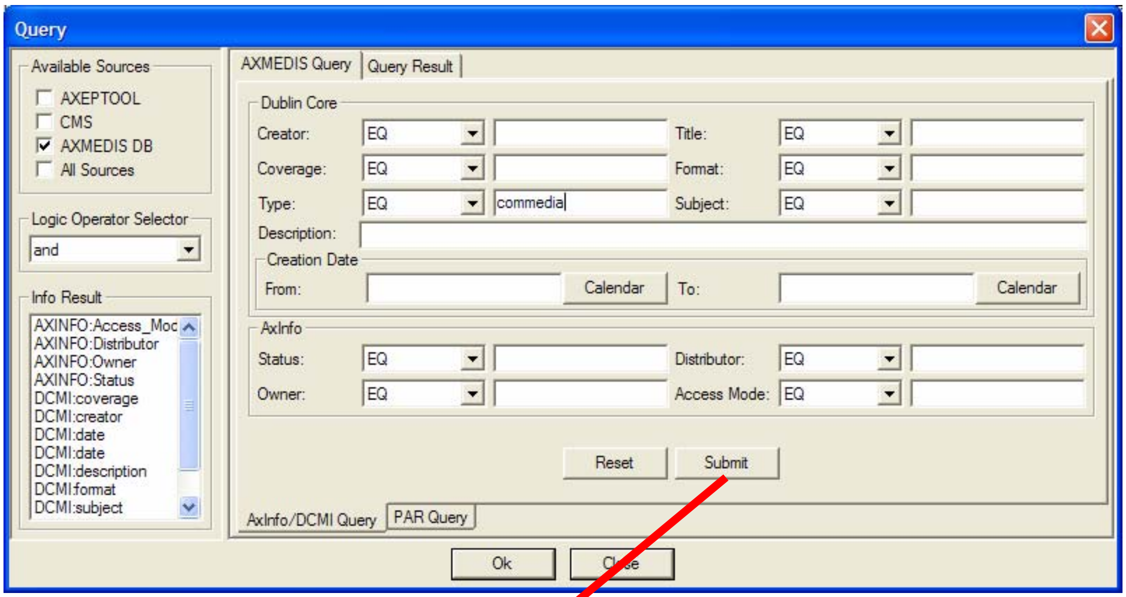

ImageProcessing: Resize(InputResource, Width, Height, KeepAspectRatio, OutputResource)
Parameters
Result
Execute
Close

in InputResource:RESOURCE	Resource [poster]	The Resource to be resized
in Width:INT32	200	The new image width
in Height:INT32	200	The new image height
in KeepAspectRatio:BOOLEAN	1	Indicates to preserve image aspect ratio or not
out OutputResource:RESOURCE	New Resource	Where the resized resource will be stored

result:STRING

The result of conversion, SUCCESS if ok, ERROR followed by a message in case of error

The following is the dialog to query for an object into the AXMEDIS Database:

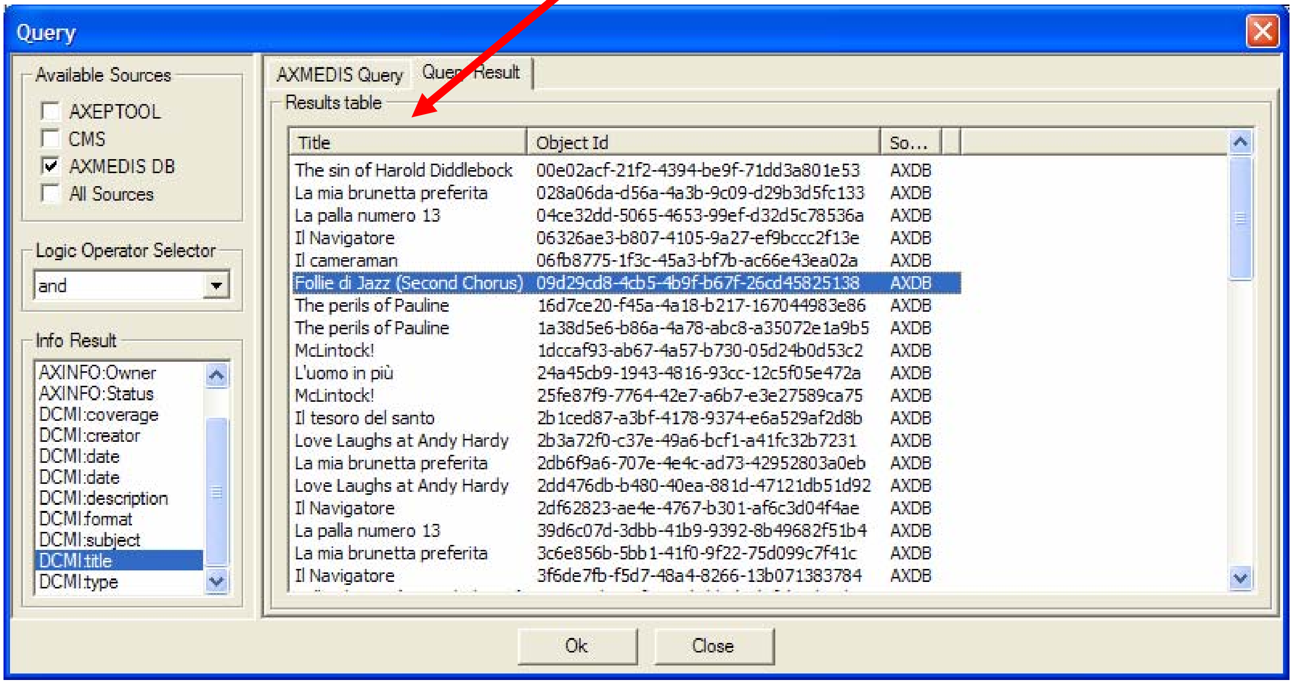


Available Sources: ☐ AXEPTOOL, ☐ CMS, ☑ AXMEDIS DB, ☐ All Sources

Logic Operator Selector: and

Info Result: AXINFO:Access_Moc, AXINFO:Distributor, AXINFO:Owner, AXINFO:Status, DCMI:coverage, DCMI:creator, DCMI:date, DCMI:date, DCMI:description, DCMI:format, DCMI:subject

Creator:	EQ		Title:	EQ	
Coverage:	EQ		Format:	EQ	
Type:	EQ	commedia	Subject:	EQ	
Description:					
From:			To:		
Status:	EQ		Distributor:	EQ	
Owner:	EQ		Access Mode:	EQ	



Available Sources: ☐ AXEPTOOL, ☐ CMS, ☑ AXMEDIS DB, ☐ All Sources

Logic Operator Selector: and

Info Result: AXINFO:Owner, AXINFO:Status, DCMI:coverage, DCMI:creator, DCMI:date, DCMI:date, DCMI:description, DCMI:format, DCMI:subject, DCMI:title, DCMI:type

Title	Object Id	So...
The sin of Harold Diddlebock	00e02acf-21f2-4394-be9f-71dd3a801e53	AXDB
La mia brunetta preferita	028a06da-d56a-4a3b-9c09-d29b3d5fc133	AXDB
La palla numero 13	04ce32dd-5065-4653-99ef-d32d5c78536a	AXDB
Il Navigatore	06326ae3-b807-4105-9a27-ef9bccc2f13e	AXDB
Il cameraman	06fb8775-1f3c-45a3-bf7b-ac66e43ea02a	AXDB
Follie di Jazz (Second Chorus)	09d29cd8-4cb5-4b9f-b67f-26cd45825138	AXDB
The perils of Pauline	16d7ce20-f45a-4a18-b217-167044983e86	AXDB
The perils of Pauline	1a38d5e6-b86a-4a78-abc8-a35072e1a9b5	AXDB
McLintock!	1dccaf93-ab67-4a57-b730-05d24b0d53c2	AXDB
L'uomo in più	24a45cb9-1943-4816-93cc-12c5f05e472a	AXDB
McLintock!	25fe87f9-7764-42e7-a6b7-e3e27589ca75	AXDB
Il tesoro del santo	2b1ced87-a3bf-4178-9374-e6a529af2d8b	AXDB
Love Laughs at Andy Hardy	2b3a72f0-c37e-49a6-bcf1-a41fc32b7231	AXDB
La mia brunetta preferita	2db6f9a6-707e-4e4c-ad73-42952803a0eb	AXDB
Love Laughs at Andy Hardy	2dd476db-b480-40ea-881d-47121db51d92	AXDB
Il Navigatore	2df62823-ae4e-4767-b301-af6c3d04f4ae	AXDB
La palla numero 13	39d6c07d-3dbb-41b9-9392-8b49682f51b4	AXDB
La mia brunetta preferita	3c6e856b-5bb1-41f0-9f22-75d099c7f41c	AXDB
Il Navigatore	3f6de7fb-f5d7-48a4-8266-13b071383784	AXDB

4.4 Technical and Installation information

References to other major components needed	• AXMEDIS Object Manger • AXMEDIS Plugin Manager • AXMEDIS Content Processing Plugins
Problems not solved	
Configuration and execution context	

4.5 Draft User Manual and Examples of Usage

See section 4.3 User Interface Description above and for more details, see AXMEDIS User Manual DE5-0-1-2 for AXTools.

4.6 Integration and compilation issues

None

4.7 Configuration Parameters

Config parameter	Possible values

4.8 Errors reported and that may occur

Error code	Description and rationales

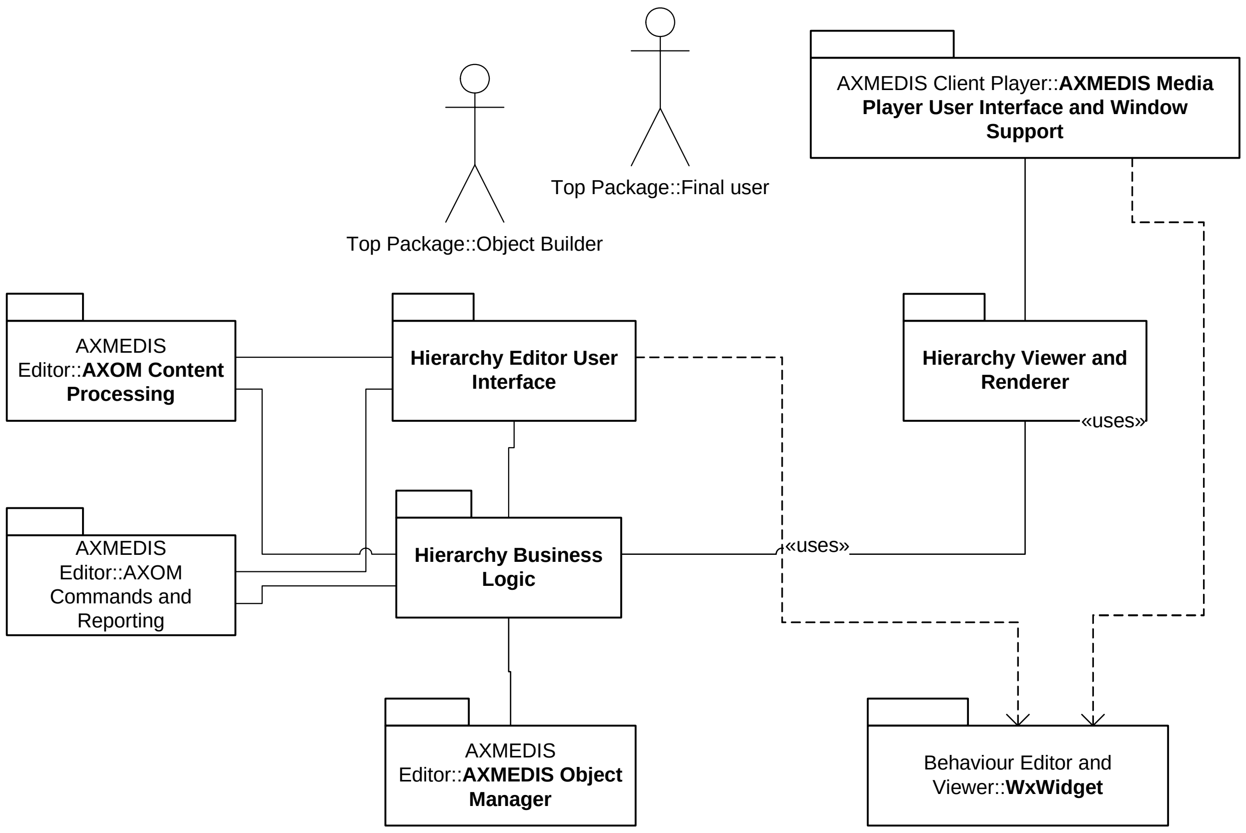
5 Module - Hierarchy Editor and Viewer (DSI)

Module/Tool Profile	
Hierarchy Editor and Viewer	
Responsible Name	Bellini
Responsible Partner	DSI
Status (proposed/approved)	
Implemented/not implemented	Implemented
Status of the implementation	
Executable or Library/module (Support)	Library
Single Thread or Multithread	
Language of Development	C++
Platforms supported	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/axeditor/hierarchy_view
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/axeditor/bin
Reference to the AXFW location of the demonstrator executable tool for public	

download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any	na	
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	yes	
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved		
Major pending requirements	undo/redo support	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
AxHierarchyView	C++	wxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets	wxWidgets 2.4.2	LGPL

5.1 General Description of the Module

Hierarchy Editor and Viewer


Top Package::Object Builder
Top Package::Final user
AXMEDIS Client Player::AXMEDIS Media Player User Interface and Window Support
AXMEDIS Editor::AXOM Content Processing
Hierarchy Editor User Interface
Hierarchy Viewer and Renderer
«uses»
AXMEDIS Editor::AXOM Commands and Reporting
Hierarchy Business Logic
«uses»
AXMEDIS Editor::AXMEDIS Object Manager
Behaviour Editor and Viewer::WxWidget

Hierarchy View should show hierarchical relationship among object subparts. Because of linearity of MPEG-21/AXMEDIS relationship among components (i.e. each component has one, and only one, father component), tree-like view (similar to the one of explorer) is the most suitable solution. Such view should permit cut and paste operation and, moreover, it should permit specific operations (through contextual-menu usage) on showed elements.

- Hierarchy View shall allow cut and paste actions. Both drag-and-drop and keyboard functionalities should be developed to make the user interface more friendly;
- Hierarchy View shall allow use of mouse right-click on showed objects to permit contextual menu usage. Such menus could be customized upon object types in order to permit implementation of specific (object-type determined) actions, e.g. right-clicking on a document should permit to open (in a readable format) it as long as the same action upon a mp3 should permit value extraction of IDv3 tags.
- Hierarchy View shall expose a functions which permits to add functionalities on the base selected element type.
- Hierarchy View shall represent each object by a specific icon. Some of such icons should be obtained by OS file association and the others will be drawn by developers.

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

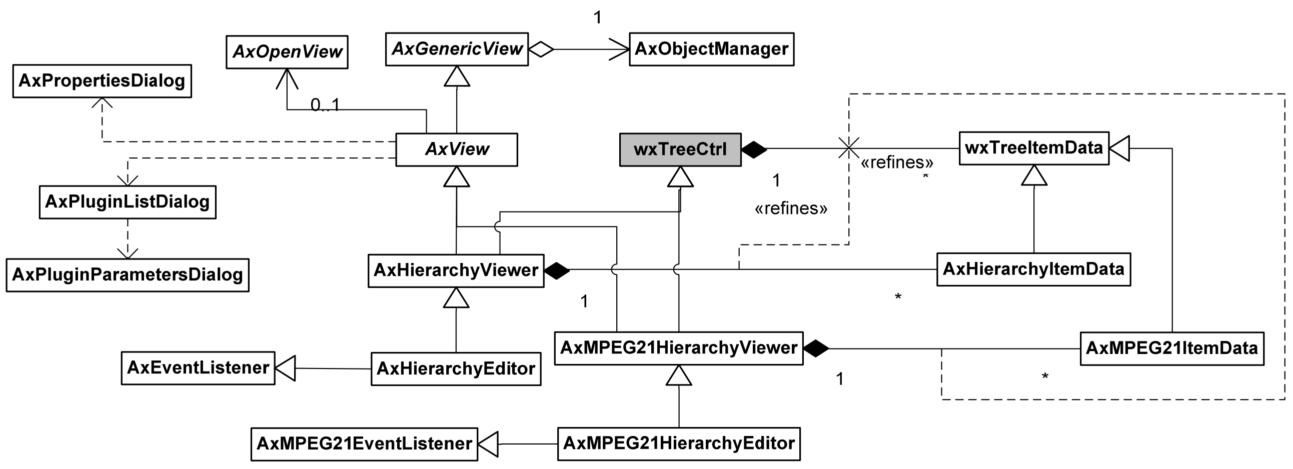
- drag & drop elements to move elements inside the object or from an object to another one
- drag & drop a file on the hierarchy to automatically add a resource
- copy & paste

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 business logic under the hierarchy interface will basically cover four action on the hierarchy element
- Add – Insert Before – Insert After
- Remove
- Cut
- Copy
- Paste
- Drag and drop
- Expand
- Collapse
- Activation of other view

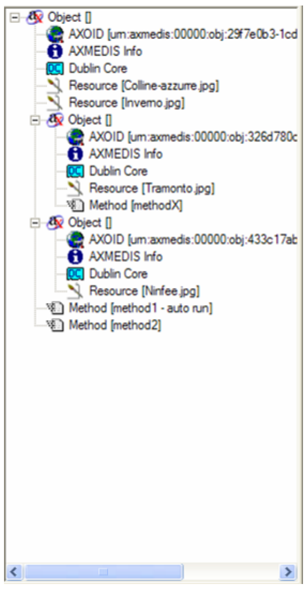
All the commands will be mapped on several command instances:
- The Add – Insert Before – InsertAfter generates a new **CommandAdd** class in order to submit such instance to the execute method exposed by AXOM, these actions will set differently the command attribute which determine the addition target in the data model, once the command is processed by the AXOM the needed rights are checked by the PMS client and the needed operation executed in the model (i.e. adapt?, enhance?, enlarge).
- The Remove operates in a similar manner requesting the execution of CommandDelete.
- The functions Cut, Copy, Paste, Drag and Drop are managed by the application clipboard and after the event processing end in a CommandMove or CommandCopy which will check their specific rights (i.e. extract).
- The function Expand needs to retrieve data model information in order to show the list of the sub-items in the hierarchy, this requests the execution of **CommandGetChildren** which after the right check (explore) allows the view to retrieve children data.

5.2 Module Design in terms of Classes



5.3 User interface description

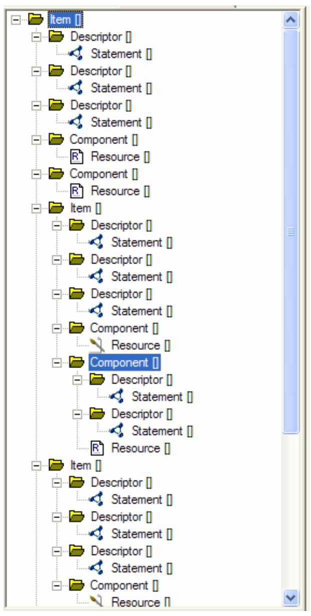
The following is an example of AXMEDIS Hierarchy:



```
Object []
    AXOID [urn:axmedis:00000:obj:29f7e0b3-1cd
    AXMEDIS Info
    Dublin Core
    Resource [Colline-azzurre.jpg]
    Resource [Inverno.jpg]
    Object []
        AXOID [urn:axmedis:00000:obj:326d780c
        AXMEDIS Info
        Dublin Core
        Resource [Tramonto.jpg]
        Method [methodX]
    Object []
        AXOID [urn:axmedis:00000:obj:433c17ab
        AXMEDIS Info
        Dublin Core
        Resource [Ninfee.jpg]
    Method [method1 - auto run]
    Method [method2]
```

Figure 1 - Example of AXMEDIS hierarchy

The following is an example of MPEG-21 Hierarchy:



```
Item []
    Descriptor []
        Statement []
    Descriptor []
        Statement []
    Descriptor []
        Statement []
    Component []
        Resource []
    Component []
        Resource []
    Item []
        Descriptor []
            Statement []
        Descriptor []
            Statement []
        Descriptor []
            Statement []
        Component []
            Resource []
        Component []
            Descriptor []
                Statement []
            Descriptor []
                Statement []
            Resource []
    Item []
        Descriptor []
            Statement []
        Descriptor []
            Statement []
        Descriptor []
            Statement []
        Component []
            Resource []
```

Figure 2 - Example of MPEG-21 hierarchy

5.4 Technical and Installation information

References to other major components needed	AXMEDIS Object Manager
Problems not solved	•
Configuration and execution context	

5.5 Draft User Manual and Examples of Usage

See section 5.3 User Interface Description above and for more details, see AXMEDIS User Manual DE5-0-1-2 for AXTools.

5.6 Integration and compilation issues

None

5.7 Configuration Parameters

Config parameter	Possible values

5.8 Errors reported and that may occur

Error code	Description and rationales

6 Module DRM Editor and Viewer (UPC)

Module/Tool Profile	
DRM Editor and Viewer	
Responsible Name	Víctor Rodríguez
Responsible Partner	FUPF
Status (proposed/approved)	Approved
Implemented/not implemented	Implemented
Status of the implementation	First version
Executable or Library/module (Support)	Library and executable
Single Thread or Multithread	Needs to be compiled with multithread runtime libraries
Language of Development	C++
Platforms supported	PC (Windows)
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Applications/drmeditorviewer
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/drmeditorviewer/bin/drm_editor_viewer/win32
Reference to the AXFW location of the demonstrator executable tool for public download	http://www.axmedis.org/documenti/view_documenti.php?doc_id=1601
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any	
Test cases (present/absent)	
Test cases location	https://cvs.axmedis.org/newrepos/Applications/drmeditorviewer/doc/test
Usage of the AXMEDIS configuration manager (yes/no)	Yes
Usage of the AXMEDIS Error Manager (yes/no)	No
Major Problems not solved	

Major pending requirements		
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
XML based (MPEG-21 REL)		
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
DRMView	C++	wxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
	WxWidgets	
	OpenSSL	
	Xerces-C	

6.1 General Description of the Module

DRM Editor and Viewer is the name for a software to express rights information.

6.1.1 The DRM Editor and Viewer forms

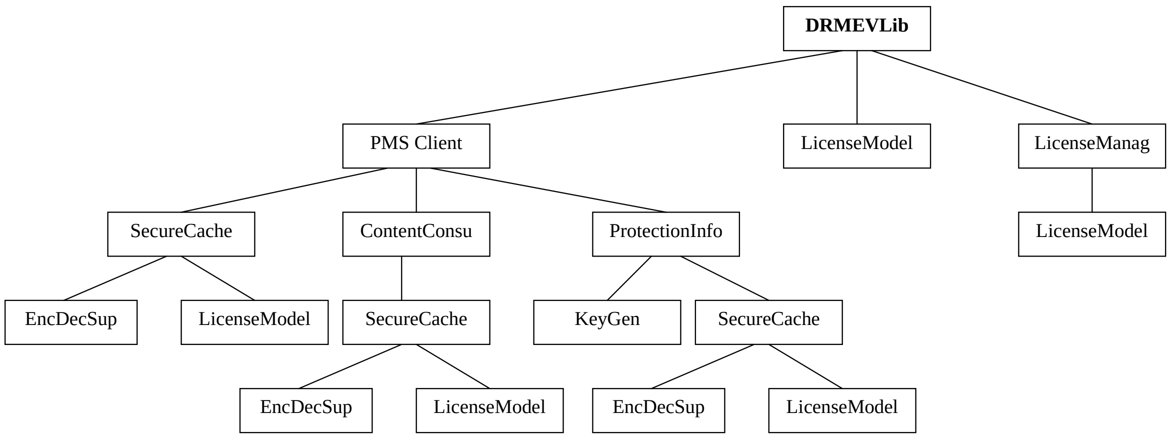
It is a highly configurable software, which enables the application to take several very different forms (what could not be properly described in the table afore). The possibilities are:

- **Standalone / Module**. It can be an independent application or a module of other applications (Axmedis Editor)
- **Editor / Viewer**. It can act as a mere viewer of information or can allow editing.
- **Tree view / User friendly view.** The user interface can be simple and complete, or user friendly (but with some limitations)

The combination of this 3 degrees of freedom, plus Debug and Release versions, would make up to 16 different versions of the binaries!! Therefore not all the binary versions are provided, and in particular the simple tree view has been initially disabled by default. It has to be considered though, that at any convenient time it can be recovered.

6.1.2 Architecture

The DRM Editor and Viewer libraries are based in a important number of libraries. The next figure shows the dependencies with other Axmedis libraries for the case of the Standalone version.
For the integrated version, further relationship with other Axmedis modules is present.



6.1.3 Functionalities

- (E,V). It permits to visualize the license, in a browsable tree structure.
- (E,V). The tree structure can be easily embedded in a wxWindows application.
- (E,V). Elements of the license are represented in the leaves of the tree, in a user readable way.
- (E,V). The tree to be visualized can be expanded or collapsed at once or individually.
- (E,V). Licenses can be open from a XML file.
- (E,V) A panel is displayed in the right side of the tree, showing partially or totally the information from the tree in an alternative format
- (E,V) The right panel can be hidden, reverting the screen to the single tree.
- (E,V) An icon in the right bottom corner shows whether there is or there is not connection with the PMS Server available. The refresh rate of this icon has been set in 10 seconds, so possible reconnections or disconnections take a few seconds in being shown.
- (E,V) A button bar contains the different possible actions for the license. This button bar is not available in the integrated version of the editor, but alternative access to the functionality is also offered.

- (E). A context menu can be displayed in order to show edit options.
- (E). Each of the leaves can be edited. (only valid values may be accepted).
- (E). Elements can be deleted. (with the restriction to those elements who cannot dissapear)
- (E). New elements can be added.
- (E). The license can be stored in a xml file and in the server.
- (E). The information in the right hand side panel can be changed. Upon the presh of a button, new introduced information is transferred into the current license.
- (E). Creation of PAR from internal PAR associated to an AXMEDIS object.
- (E). Creation of new license from PAR associated to an AXMEDIS object.
- (E) From the integrated DRM PAR editor it is possible to launch the Editor to create licenses

E: Available only in the Editor
V: Available only in the Viewer

6.1.4 DRM Editor Business Logic

The business logic under the DRM viewer interface will cover the actions we can do over three kinds of DRM information:

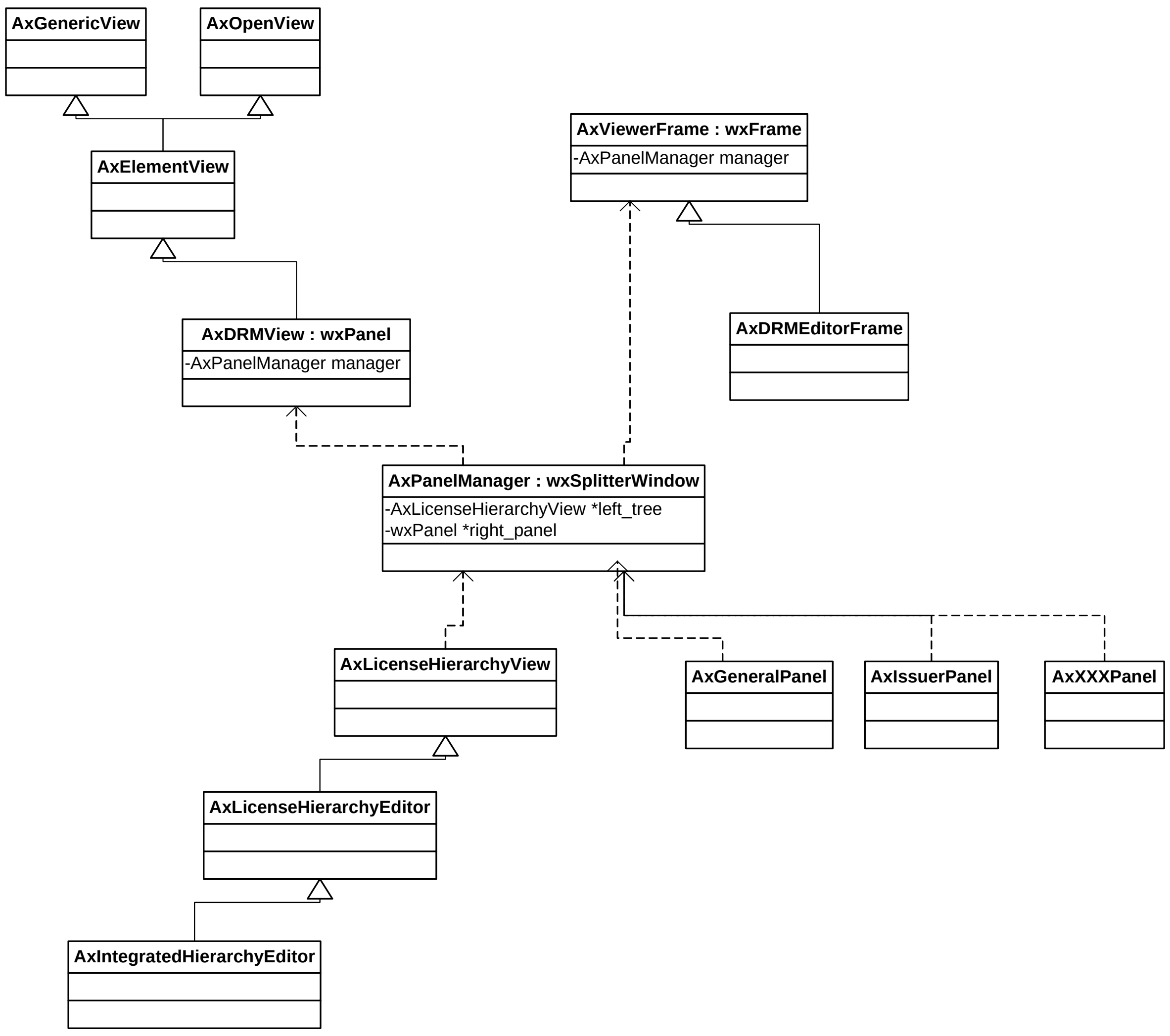
- Licenses
- License template
- Potential Available Rights (both internal and external)

This information will be generated in XML format following the MPEG-21 REL language. For this reason, the editor will construct a tree, with some restrictions, mainly imposed by the structure of an REL license (for instance, we always have to define a right but the rest of elements inside a grant are optional).

6.2 Module Design in terms of Classes

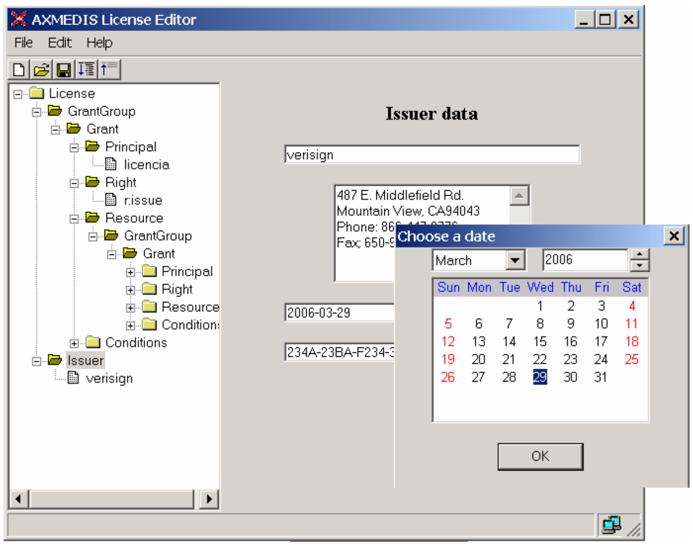
The following class diagram shows the main classes of the DRM Editor and Viewer. Methods inside the classes are not described in detail to facilitate hierarchy view.

For the DRM Editor and Viewer as a module, AxPanelManager inherits from AxDRMView (and higher classes), while for the case of a standalone application, AxViewerFrame is the parent.
Note that this has the advantage of making the standalone version independent of complex Axmedis modules, such as AxInfo, commons etc.



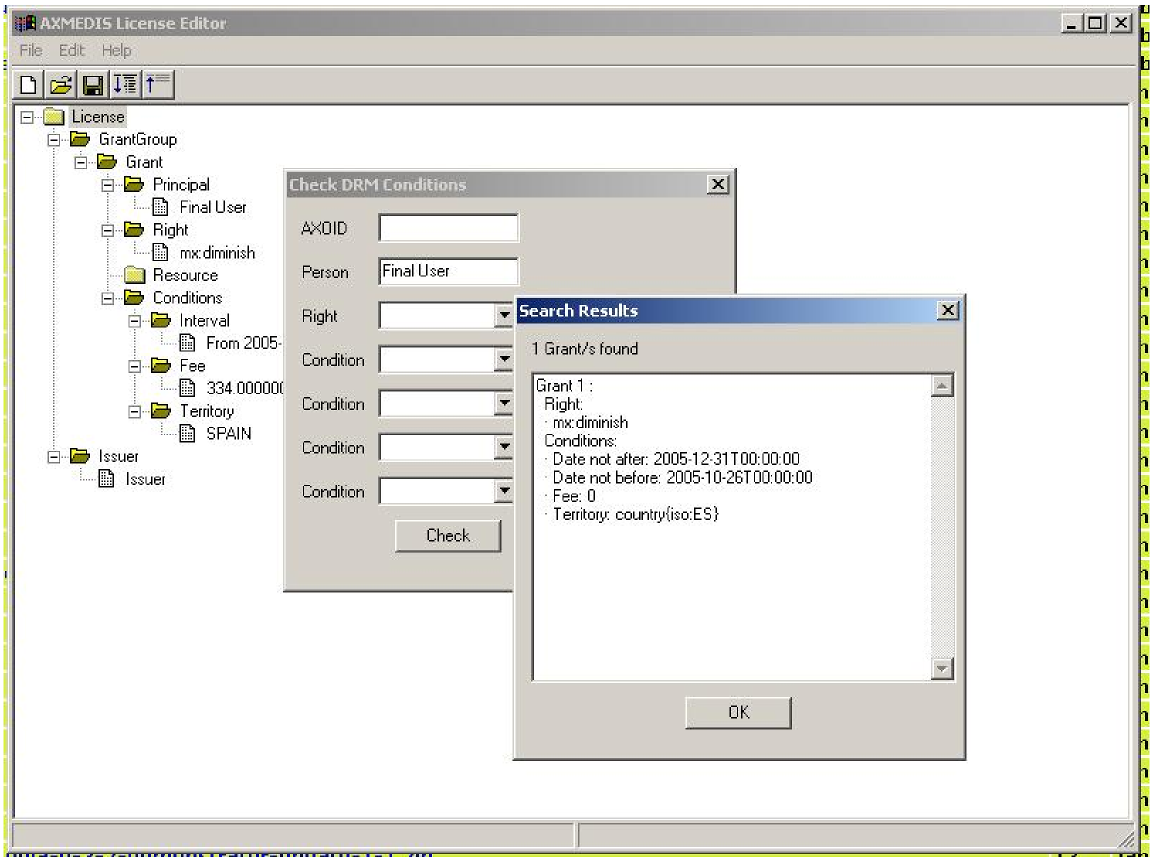
6.3 User interface description

The following figure shows the user interface for the creation of DRM licenses. It follows a tree-like structure, where the elements could be added following the MPEG-21 REL structure.



Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Figure. DRM Editor and Viewer user interface



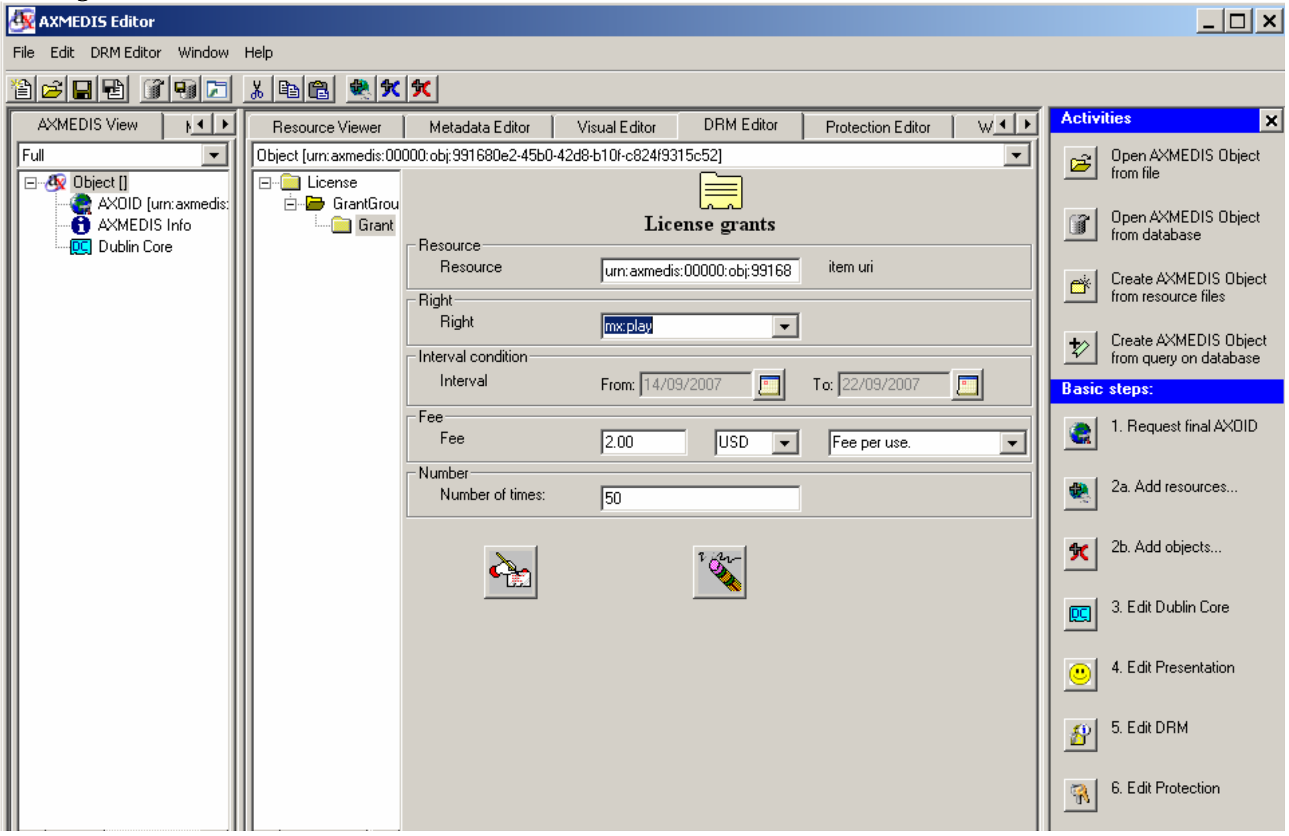
Person: Final User

1 Grant/s found

```
Grant 1 :
  Right:
   · mx:diminish
  Conditions:
   · Date not after: 2005-12-31T00:00:00
   · Date not before: 2005-10-26T00:00:00
   · Fee: 0
   · Territory: country(iso:ES)
```

Figure. Check if license accomplishes conditions

This tree view can also be achieved as the program is ready to switch to this simpler interface. However, the most typical view will be the one present as a component of other programs, i.e., the Axmedis Editor. The next figure shows such a view.

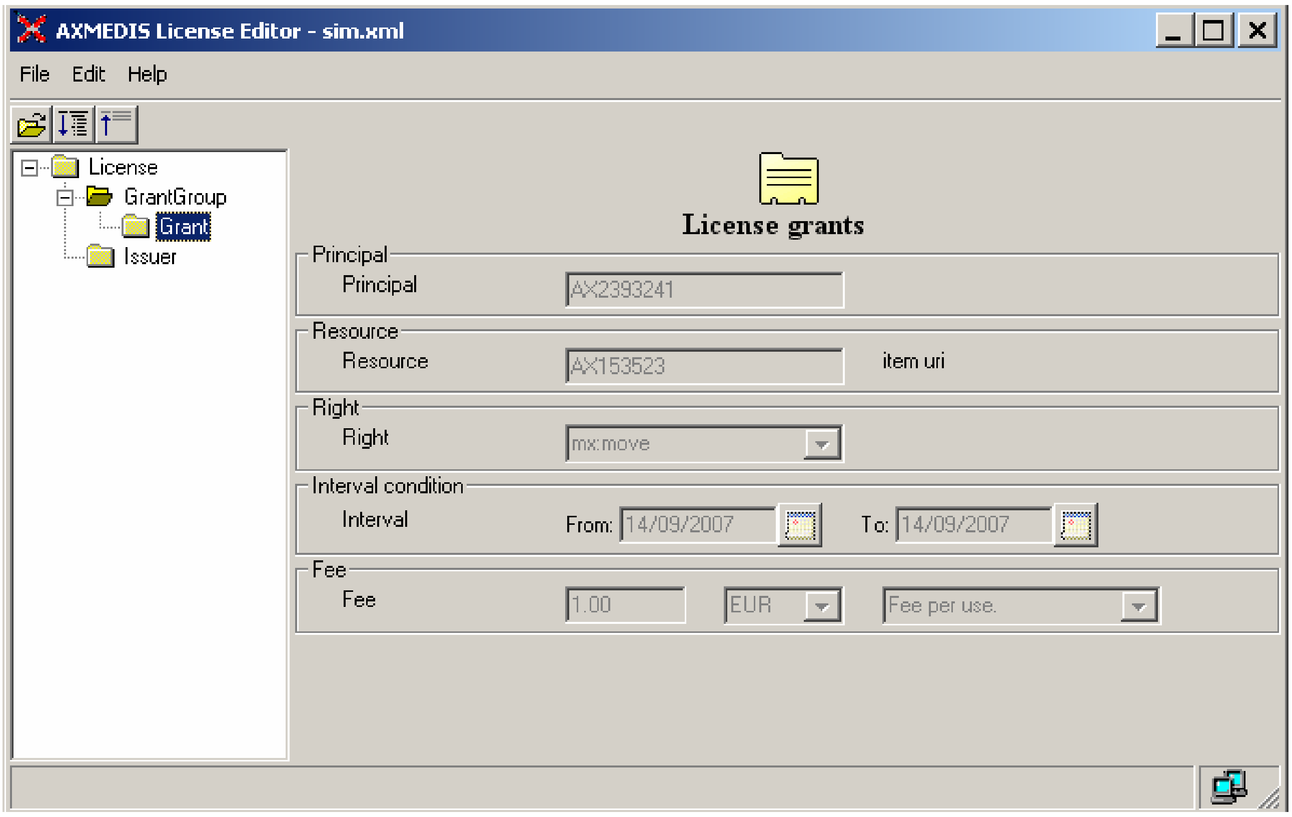


Object [urn:axmedis:00000:obj:991680e2-45b0-42d8-b10f-c824f9315c52]

Resource	Resource	urn:axmedis:00000:obj:99168	item uri	
Right	Right	mx:play		
Interval condition	Interval	From: 14/09/2007	To: 22/09/2007	
Fee	Fee	2.00	USD	Fee per use.
Number	Number of times:	50		

6.3.1 DRM Viewer

The DRM viewer only shows the structure of DRM information in the shape of a tree. The editing functionalities will be disabled, and only the possibility to ask for available actions based on DRM information will be permitted. The next figure shows how the fields are protected.



Principal — Principal: AX2393241
Resource — Resource: AX153523 item uri
Right — Right: mx:move
Interval condition — Interval: From: 14/09/2007 To: 14/09/2007
Fee — Fee: 1.00 EUR Fee per use.

6.4 Technical and Installation information

References to other major components needed	License Model PMS Client PMS Server
Problems not solved	
Configuration and execution context	It is provided with an installable version for Windows OS

7 Module - Protection Editor and Viewer (FHGIGD)

Module/Tool Profile	
Protection Editor and Viewer	
Responsible Name	Alexander Opel
Responsible Partner	FHGIGD
Status (proposed/approved)	Approved
Implemented/not implemented	Implemented
Status of the implementation	First prototype
Executable or Library/module	Library

(Support)		
Single Thread or Multithread	Single thread	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/include/protection_editor_viewer/ https://cvs.axmedis.org/newrepos/Framework/project/protection_editor_viewer/ https://cvs.axmedis.org/newrepos/Framework/source/protection_editor_viewer/	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/axeditor/bin	
Reference to the AXFW location of the demonstrator executable tool for public download	-	
Address for accessing to WebServices if any, add accession information (user and Passwd) if any	-	
Test cases (present/absent)	Absent	
Test cases location	-	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements		
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
yes	Protection Processor	
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section

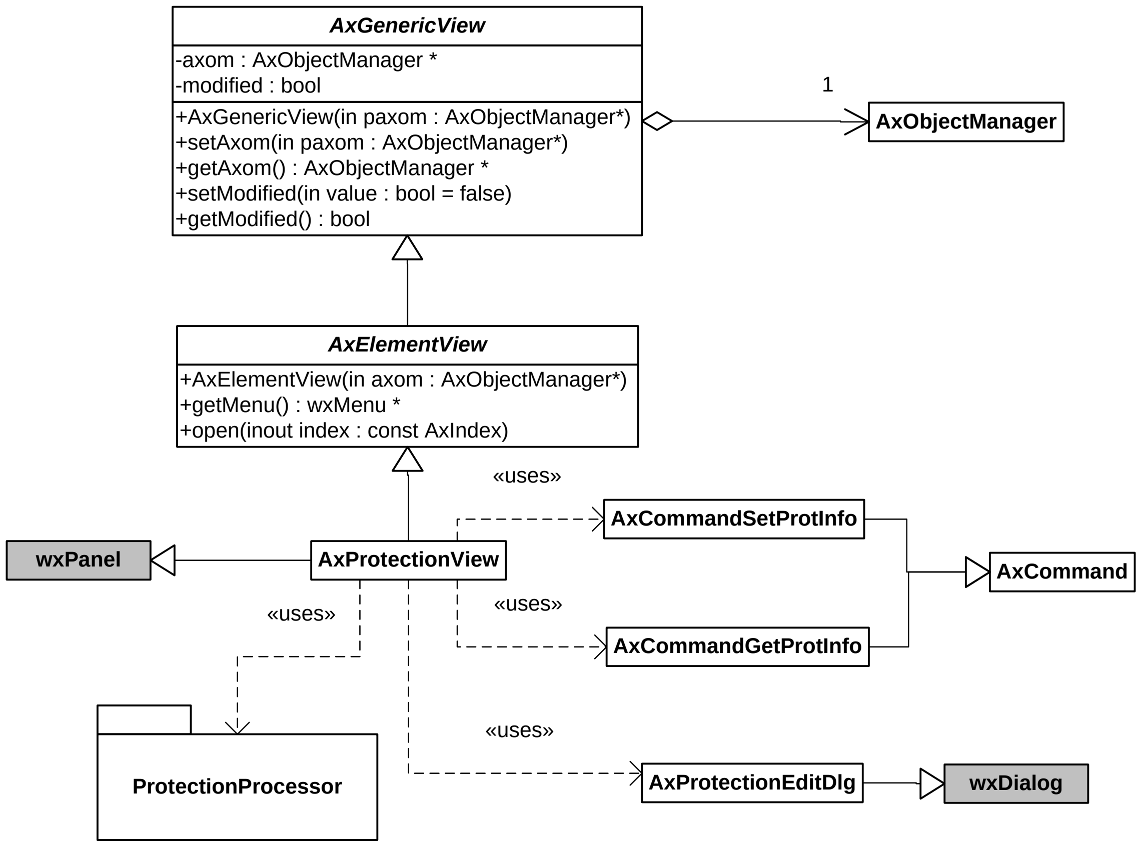
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
yes	C++	
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
yes	WxWidgets	wxWindows licence (http://www.wxwidgets.org/newlicen.htm)

7.1 General Description of the Module

The Protection Information Editor and Viewer provides the functionalities to view and edit protection information:

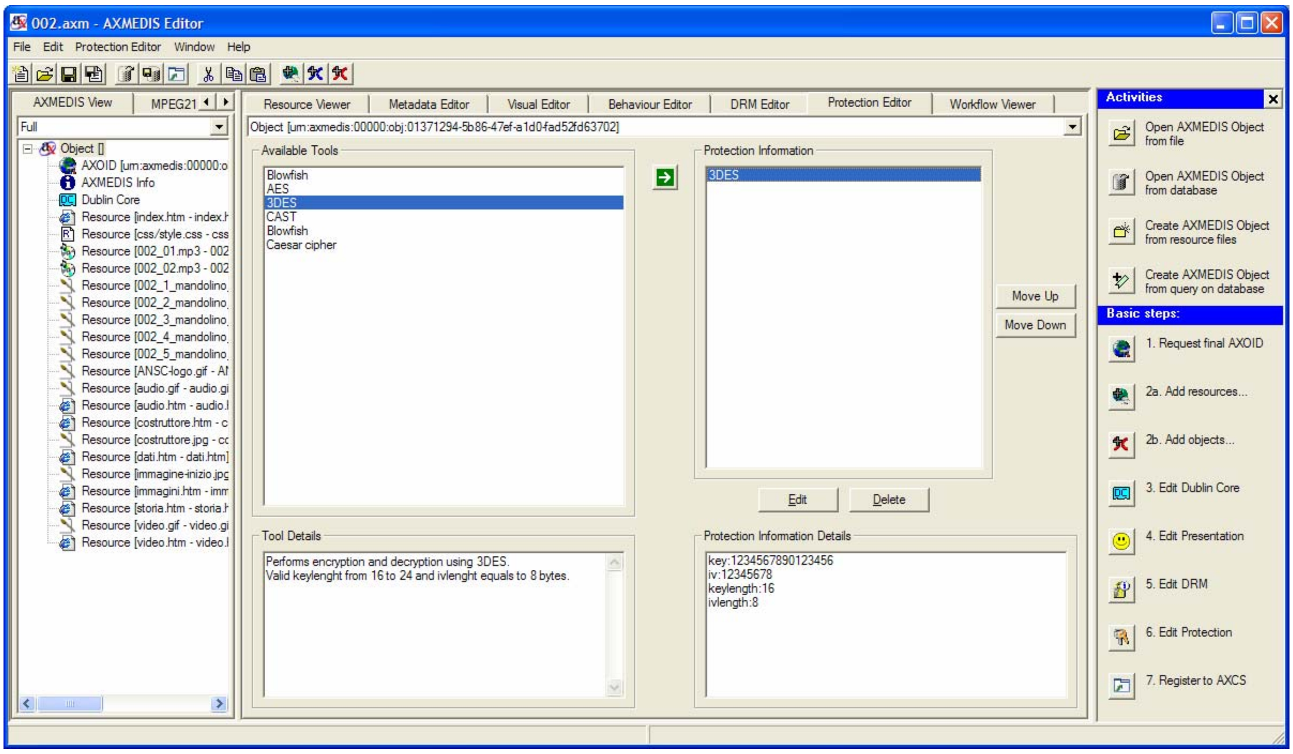
- The user can browse the protection information, the list of protection operations that were applied to the selected part of an AXMEDIS object.
- The user can view detailed information about a specific protection operation including all parameters and the protection target.
- The user can alter the order of different protection operations.
- The user can delete one of the protection operations from the list of protection steps.
- The user can select one of the available tools for protection, e.g. encryption, scrambling or compression, and add an additional protection operation to a specific part of an AXMEDIS object.

7.2 Module Design in terms of Classes


AxGenericView
-axom : AxObjectManager *
-modified : bool
+AxGenericView(in paxom : AxObjectManager*)
+setAxom(in paxom : AxObjectManager*)
+getAxom() : AxObjectManager *
+setModified(in value : bool = false)
+getModified() : bool
1
AxObjectManager
AxElementView
+AxElementView(in axom : AxObjectManager*)
+getMenu() : wxMenu *
+open(inout index : const AxIndex)
«uses»
AxCommandSetProtInfo
wxPanel
AxProtectionView
AxCommand
«uses»
«uses»
AxCommandGetProtInfo
ProtectionProcessor
«uses»
AxProtectionEditDlg
wxDialog

7.3 User interface description

The following figure shows the current user interface for the protection of resources and the vieweing of protection information. It follows a list structure where the elements represent the different protection steps.



Object []
AXOID [urn:axmedis:00000:o
AXMEDIS Info
Dublin Core
Resource [index.htm - index.h
Resource [css/style.css - css
Resource [002_01.mp3 - 002
Resource [002_02.mp3 - 002
Resource [002_1_mandolino.
Resource [002_2_mandolino.
Resource [002_3_mandolino.
Resource [002_4_mandolino.
Resource [002_5_mandolino.
Resource [ANSC-logo.gif - AN
Resource [audio.gif - audio.gi
Resource [audio.htm - audio.l
Resource [costruttore.htm - c
Resource [costruttore.jpg - cc
Resource [dati.htm - dati.htm]
Resource [immagine-inizio.jpg
Resource [immagini.htm - imm
Resource [storia.htm - storia.h
Resource [video.gif - video.gi
Resource [video.htm - video.l

Object [urn:axmedis:00000:obj:01371294-5b86-47ef-a1d0-fad52fd63702]

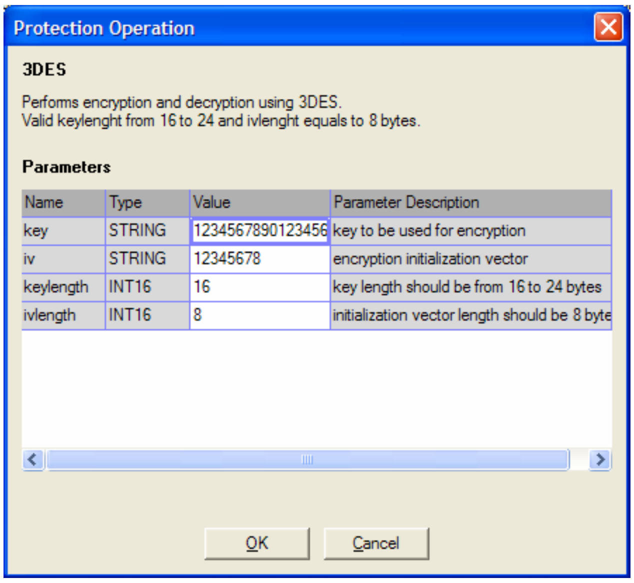
Blowfish
AES
3DES
CAST
Blowfish
Caesar cipher

3DES

Performs encryption and decryption using 3DES.
Valid keylenght from 16 to 24 and ivlenght equals to 8 bytes.

key:1234567890123456
iv:12345678
keylength:16
ivlength:8

Figure. Protection Information Editor and Viewer user interface



3DES

Performs encryption and decryption using 3DES.
Valid keylenght from 16 to 24 and ivlenght equals to 8 bytes.

Parameters

Name	Type	Value	Parameter Description
key	STRING	1234567890123456	key to be used for encryption
iv	STRING	12345678	encryption initialization vector
keylength	INT16	16	key length should be from 16 to 24 bytes
ivlength	INT16	8	initialization vector length should be 8 byte

Figure: Parameter setting for a specific Protection Operation

7.4 Technical and Installation information

References to other major components needed	Protection Processor
Problems not solved	
Configuration and execution context	The AXMEDIS Protection Information Editor and Viewer is executed within the AXMEDIS Editor.

7.5 Draft User Manual and Example of Usage

See section 7.3 User Interface Description above and for more details, see AXMEDIS User Manual DE5-0-1-2 for AXTools.

7.6 Integration and compilation issues

None

7.7 Configuration Parameters

Config parameter	Possible values

7.8 Errors reported and that may occur

Error code	Description and rationales

8 Module Visual Editor and Viewer (EPFL)

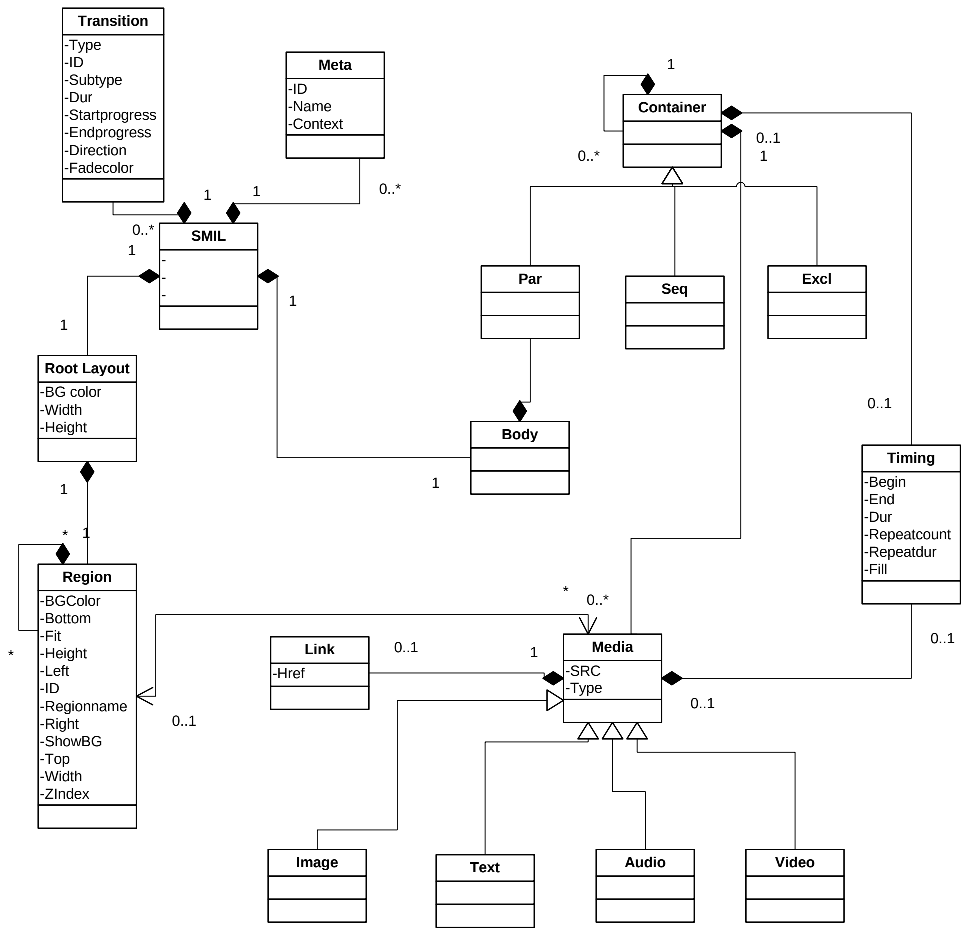
Module/Tool Profile		
Visual Editor and Viewer		
Responsible Name	Beilu Shao, Samuel Keller	
Responsible Partner	EPFL	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	100%	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows and Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/visual_editor_viewer https://cvs.axmedis.org/newrepos/Framework/source/smil_structure https://cvs.axmedis.org/newrepos/Framework/source/smil_view https://cvs.axmedis.org/newrepos/Framework/source/smil_parser https://cvs.axmedis.org/newrepos/Framework/source/smil_visual https://cvs.axmedis.org/newrepos/Framework/source/smil_dialog https://cvs.axmedis.org/newrepos/Framework/source/smil_ogl	
Reference to the AXFW location of the demonstrator executable tool for internal download		
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any	N/A	
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	Yes	
Usage of the AXMEDIS Error Manager (yes/no)		
Major Problems not solved	None	
Major pending requirements	None	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)

Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
	C++	wxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets	wxWidgets 2.4.2	LGPL

8.1 General Description of the Module

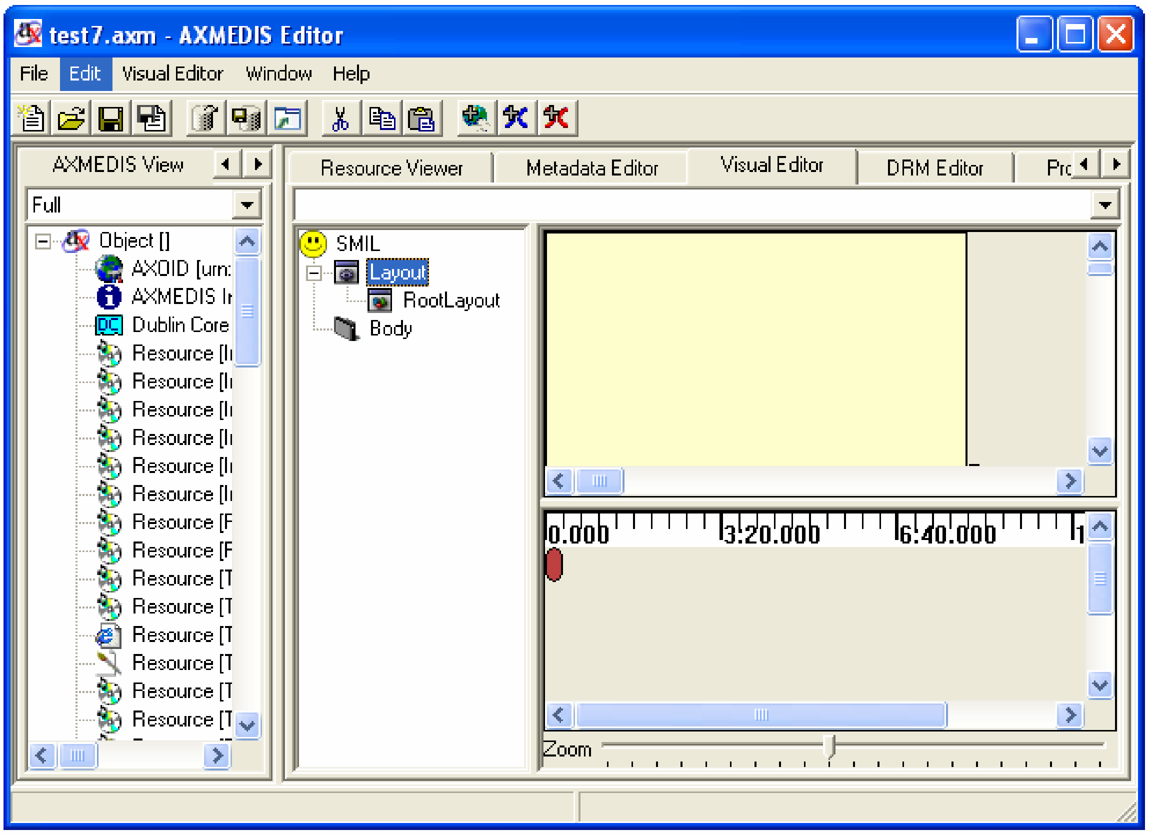
The Visual Editor and Viewer is responsible for defining how to present the media resources that are present inside an AXMEDIS object. For the presentation definition it is used the SMIL language. For this reason

8.1.1 Class diagram of module design for SMIL structure

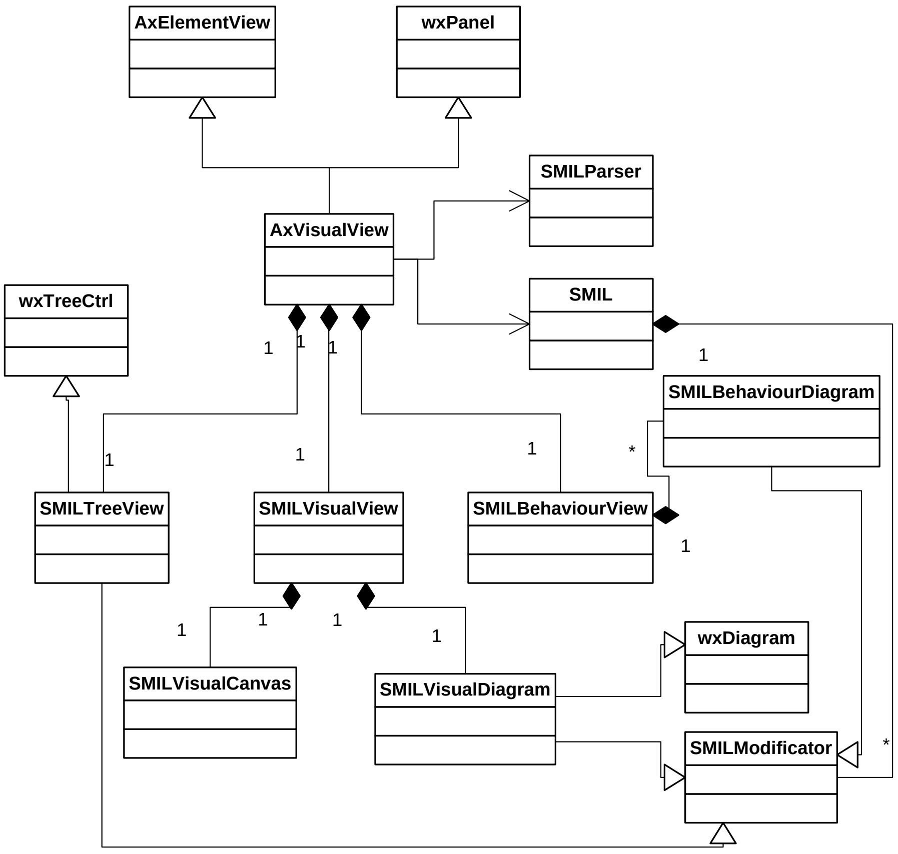


8.1.2 Module design for SMIL editor

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.



8.1.3 Class diagram of module design for SMIL editor



8.2 Tree View part

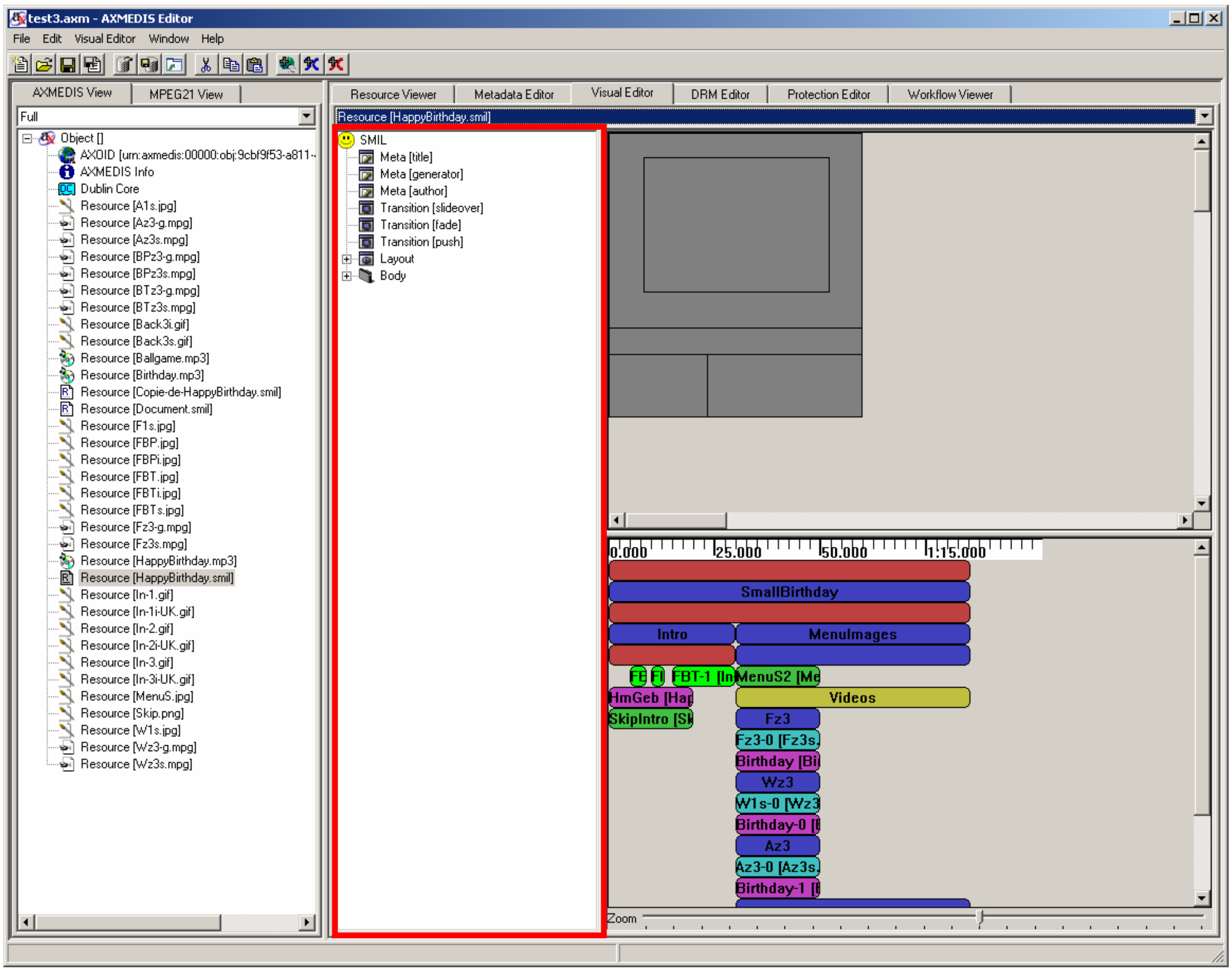
All the SMIL components are stored inside the hierarchy structure between the tags **<body>** and **</body>.** There could be any combinations of tags of <par> and <seq> to support parallel and sequential temporal behaviour of the media objects. The following are some examples of Hierarchy structure 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

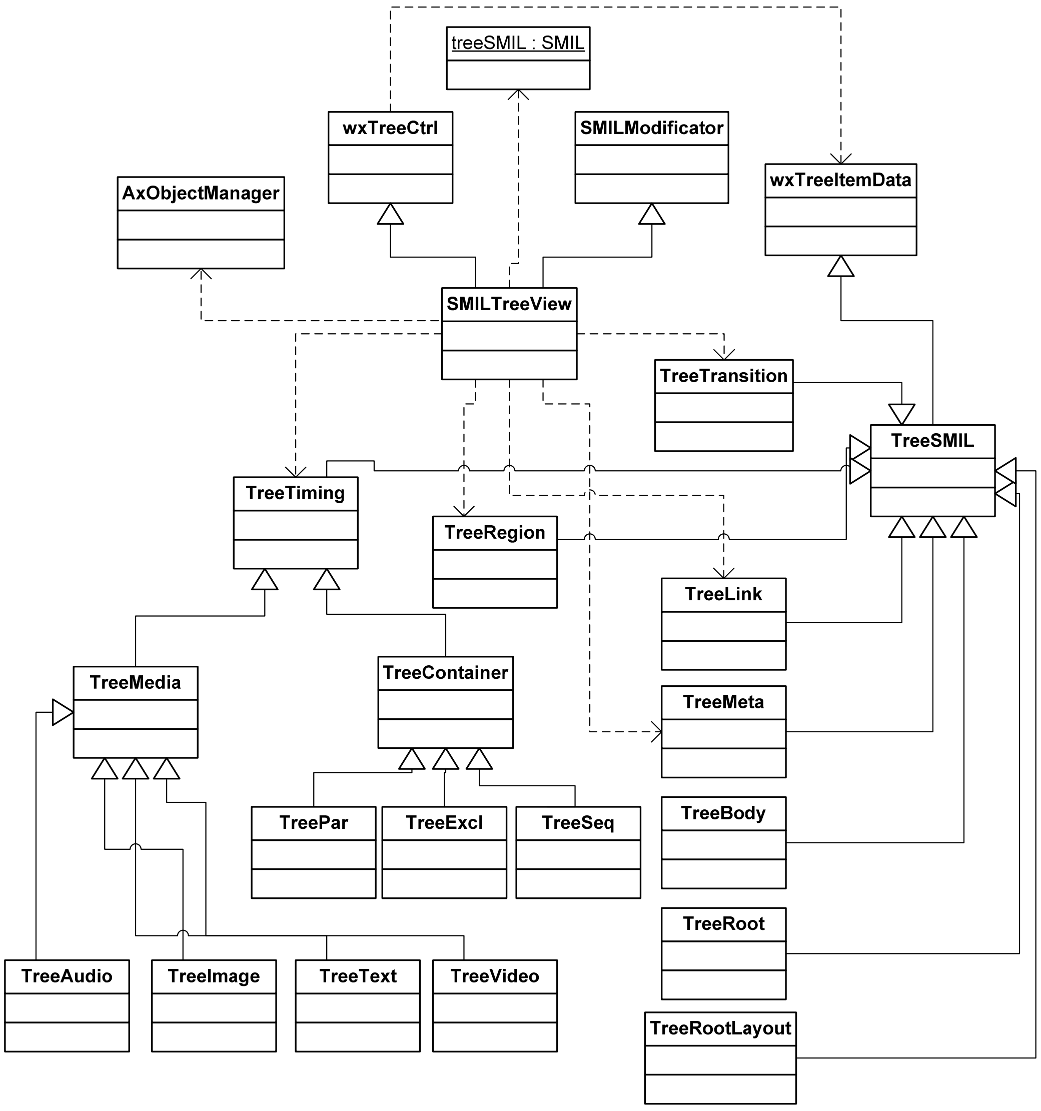
```
<body>
        <par>
                <par dur="30s">
                   <img id="lamp-picture" region="content-image"
             src="LITEdataCE/lamp1.jpg"/>
                   <audio id="Welcome-US" region="LogoRegion"
             src="LITEdataCE/Welcome-US.mp3"  dur="21.4s"/>

                </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>
```

The hierarchy editing of SMIL components is the first step before the spatial visual editing and temporal synchronization. To create an element inside SMIL object, we have to determine what kind of structure of these elements. To be more specific, we have to define it is container (par/seq) or object. The following is a snapshot of the GUI of the hierarchy structure. The tree part is located at the left of the SMIL Editor.



The tree view part of the SMIL editor is used for manipulating the whole SMIL with his internal structure. It is the only view that can display/edit completely Metas, Transitions and Links.

8.2.1 Class diagram of module design for Tree View



8.3 Visual View part

Spatial View shall show object placement in a 2-D (or possibly 3-D) environment. Moreover, Spatial View shall permit managing (i.e. moving, deleting, adding, etc…) object subparts which have spatial properties or constraints.

- Spatial View shall be able to proportionally represent components in all significant spatial directions;
- Spatial View shall permit to modify spatial properties and constraints by means of graphical actions such as drag-and-drop, contextual menu, etc…
- Component spatial properties should be relative to the comprising container or to other items. Otherwise, those properties should be absolute respects to the entire object;
- Position constraints should be represented as labeled solid line where the label contains the distance measure between the components and the constraint reference, e.g. if the position of a component is absolute, a line for each direction will show the distance from the reference visualization edge.

The Visual Editor will arrange the media objects (video, photo, text, etc) on the screen. The Visual Business Logic will store and manage an XML document that will describe the placement and dimensions of the media objects on the screen i.e. the layout. This XML document will be integrated inside the AXMEDIS document enclosed inside a <Component> tag pair. The Business Logic will access the AXMEDIS document through the Command Manager.

```
<?xml version="1.0" encoding="UTF-8"?><DIDL
xmlns:xi="http://www.w3.org/2001/XInclude" >
...
...
<Item>
                  <Component id="Colline-azzurre">
                        <Resource mimeType="image/jpeg"
encoding="base64">...</Resource>
                  </Component>
                  <Component id="smilcomponent">
                        <Resource mimeType="application/smil"
encoding="base64">...</Resource>
                  </Component>
            </Item>
      </Item>
</DIDL>
```

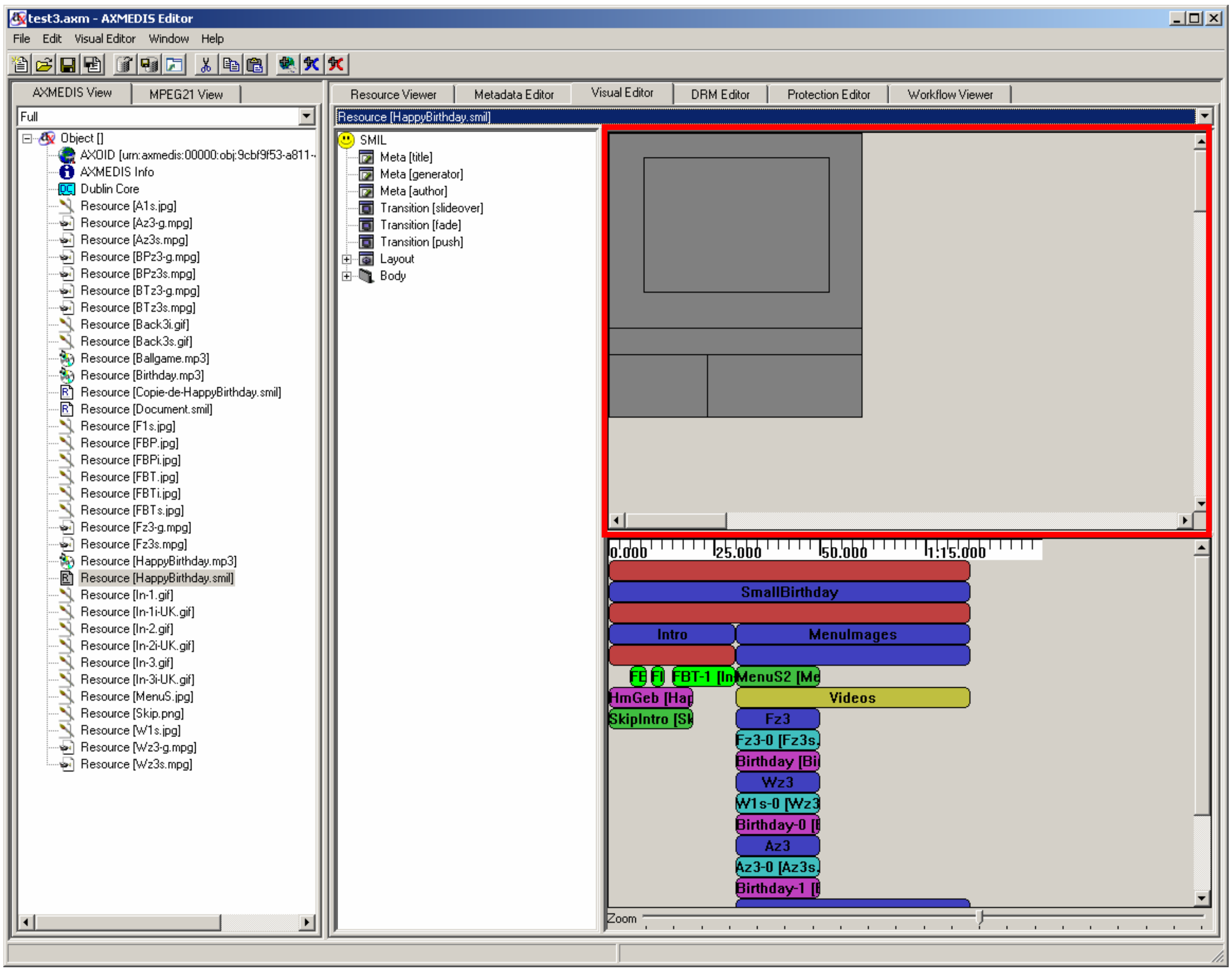
As the user interacts with the GUI, the Business Logic will update the underlying data representation of the layout by editing the XML document. The layout description will be done according to the SMIL W3C standard. SMIL stands for Synchronized Multimedia Integration Language; it is an HTML-like language for describing audiovisual presentations in XML. A part of the SMIL language is devoted to describing the size and location on the screen of the media objects. For this reason, the data architecture of the Visual Business Logic will be that of SMIL. The Visual Business Logic will implement a subset of the SMIL layout features. More SMIL features can be added as the project evolves if we deem it necessary.
Below I will explain the list of SMIL XML elements and attributes that will allow the Visual Business Logic to organize and store the layout of the presentation. The following is the basic skeleton of the SMIL presentation that we will have to support:

```
<?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>
```

The "smil" element is needed to be SMIL compatible. The "smil" element is the root element of any presentation. It only has an identifier attribute. The "head" element is a child of the "smil" element and it will be needed to be SMIL compatible. Like the "smile" element, it only has an identifier attribute. Inside the "head" element, there is the "layout" element that, as the name states, contains the layout information. The "layout" element has two attributes: an identifier attribute and a "type" attribute. The "type" attribute specifies which layout language is used in the layout element. In our implementation, we intend to support only one language for the layout description, the SMIL Basic Layout Language. Therefore, we will set the "type" attribute to the fixed value "text/smil-basic-layout" or we will omit the attribute because the default value is the one that we want. The "body" element is explained in the Behaviour Editor section since this element deals with time synchronization issues. Finally, we have the two most important elements to describe the spatial placing of the media objects; these are the "root-layout" and the "region" elements. Both are children of the "layout" element. Their purpose is to define rectangular regions on the screen. Each rectangular region serves as placeholder for one or more media objects. For instance, imagine defining a single rectangle where three videos will be rendered one after the other. Every rectangular region has to have an identifier. This identifier is very important because it will be used in the Behaviour Editor to associate the rectangle with the media objects. See Behaviour editor for more details. The “root-layout” element is a little bit special because it defines the rectangular region where the whole presentation will be displayed. See two examples of usage:

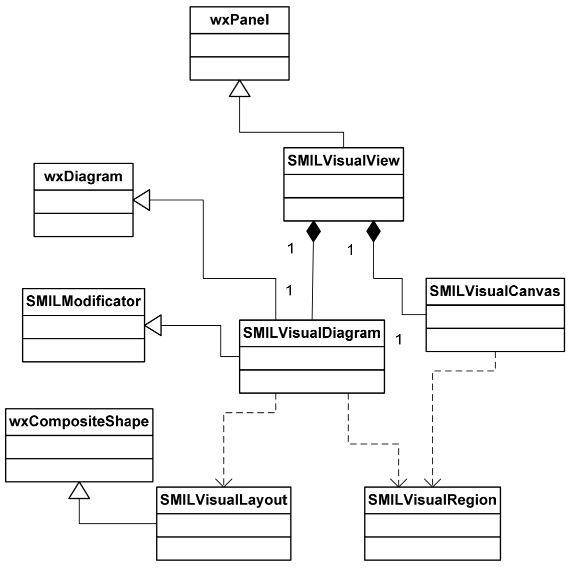
- <root-layout>: to set the width and height for the window in which the presentation will be rendered. E.g.: **<root-layout width="300" height="200" background-color="white"/>**
- <region>: to define a rectangular region of the display area where a media object will be placed. E.g.: **<region id = ”some_id” left = ”0” top = ”0” width = ”32” height = “32” />**

The visual part is located at the right top of the SMIL Editor.



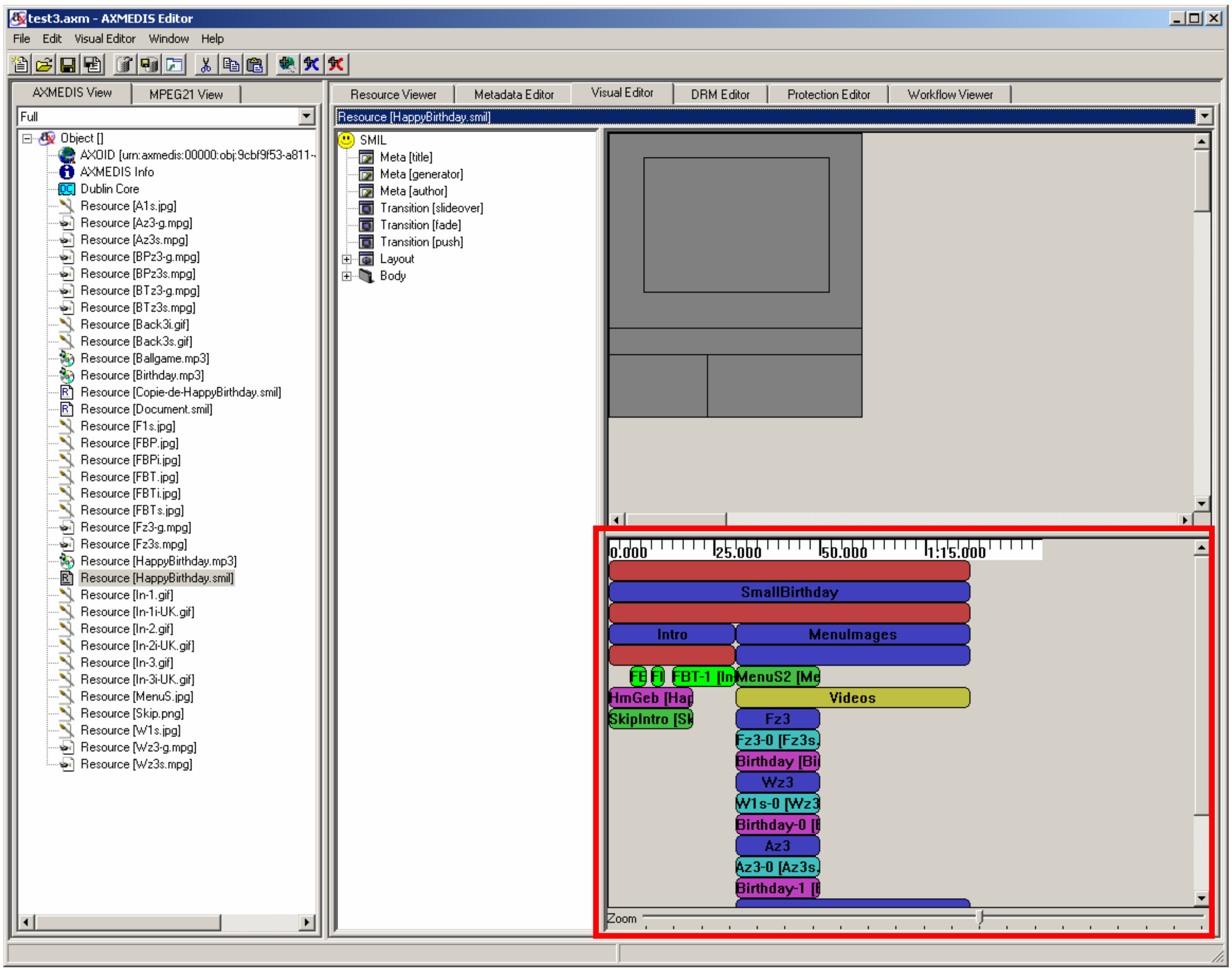
The visual view part of the SMIL editor is used for manipulating the SMIL layout which including the Rootlayout, Regions, Sub-regions of regions.

8.3.1 Module Design in terms of Classes for Visual View part



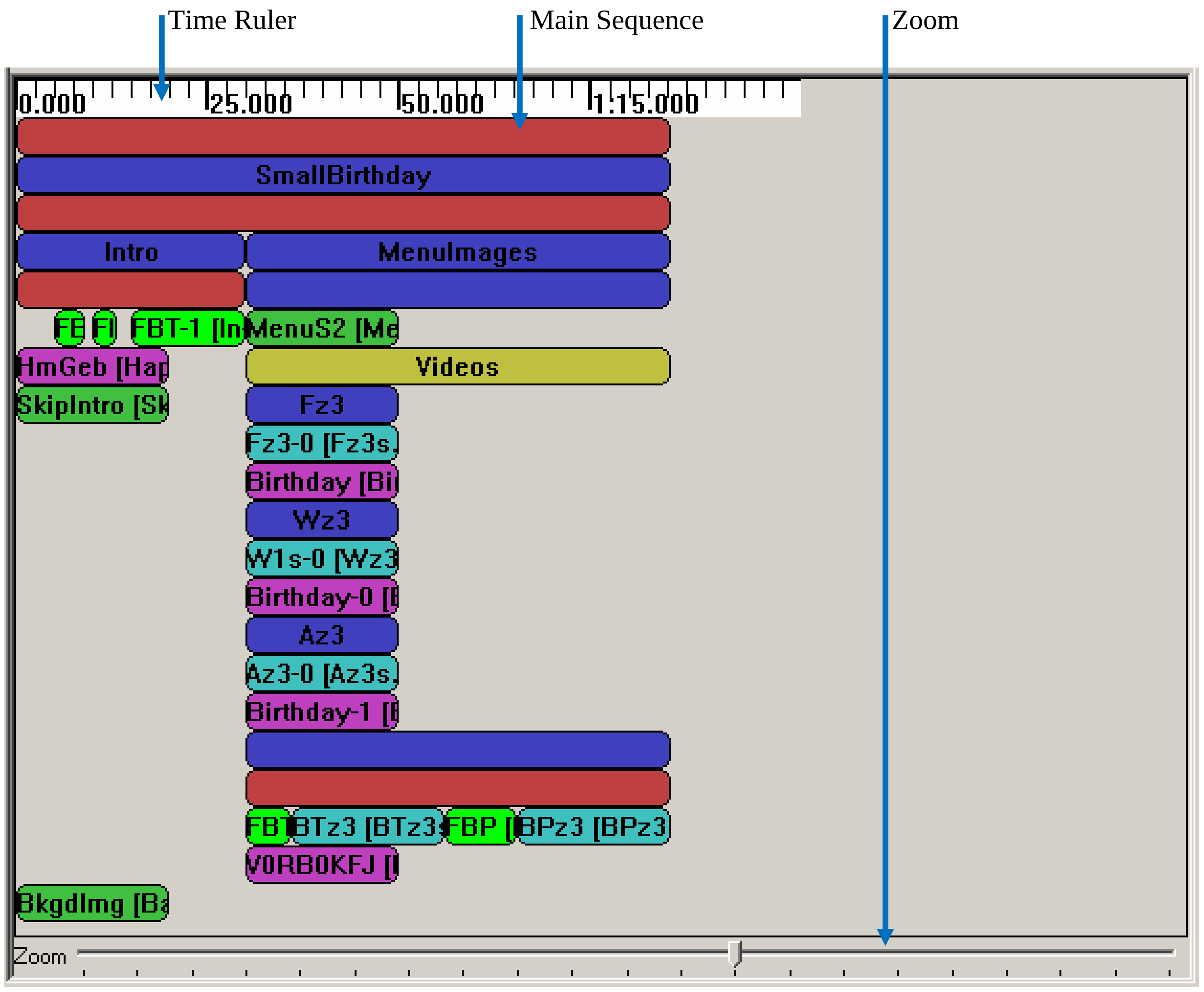
8.4 Behaviour view part

The behaviour part is located at the right bottom of the SMIL Editor.

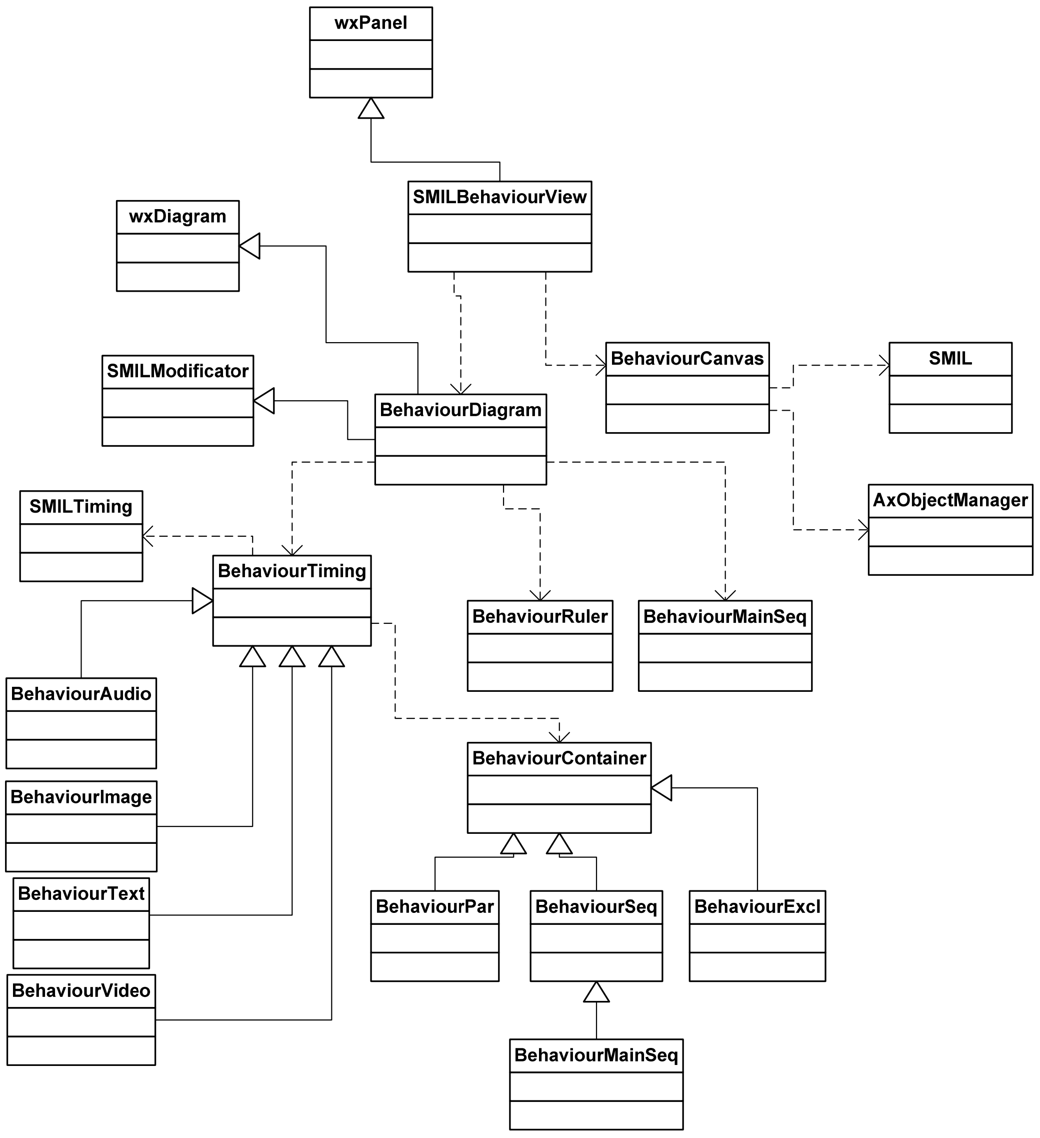


8.4.1 Description of components

At the top of the behaviour view you can see the time ruler. This ruler gives timing information and a scaling view for the whole SMIL timing.
At the centre you can see and modify the timing attributes of the SMIL. At the top there is always an not modifiable anonym red bar that represent the main sequence who is the basic timing structure of the SMIL file. Under it, that means "in it", 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). You can also add/modify/delete the medias that will be played.
If the timings are too big navigation scroll bars appears.
At the bottom, you can see and modify the zoom scale used for timing display.



8.4.2 Class diagram of module design for Behaviour View part



8.5 Technical and Installation information

References to other major components needed	AXMEDIS Object Manager
Problems not solved	None
Configuration and execution context	

8.6 Draft User Manual

See AXMEDIS User Manual DE5-0-1-2 for AXTools.

8.7 Examples of usage

Please See the User manual

8.8 Integration and compilation issues

None

8.9 Configuration Parameters

Config parameter	Possible values

8.10 Errors reported and that may occur

Error code	Description and rationales

9 Module Behaviour and Functional Editor and Viewer (DSI)

Module/Tool Profile	
Behaviour and Functional Editor and Viewer	
Responsible Name	Pierfrancesco Bellini
Responsible Partner	DSI
Status (proposed/approved)	Approved
Implemented/not implemented	Implemented
Status of the implementation	80%
Executable or Library/module (Support)	Library
Single Thread or Multithread	Multithread
Language of Development	C++
Platforms supported	Windows, Linux
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/behaviour_editor_viewer
Reference to the AXFW location of the demonstrator executable tool for internal download	
Reference to the AXFW location of the demonstrator executable tool for public download	
Address for accessing to WebServices if any, add	

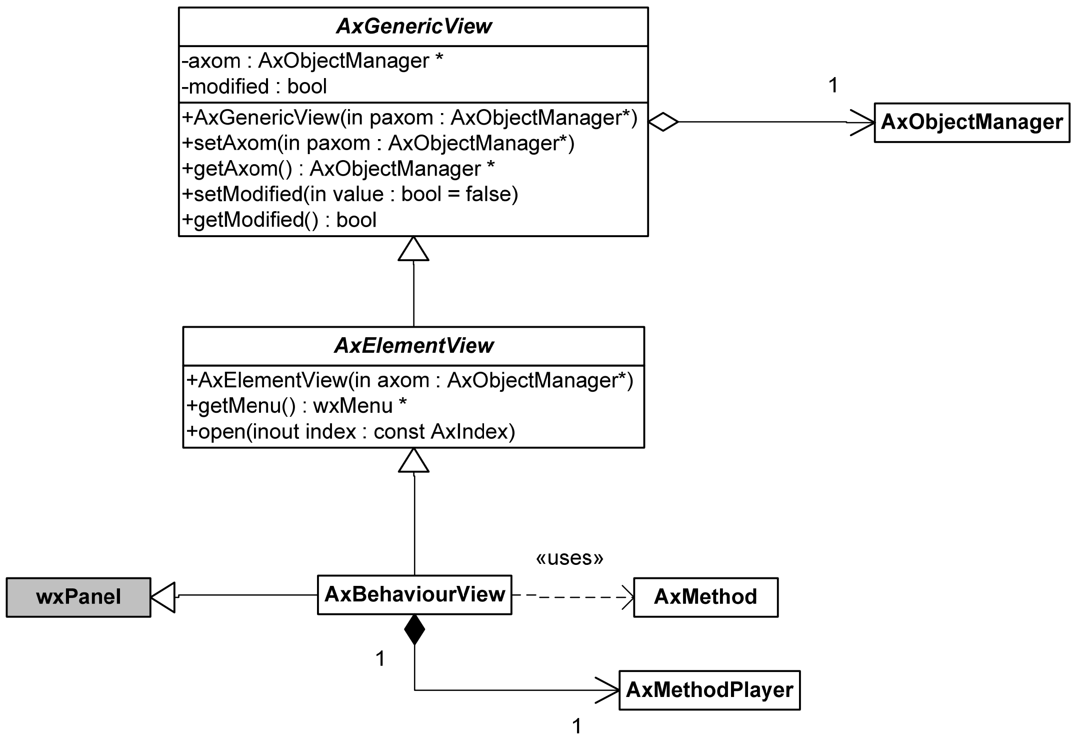
accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	Yes	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
	C++	wxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets	wxWidgets 2.4.2	LGPL

9.1 General Description of the Module

The Behaviour Editor it is used to create MPEG 21 DIP methods that will be stored in the AXMEDIS Objects.
Theses methods are written in Javascript allowing to define some specific behaviour that will be used from the end user (see DE 3.1.2.2.3)
The Method editor allows to:

- define the method text
- define if the method is auto run or not
- define the arguments
- create a new method
- save the method in the AXMEDIS object
- play the method in the Method Player

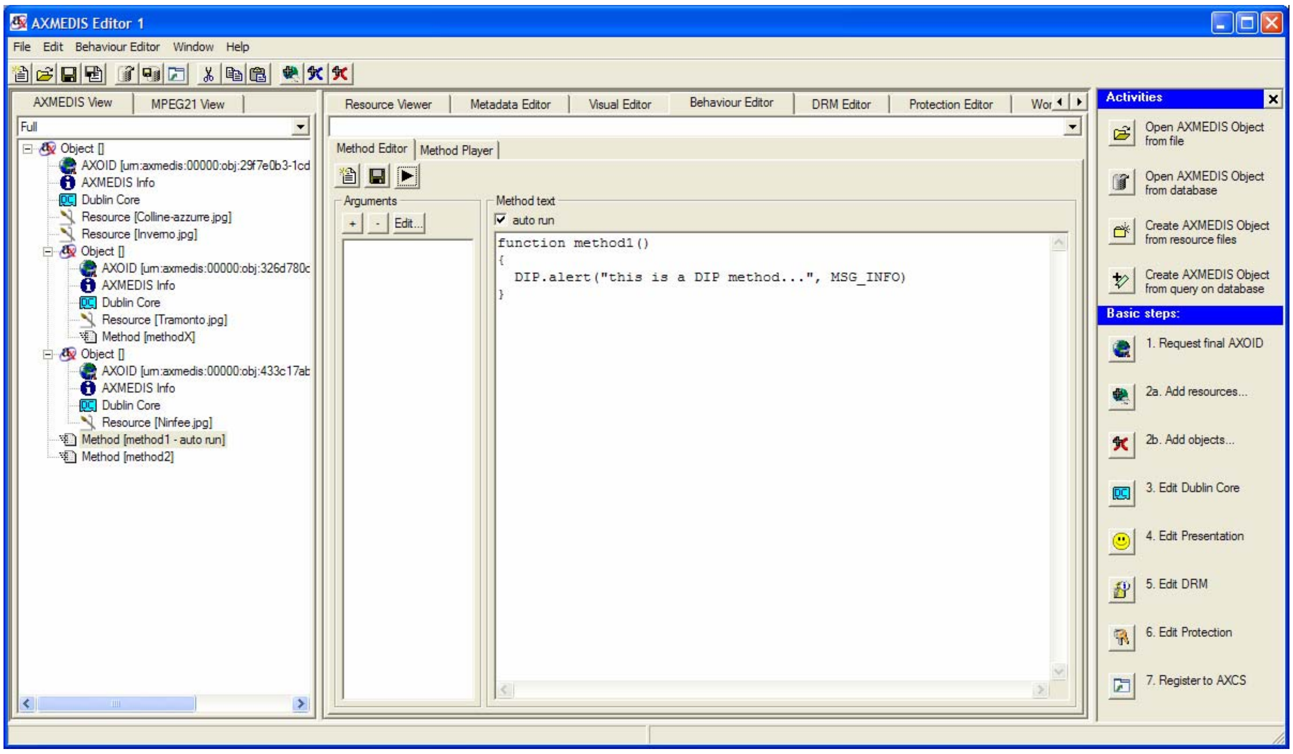
9.2 Module Design in terms of Classes


AxGenericView
-axom : AxObjectManager *
-modified : bool
+AxGenericView(in paxom : AxObjectManager*)
+setAxom(in paxom : AxObjectManager*)
+getAxom() : AxObjectManager *
+setModified(in value : bool = false)
+getModified() : bool
1
AxObjectManager
AxElementView
+AxElementView(in axom : AxObjectManager*)
+getMenu() : wxMenu *
+open(inout index : const AxIndex)
«uses»
wxPanel
AxBehaviourView
AxMethod
1
AxMethodPlayer
1

The *AxBehaviourView* class is responsible for the user interface, it allows to write the method and store it in the *AxObjectManager* object, it uses the *AxMethodPlayer* class to execute the method.

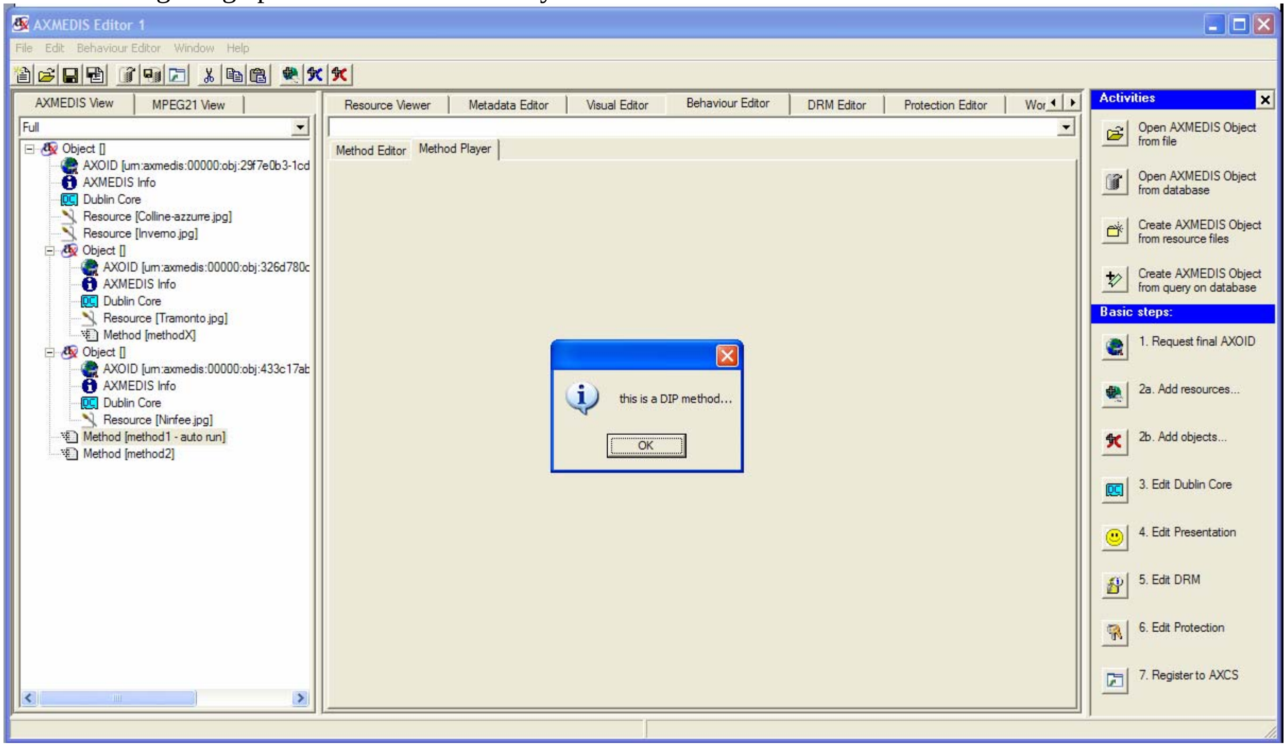
9.3 User interface description

The following image presents the Method Editor user interface


AXMEDIS Editor 1
File Edit Behaviour Editor Window Help
AXMEDIS View MPEG21 View
Full
Object []
AXOID [urn:axmedis:00000:obj:29f7e0b3-1cd
AXMEDIS Info
Dublin Core
Resource [Colline-azzurre.jpg]
Resource [Inverno.jpg]
Object []
AXOID [urn:axmedis:00000:obj:326d780c
AXMEDIS Info
Dublin Core
Resource [Tramonto.jpg]
Method [methodX]
Object []
AXOID [urn:axmedis:00000:obj:433c17ab
AXMEDIS Info
Dublin Core
Resource [Ninfee.jpg]
Method [method1 - auto run]
Method [method2]
Resource Viewer Metadata Editor Visual Editor Behaviour Editor DRM Editor Protection Editor Wor
Method Editor Method Player
Arguments
+ - Edit...
Method text
auto run
Activities
Open AXMEDIS Object from file
Open AXMEDIS Object from database
Create AXMEDIS Object from resource files
Create AXMEDIS Object from query on database
Basic steps:
1. Request final AXOID
2a. Add resources...
2b. Add objects...
3. Edit Dublin Core
4. Edit Presentation
5. Edit DRM
6. Edit Protection
7. Register to AXCS

```
function method1()
{
  DIP.alert("this is a DIP method...", MSG_INFO)
}
```

The following image presents the Method Player used to execute the method:


AXMEDIS Editor 1
File Edit Behaviour Editor Window Help
AXMEDIS View MPEG21 View
Full
Object []
AXOID [urn:axmedis:00000:obj:29f7e0b3-1cd
AXMEDIS Info
Dublin Core
Resource [Colline-azzurre.jpg]
Resource [Inverno.jpg]
Object []
AXOID [urn:axmedis:00000:obj:326d780c
AXMEDIS Info
Dublin Core
Resource [Tramonto.jpg]
Method [methodX]
Object []
AXOID [urn:axmedis:00000:obj:433c17ab
AXMEDIS Info
Dublin Core
Resource [Ninfee.jpg]
Method [method1 - auto run]
Method [method2]
Resource Viewer Metadata Editor Visual Editor Behaviour Editor DRM Editor Protection Editor Wor
Method Editor Method Player
OK
Activities
Open AXMEDIS Object from file
Open AXMEDIS Object from database
Create AXMEDIS Object from resource files
Create AXMEDIS Object from query on database
Basic steps:
1. Request final AXOID
2a. Add resources...
2b. Add objects...
3. Edit Dublin Core
4. Edit Presentation
5. Edit DRM
6. Edit Protection
7. Register to AXCS

this is a DIP method...

9.4 Technical and Installation information

References to other major components needed	AXMEDIS Object Manager
Problems not solved	
Configuration and execution context	

9.5 Draft User Manual

9.5.1 Creating a new method

To create a new method in the object:
1. select the Behaviour View
2. press the new method button in the Method Editor Tool bar or use the Behaviour View/New method… menu item, then it asks for the method name (no spaces in the name)
3. write the method name (e.g. show)
4. a predefined function is created in the method text
5. if necessary some method arguments can be defined:
5.1 to add an argument select the + button in the argument list and it asks for argument type
5.2 write the urn that is associated with the argument type (e.g. "urn:example:music") and press OK
5.3 add manually in the function the name of the parameter that will contain the argument. This parameter will contain the DOM node with the Object type that has been specified as argument type.
6. write the function text
7. save the method in the object using the save button in the Method Editor toolbar or use the Behaviour View/Save method menu item
8. Play the method using the run button

9.6 Examples of usage

See the Draft User Manual

9.7 Integration and compilation issues

None

9.8 Configuration Parameters

Config parameter	Possible values

9.9 Errors reported and that may occur

Error code	Description and rationales

10 Module Metadata Editor and Viewer (UNIVLEEDS)

Module/Tool Profile		
Metadata Editor and Viewer		
Responsible Name	Kia Ng and Royce Neagle	
Responsible Partner	UNIVLEEDS	
Status (proposed/approved)		
Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multi Thread	
Language of Development	C++	
Platforms supported	Windows, Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/metadata_editor_viewer https://cvs.axmedis.org/newrepos/Framework/source/metadataeditor https://cvs.axmedis.org/newrepos/Framework/source/metadatamodel	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/axeditor/bin	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	Yes	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section

Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Yes	C++	wxWidgets, Windows,
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWindows	wxWindows, v-2.5.3 or v-2.4.2	LGPL
Xerces	Xerces-c++ v-2.6.0	Apache Software License, v2.0
Xalan	Xalan-c++ v-1.9	Apache Software License, v1.1
wxMozilla	wxMozilla v-0.5.3	GPL

10.1 General Description of the Module

Metadata Editor and Viewer provide functionality to display and edit information contained within the metadata associated with an AXMEDIS object. Due to the complex nature of AXMEDIS object, there may be one or more metadata sections with different schema, including MPEG21, Dublin Core, and AXInfo.

- Metadata Editor and Viewer shall be able to adapt itself (e.g. by analysing the data-related XML schema) automatically to the metadata structure;
- Metadata Editor and Viewer shall be fully configurable, i.e. user shall be able to select, for each set of metadata and for each kind of components, which metadata have to be displayed;
- Specific set of valuable metadata, such as authoring MPEG-7 metadata, should be included into AXMEDIS Editor basic release;

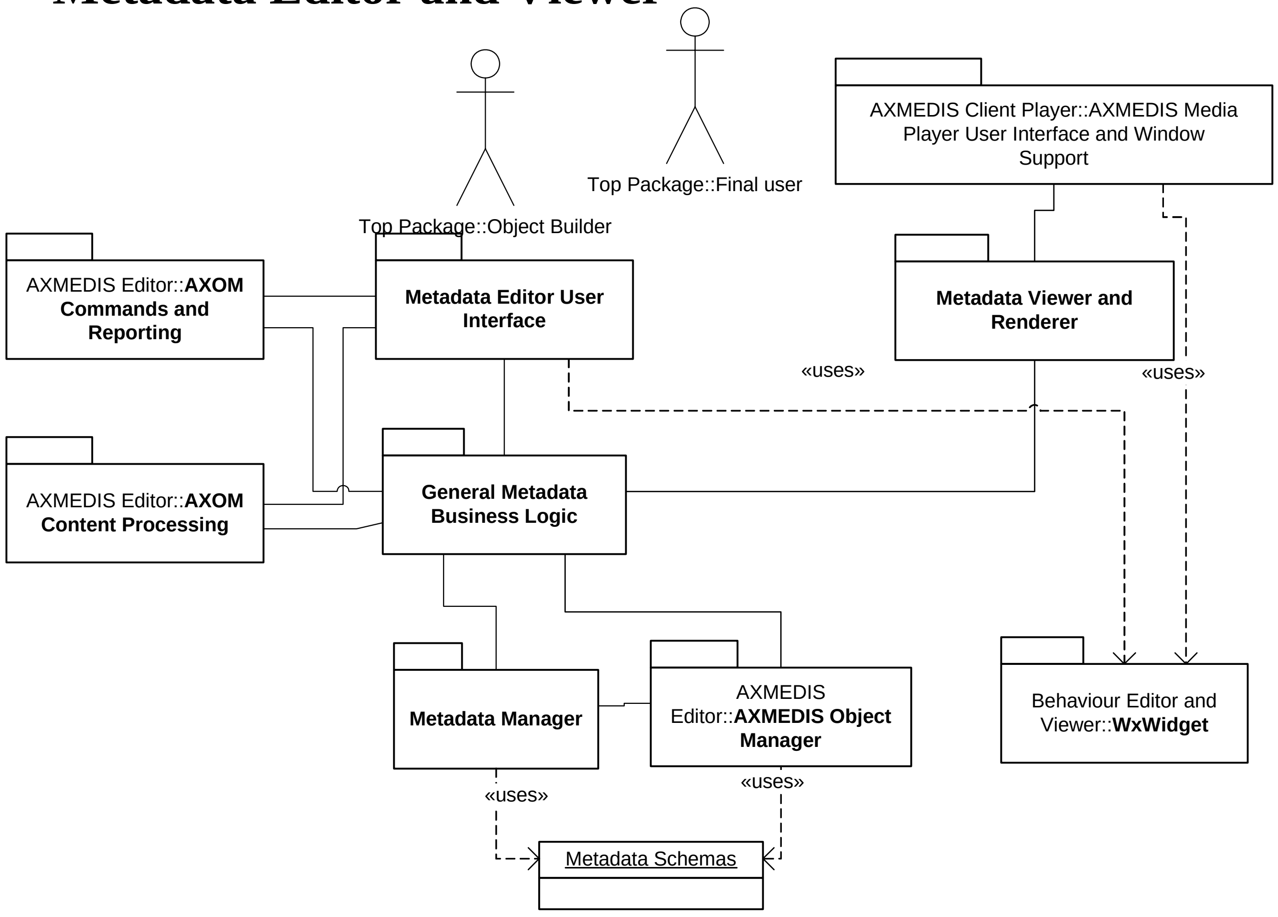
Scenario for editing metadata

1. Open an AxObject from the local file system or a remote database
2. Extract the metadata located in the AxObject and parse it to obtain a list of metadata tag and value
3. Generate visualisation for metadata
4. Provide functionality for the actor to modify the metadata
5. Provide functionality to embed back the revised metadata to the AxObject

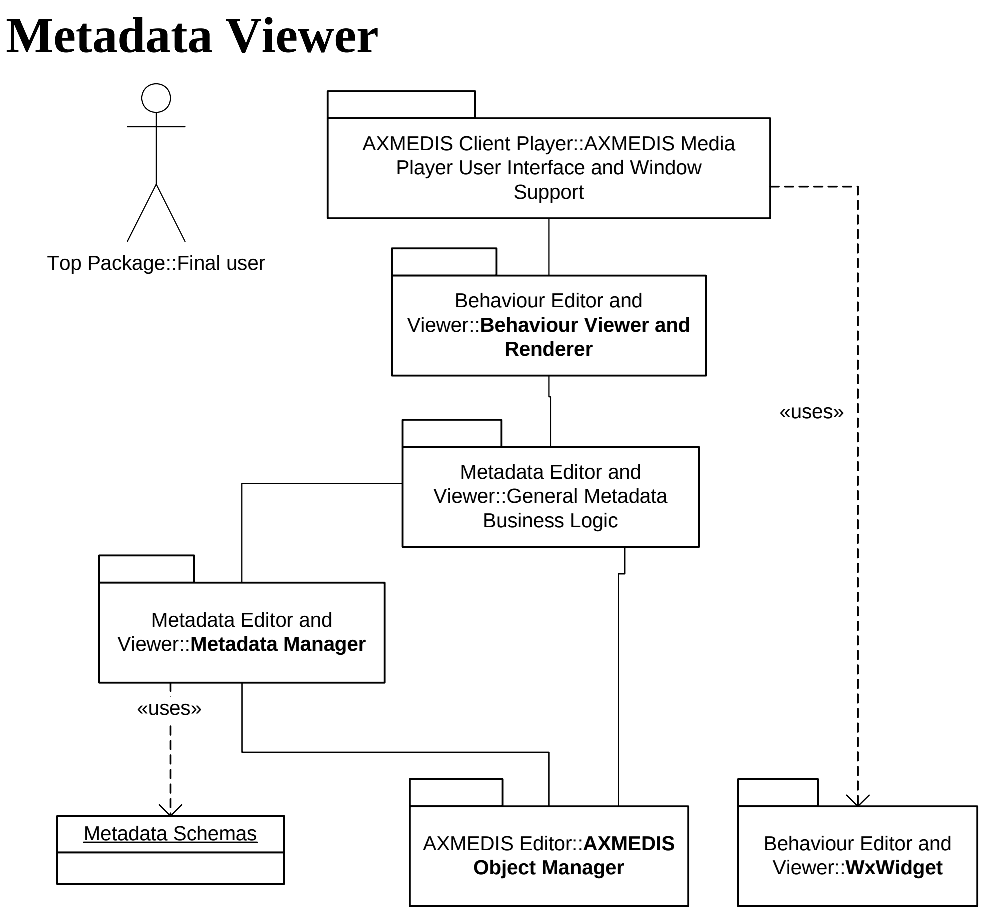
For Metadata visualisation, two possible approaches can be done:

1. Using the above editor without the manipulation and save functionality (4 & 5) activated
2. Opening the Metadata Viewer which renders the metadata in an HTML browser (with CSS)

Metadata Editor and Viewer


Top Package::Object Builder
Top Package::Final user
AXMEDIS Client Player::AXMEDIS Media Player User Interface and Window Support
AXMEDIS Editor::AXOM Commands and Reporting
Metadata Editor User Interface
Metadata Viewer and Renderer
«uses»
«uses»
AXMEDIS Editor::AXOM Content Processing
General Metadata Business Logic
Metadata Manager
AXMEDIS Editor::AXMEDIS Object Manager
Behaviour Editor and Viewer::WxWidget
«uses»
«uses»
Metadata Schemas

This architecture should allow to cope with different Metadata Sets simply by changing the Metadata Schemas (also taken from the AXMEDIS object, and in particular from the AXInfo): The interested test cases should be UNIMARC, Doublin Core, and all the AXInfo data. The Role of the Metadata Manager is that of reading the schema and creating data structure and logic on the basis of the General Metadata Business Logic. The Metadata Manager can have in the same AXMEDIS object different sections with different Metadata differentiated for: Model, language, schema, etc. The Metadata Viewer and Renderer can be a simple translator in HTML or XML and the real renderer can be the HTML renderer inside the AXMEDIS Media Player.


Metadata Viewer
Top Package::Final user
AXMEDIS Client Player::AXMEDIS Media Player User Interface and Window Support
Behaviour Editor and Viewer::Behaviour Viewer and Renderer
«uses»
Metadata Editor and Viewer::General Metadata Business Logic
Metadata Editor and Viewer::Metadata Manager
«uses»
Metadata Schemas
AXMEDIS Editor::AXMEDIS Object Manager
Behaviour Editor and Viewer::WxWidget

10.1.1 General Metadata Business Logic

The General Metadata Business Logic provides the navigation functionalities to traverse a given XML document based on the structure and relationships modelled by the Metadata Manager using a schema. It is particularly important for the Metadata Editor to know the valid children of a particular node where the user intends to add an element. The Business Logic preserves the structure integrity and ensures the correctness of the updated XML.

10.1.2 Metadata Manager

With a given schema, the Metadata Manager creates a representation of the structure and representation which include all the valid nodes, elements, parent child relationships, and each individual type. At this level the XML Document Object Model (DOM) is used to provide a way on how the XML document can be accessed and manipulated. The structure of this structure is used by the General Metadata Business Logic which navigates the structure. For the Metadata Editor, this structure, nodes, elements, etc are used to allow the user to add new elements with validation.

Metadata structure can be complex with recursive references. The AXMEDIS Metadata Manager extracts basic structure and applies a linearization to the structure in order to minimise unnecessary complexity unrelated to metadata editing purposes, removing recursive references.

10.1.3 Metadata Schemas

In this case, for the Metadata Editor, the Metadata Schema is required as a means for defining the structure and content of the XML documents. One or more metadata schema(s) is/are required for the AXMEDIS Metadata Editor in order to validate the correctness of the structure and elements of the metadata description of the AXMEDIS object. This is particularly important to allow the adding of a new element (which may be optional and not included in the original description).

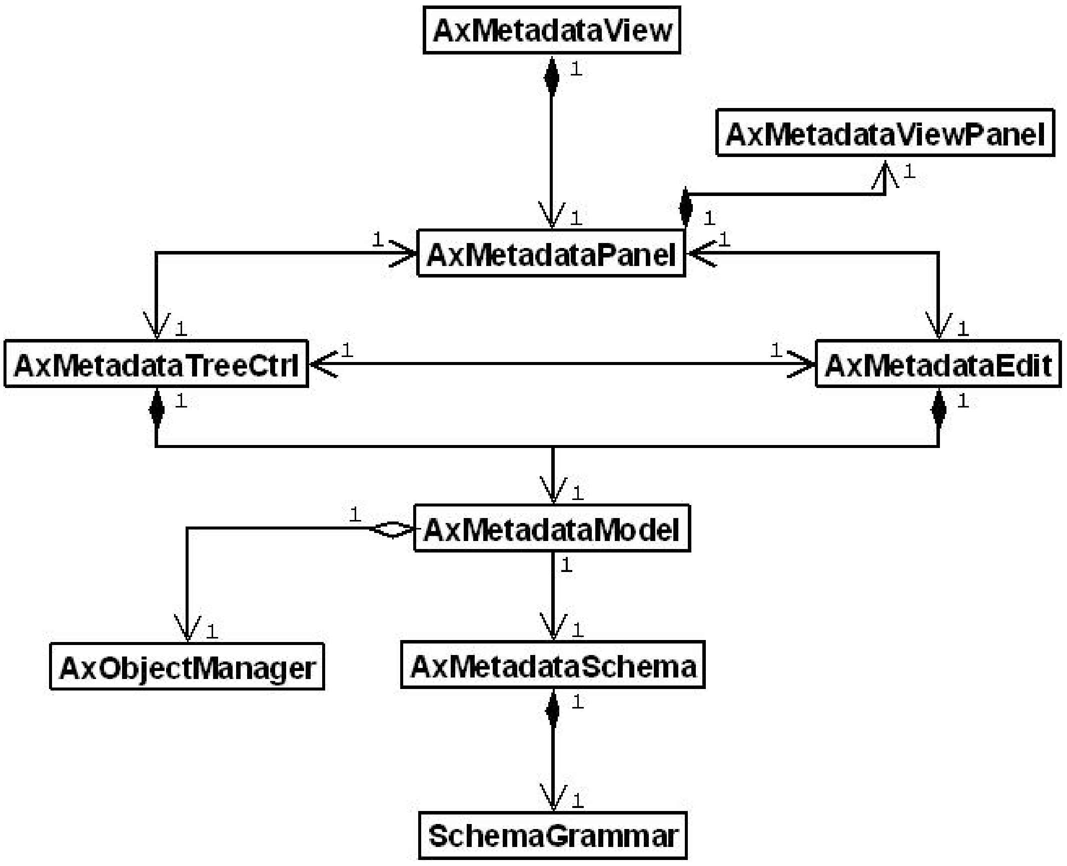
If no schema is available for the editor, the editor will still provide the functionality of modifying existing elements and try to preserve the original type. However, no new elements can be added since there is a potential danger of corrupting the original object description.

10.1.4 Metadata Viewer and Render

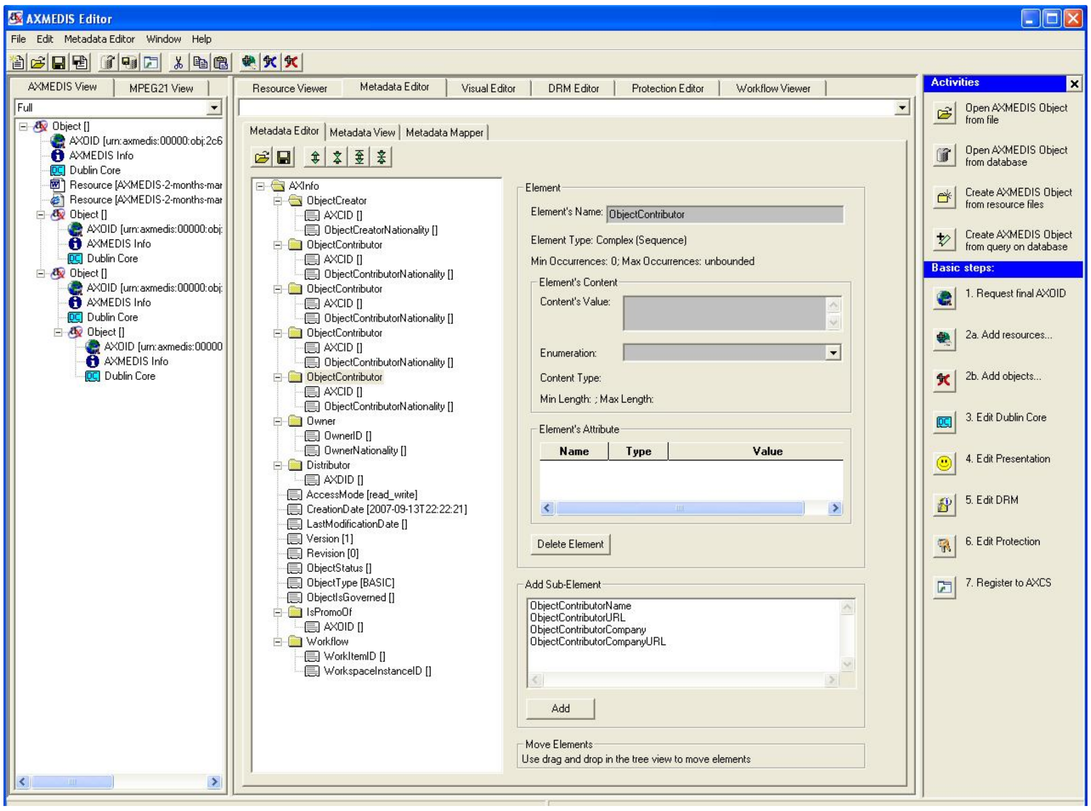
There are two approaches to achieve the metadata visualisation:

1. Using the metadata editor (as described above) without editing.
2. Opening the Metadata View tab to view metadata in an HTML embedded browser (with pre-defined or user-defined CSS).

10.2 Module Design in terms of Classes



10.3 User interface description



```
Object []
  AXOID [urn:axmedis:00000:obj:2c6
  AXMEDIS Info
  Dublin Core
  Resource [AXMEDIS-2-months-mar
  Resource [AXMEDIS-2-months-mar
  Object []
    AXOID [urn:axmedis:00000:obj:
    AXMEDIS Info
    Dublin Core
  Object []
    AXOID [urn:axmedis:00000:obj:
    AXMEDIS Info
    Dublin Core
    Object []
      AXOID [urn:axmedis:00000
      AXMEDIS Info
      Dublin Core
```

```
AXInfo
  ObjectCreator
    AXCID []
    ObjectCreatorNationality []
  ObjectContributor
    AXCID []
    ObjectContributorNationality []
  ObjectContributor
    AXCID []
    ObjectContributorNationality []
  ObjectContributor
    AXCID []
    ObjectContributorNationality []
  ObjectContributor
    AXCID []
    ObjectContributorNationality []
  Owner
    OwnerID []
    OwnerNationality []
  Distributor
    AXDID []
  AccessMode [read_write]
  CreationDate [2007-09-13T22:22:21]
  LastModificationDate []
  Version [1]
  Revision [0]
  ObjectStatus []
  ObjectType [BASIC]
  ObjectIsGoverned []
  IsPromoOf
    AXOID []
  Workflow
    WorkItemID []
    WorkspaceInstanceID []
```

Element
Element's Name: ObjectContributor
Element Type: Complex (Sequence)
Min Occurrences: 0; Max Occurrences: unbounded

Element's Content
Content's Value:
Enumeration:
Content Type:
Min Length: ; Max Length:

Element's Attribute

Name	Type	Value

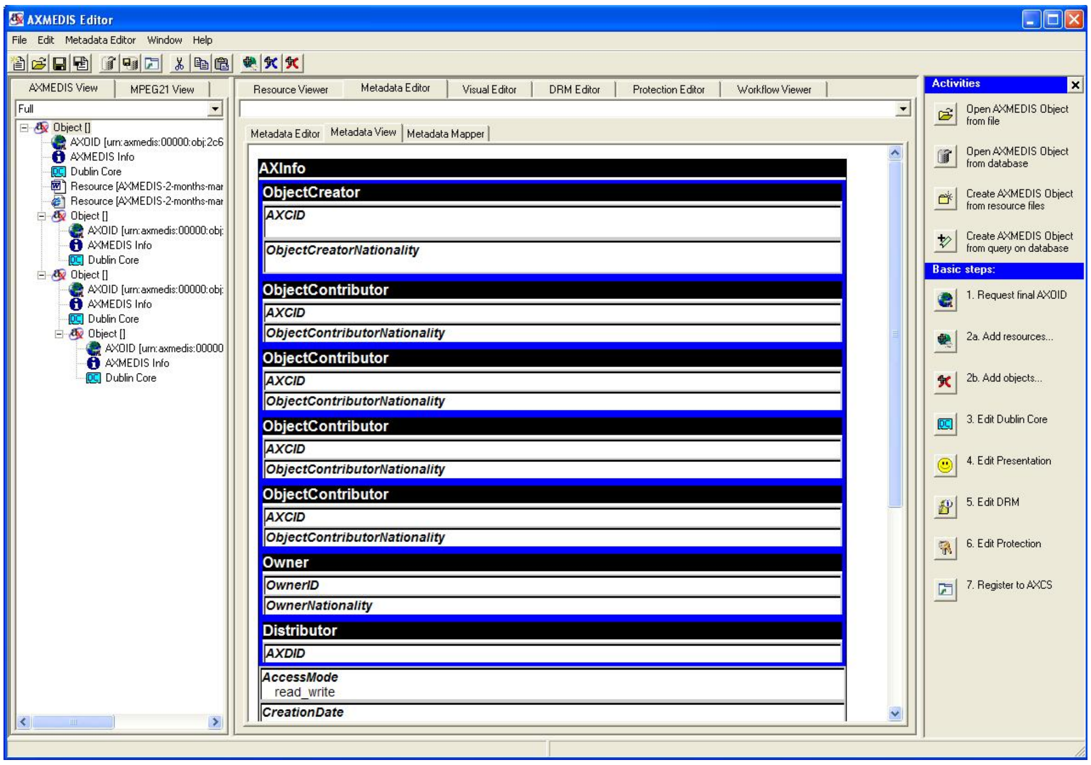
Delete Element

Add Sub-Element
ObjectContributorName
ObjectContributorURL
ObjectContributorCompany
ObjectContributorCompanyURL

Add

Move Elements
Use drag and drop in the tree view to move elements

Figure: Screenshot of the Metadata Editor opened in the AXMEDIS Editor showing the metadata elements (right window). The Metadata Editor can be opened by double clicking on the metadata set (e.g. AxInfo, Dublin Core, etc.) in AXMEDIS Hierarchy View of the AXMEDIS Object (left window).



```
AXInfo
  ObjectCreator
    AXCID
    ObjectCreatorNationality
  ObjectContributor
    AXCID
    ObjectContributorNationality
  ObjectContributor
    AXCID
    ObjectContributorNationality
  ObjectContributor
    AXCID
    ObjectContributorNationality
  ObjectContributor
    AXCID
    ObjectContributorNationality
  Owner
    OwnerID
    OwnerNationality
  Distributor
    AXDID
  AccessMode
    read_write
  CreationDate
```

Figure: By selecting the Metadata View tab, users can view the metadata elements in a visualised form.

10.4 Technical and Installation information

The Metadata Editor and Viewer are integrated into the AxEditor. Please see AxEditor technical and installation information for more details.

References to other major components needed	
Problems not solved	
Configuration and execution context	

10.5 Draft User Manual

The use of the Metadata Editor and Viewer is straightforward. See section 11.3 User Interface Description above and for more details, see AXMEDIS User Manual DE5-0-1-2 for AXTools.

10.6 Examples of usage

The use of the Metadata Editor and Viewer is straightforward. See section 11.3 User Interface Description above and for more details, see AXMEDIS User Manual DE5-0-1-2 for AXTools.

10.7 Integration and compilation issues

None

10.8 Configuration Parameters

Config parameter	Possible values

10.9 Errors reported and that may occur

Error code	Description and rationales

11 Module - Metadata Mapper Editor and Viewer (UNIVLEEDS)

Module Profile	
Metadata Mapper Editor and Viewer	
Responsible Name	Royce Neagle and Minh Thang Dang
Responsible Partner	UNIVLEEDS
Status (proposed/approved)	

Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multi Thread	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/metadatamapper_editor_viewer https://cvs.axmedis.org/newrepos/Framework/source/metadatamapper/	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/axeditor/bin	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved		
Major pending requirements	GUI improvements	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)

Formats Used	Shared with	format name or reference to a section
XSLT		XSLT, Standard format
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
yes	C++	wxWidgets, Object Graphics Library, PC,
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWindows	wxWindows, v-2.5.3 or v-2.4.2	LGPL
Xerces	Xerces-c++ v-2.6.0	Apache Software License, v2.0
Xalan	Xalan-c++ v-1.9	Apache Software License, v1.1

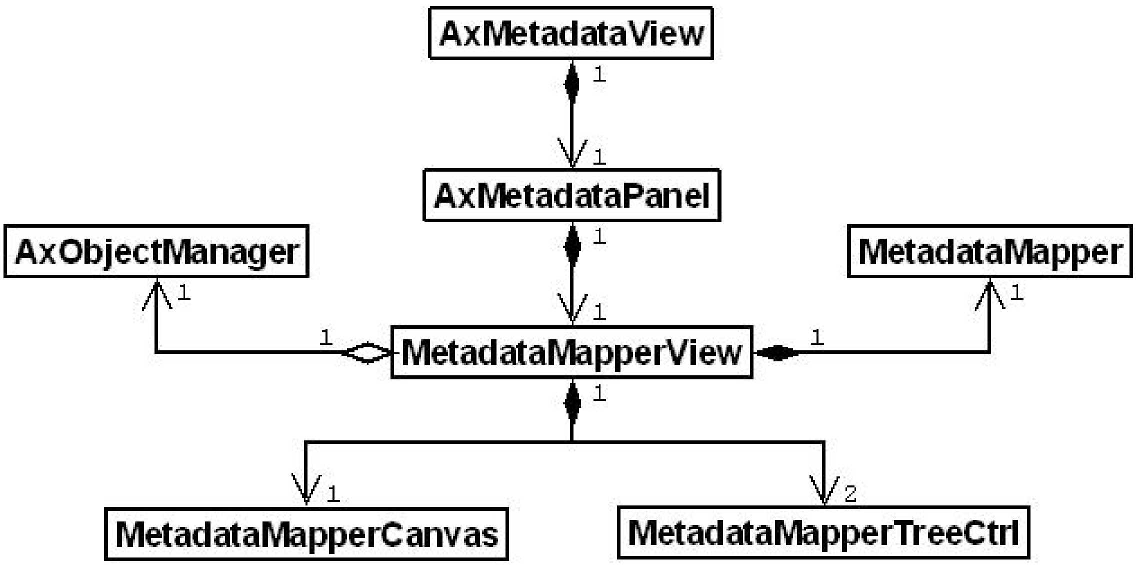
11.1.1 General Description of the Module

The Metadata Mapper has been integrated into the AxEditor. It allows a user to generate metadata mappings from one XML language to another using a graphical interface. Source and destination metadata files are loaded into the application and then the user defines links with pair(s) of a source element to a destination element. The mappings can be saved to file (XSLT) and this file can be used to convert a metadata file in the source language to a metadata file in the destination language.

The Mapper comprises of two parts:

1) A GUI which provides a graphical tool for mapping between one metadata set to another.
2) The MetadataMapper library (see DE3-1-2-2-7) which builds the XSLT document from the mapping information provided by the GUI

11.1.2 Module Design in terms of Classes



11.1.3 User interface description

The Metadata Mapper is integrated into the AxEditor. It is resided as a tab panel within the Metadata Editor. The user interface consists of three components:

- the left component displays a tree view of the source metadata language
- the middle component is a canvas used to display connections between elements on either side
- the right component displays a tree view of the target metadata language
- The user creates connections between source and target elements and a connection line is drawn on the canvas. These connections are converted to XSLT when the user generates a map file

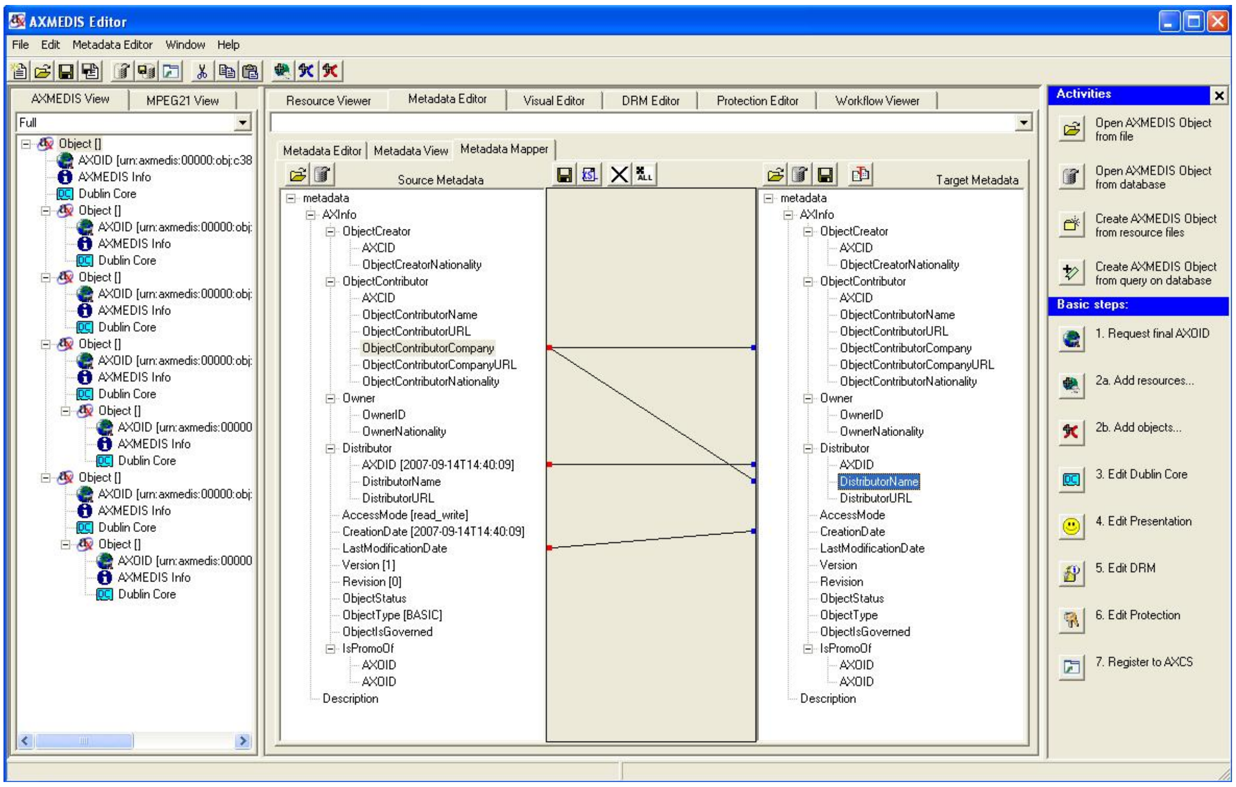


Figure: The Metadata Mapper integrated in the AxEditor.

The source and target metadata are initially loaded from the current top level metadata in the AXMEDIS Hierarchy View. Users can load other source and target metadata from local file system or remote database.

The XSLT file can be saved using the first two buttons in the middle pane. Connection links can be clear using the last two buttons in the middle pane.

11.1.4 Technical and Installation information

The Metadata Mapper is integrated into the AxEditor. Please see AxEditor technical and installation information for more details.

References to other major components needed	
Problems not solved	
Configuration and execution context	

11.1.5 Draft User Manual

See section 12.1.3 User Interface Description above and for more details, see AXMEDIS User Manual DE5-0-1-2 for AXTools.

11.1.6 Examples of usage

First a user loads a source and target metadata file. Next he connects elements to create mapping information. When the user has mapped all the required elements, he can save the XSLT file by clicking on the save button in the middle pane. The file will be saved on the user's file system which can be used for metadata adaptation.

11.1.7 Integration and compilation issues

None

11.1.8 Configuration Parameters

Config parameter	Possible values

11.1.9 Errors reported and that may occur

Error code	Description and rationales
5	`out of memory exception  when trying to create a new stylesheet`
2	DOM Exception `when trying to create a new stylesheet`
`error in generate xsl function`	`Undefined exception when trying to create a new stylesheet`

12 Module Workflow Editor and Viewer (DSI)

Module/Tool Profile

Workflow User Interface		
Responsible Name	Bellini	
Responsible Partner	DSI	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First Prototype Completed	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithreaded	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/workflow_editor_viewer	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/axeditor	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any		
Test cases (present/absent)	Absent	
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	Yes	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved		
Major pending requirements		
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
WorkFlow Engine		Via Workflow Plugin
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section

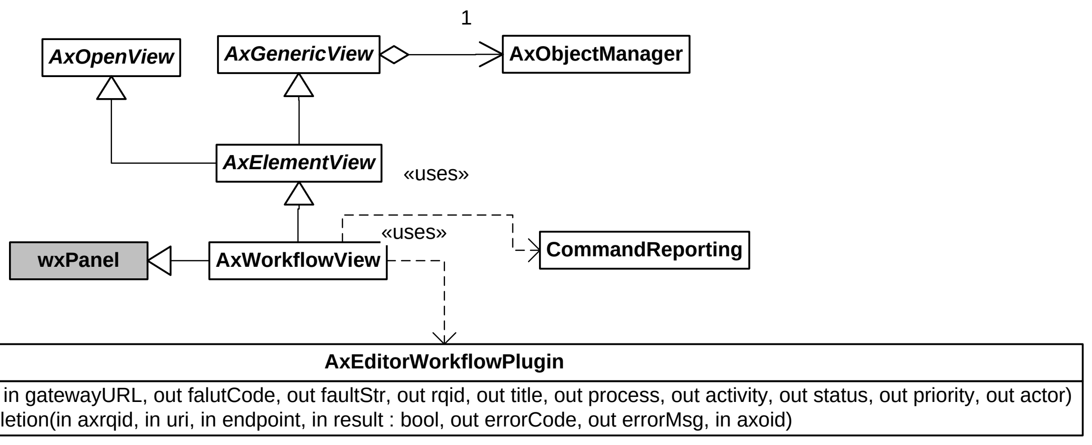
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
	C++	wxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
	wxWidgets 2.4.2	LGPL

12.1 General Description of the Module

The workflow editor and viewer integrated inside the AXMEDIS Editor is the gateway interface for accessing to the workflow management system.
The functionalities supported by the module are:

- view the status in the Workflow Management System of the currently opened object
- update inside the object the workflow status that is present in the AXInfo metadata
- give access to the workflow web interface

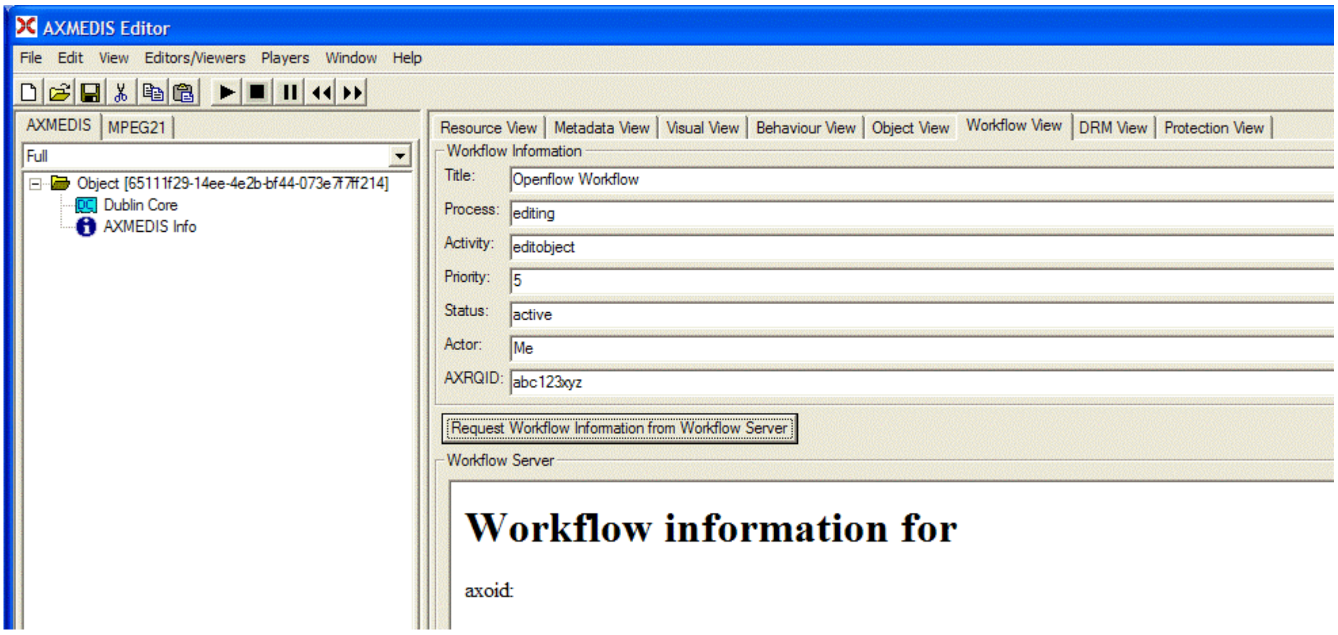
12.2 Module Design in terms of Classes



+getWFInfo(in axoid, in gatewayURL, out falutCode, out faultStr, out rqid, out title, out process, out activity, out status, out priority, out actor)
+notifyRequestCompletion(in axrqid, in uri, in endpoint, in result : bool, out errorCode, out errorMsg, in axoid)

12.3 User interface description

The User Interface integrated inside the AXMEDIS Editor is the following.


AXMEDIS Editor
File Edit View Editors/Viewers Players Window Help
AXMEDIS MPEG21
Full
Object [65111f29-14ee-4e2b-bf44-073e7f7ff214]
Dublin Core
AXMEDIS Info
Resource View Metadata View Visual View Behaviour View Object View Workflow View DRM View Protection View
Workflow Information
Request Workflow Information from Workflow Server
Workflow Server

Title: Openflow Workflow
Process: editing
Activity: editobject
Priority: 5
Status: active
Actor: Me
AXRQID: abc123xyz

Workflow information for

axoid:

When the user clicks on the "Request Workflow Information" Button, the server is contacted via the WF plugin to obtain the information on the object that is currently opened
The information acquired is:

- Title
- Process
- Activity
- Priority
- Status
- Actor
- AXRQID

If needed the information stored inside the AxInfo on the workflow is updated.

On the bottom part of the view the Workflow Server web interface is provided.

12.4 Technical and Installation information

References to other major components needed	
Problems not solved	•
Configuration and execution context	

12.5 Draft User Manual

See User Interface Description

12.6 Examples of usage

See User Interface Description

12.7 Integration and compilation issues

None

12.8 Configuration Parameters

Config parameter	Possible values
WORKFLOW/workflowUrl	the url of the workflow manager
WORKFLOW/gatewayUrl	the url of the gateway

12.9 Errors reported and that may occur

Error code	Description and rationales

13 Module - AXMEDIS Content Tool Error Manager (DSI)

Module/Tool Profile	
<name of the module>	
Responsible Name	Vallotti
Responsible Partner	DSI
Status (proposed/approved)	
Implemented/not implemented	not implemented
Status of the implementation	
Executable or Library/module (Support)	Library
Single Thread or Multithread	Single thread
Language of Development	C++
Platforms supported	
Reference to the AXFW location of the source code demonstrator	NA
Reference to the AXFW location of the demonstrator executable tool for internal download	NA
Reference to the AXFW location of the demonstrator executable tool for public download	
Address for accessing to WebServices if any, add accession information (user and Passwd) if any	
Test cases (present/absent)	absent
Test cases location	

Usage of the AXMEDIS configuration manager (yes/no)	no	
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Error Coding		Section 25
Error Log		Section **Errore. L'origine riferimento non è stata trovata.**
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
WxWIDGET	wxWidgets 2.4.2	LGPL
log4cxx		Apache License version 2.0

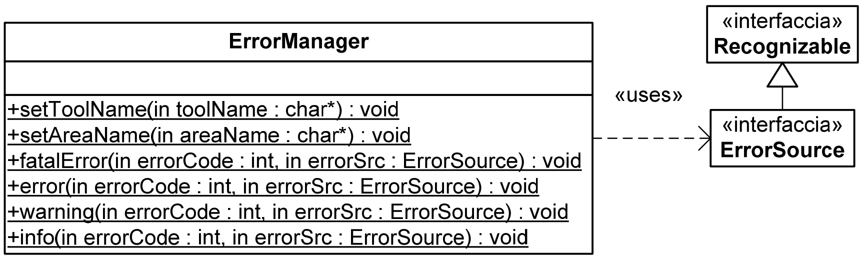
13.1 General Description of the Module

AXMEDIS Content Tool Error Manager is an interface through which all software modules are allowed to log errors in an independent way w.r.t. to the language. Moreover, the error manager allow the user to graphically visualize the error logs.
To achieve language independence errors have to be pre-defined and identified and, at runtime, they should be handle through identifiers instead of using their text descriptor. Therefore, in AXMEDIS Framework errors will be uniquely identified by the following information:

- the *area name* the error refers to, e.g. workflow, axeptool, editor, player, engine, scheduler, etc…
- the *full class name* of the class (i.e. the name comprehensive of containing namespace) the error is raised by
- an *error code*

As stated after in this section, error identifiers (area name, full class name, error code) are someway associated with other useful information such as a short description, recovery note, etc which are language dependant.
The above data can be statically defined in the code or dynamically determined at runtime. To determine that information at runtime an infrastructure for error management is needed. That infrastructure should consist of one or more classes which has to be used by all modules of AXMEDIS Framework (see below).

13.2 Module Design in terms of Classes

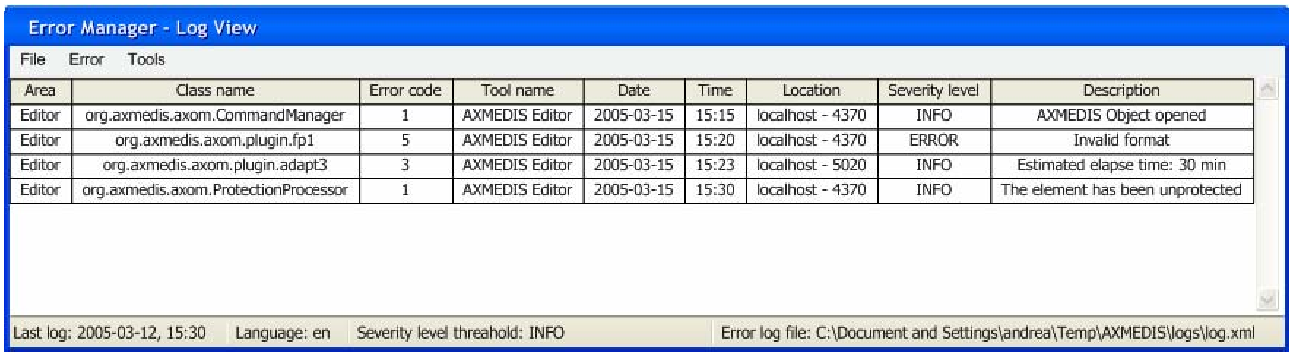

ErrorManager
+setToolName(in toolName : char*) : void
+setAreaName(in areaName : char*) : void
+fatalError(in errorCode : int, in errorSrc : ErrorSource) : void
+error(in errorCode : int, in errorSrc : ErrorSource) : void
+warning(in errorCode : int, in errorSrc : ErrorSource) : void
+info(in errorCode : int, in errorSrc : ErrorSource) : void
«uses»
«interfaccia»
Recognizable
«interfaccia»
ErrorSource

ErrorManager class exposes the following static functions

- **setToolName** – has to be called at tool start-up time. In that way, every time an error is logged the **ErrorManager** knows the tool name avoiding hardly readable and repetitive code.
- **setAreaName** – has to be called at tool start-up time. In that way, every time an error is logged the **ErrorManager** knows the area name avoiding hardly readable and repetitive code.
- **fatalError, error, warning, info** – those functions work in the same way the unique differences is the severity level of the logged error they produce. Using four different functions we avoid the need to use an additional parameter for the functions and the code will result much more readable. The first parameter of those functions is the error source, i.e. the instance which wants to raise the error, the second parameter is the error code, which will be merged with other information to determine the error identifier.

As depicted in the figure above, **ErrorSource** is an interface extending the interface Recognizable thus **ErrorManager** is able to retrieve information on the class without boring the programmer with to much code.

13.3 User interface description



Area	Class name	Error code	Tool name	Date	Time	Location	Severity level	Description
Editor	org.axmedis.axom.CommandManager	1	AXMEDIS Editor	2005-03-15	15:15	localhost - 4370	INFO	AXMEDIS Object opened
Editor	org.axmedis.axom.plugin.fp1	5	AXMEDIS Editor	2005-03-15	15:20	localhost - 4370	ERROR	Invalid format
Editor	org.axmedis.axom.plugin.adapt3	3	AXMEDIS Editor	2005-03-15	15:23	localhost - 5020	INFO	Estimated elapse time: 30 min
Editor	org.axmedis.axom.ProtectionProcessor	1	AXMEDIS Editor	2005-03-15	15:30	localhost - 4370	INFO	The element has been unprotected

Last log: 2005-03-12, 15:30 Language: en Severity level threahold: INFO Error log file: C:\Document and Settings\andrea\Temp\AXMEDIS\logs\log.xml

The AXMEDIS Error Manager User Interface allows the user to visualize and manage the logged error and to handle log-related options. The user interface reads data from error definition files and error log file (whose format are specified in this section) and mixes that information to render the errors in a human-readable format.
The GUI exposes the following functionalities:

- Menu *File* – it contains file-related actions:
  - *Load* – loads an error log file and visualize it
  - *Save* – saves the visualized logs on a given location
- Menu *Error* – it contains error-related actions:
  - *Flush* – deletes all the logs in the opened error log file
  - *Delete* – deletes the selected error from the opened error log file
  - *More info…* - opens a dialog which shows detailed information on the error
  - *Order by…* - the user the ordering criteria of the logged error table
- Menu *Tools* – allows to configure the Error Manager and the GUI. In particular, it allows to modify the following options:
  - *Language* – sets the language used for the description field and for the "More info…" dialog
  - *Severity threshold level* – only error whose severity level is greater than the threshold are showed in the table. The severity level are those defined in the error log schema
  - *Redirection* – the user can decide where error should be logged. He/She can choose among: local redirection, remote redirection and both redirections
  - *Log activation* – the user can activate/deactivate the log mechanism

Moreover, ordering action can be directly performed on the table as well as display of "More info" dialog.

13.4 Draft User Manual

See User Interface Description

13.5 Examples of usage

See User Interface Description

13.6 Configuration Parameters

Config parameter	Possible values
ERROR_MANAGER – LOG_FILE_PATH	Any valid path representing the directory containing the log file
ERROR_MANAGER – LOG_FILE_NAME	Any valid name for the log file
ERROR_MANAGER – LANGUAGE	Any valid language code as defined in the schema
ERROR_MANAGER – SEVERITY_LEVEL	One of the following strings: info, warning, error, fatal

13.7 Errors reported and that may occur

Error code	Description and rationales

14 Module - AXMEDIS Editor Configuration Manager (DSI)

Module/Tool Profile		
AXMEDIS Editor Configuration Manager		
Responsible Name	Vallotti	
Responsible Partner	DSI	
Status (proposed/approved)		
Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported		
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/common	
Reference to the AXFW location of the demonstrator executable tool for internal download		
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)	absent	
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	no	
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
XML based for configuration info	All the Editors tools, editors and viewers written in C++ and related AXOM tools	Section **Errore. L'origine riferimento non è stata trovata.**

Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
For Error log	C++	WxWIDGET
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
WxWIDGET	wxWidgets 2.4.2	LGPL

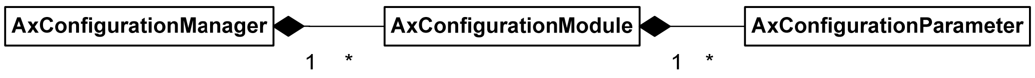
14.1 General Description of the Module

AXMEDIS Editor Configuration Manager will be the unique access point to AXMEDIS Editor modules configurations. Each configurable AXMEDIS Editor modules (and sub modules) will respect AXMEDIS Editor Configuration Manager requirements.
AXMEDIS Editor Configuration Manager User Interface will be flexible enough to manage the widest range of settings possible. That should be possible by means of plug-in support.
Further, AXMEDIS Editor Configuration Manager will manage and provide AXMEDIS Editor general configurations, such as multi-language.

- AXMEDIS Editor Configuration Manager shall be the unique configuration access point;
- AXMEDIS Editor Configuration Manager shall provide an interface to allow configuration access and modification (AXMEDIS Editor Configuration Manager User Interface);
- AXMEDIS Editor Configuration Manager shall include a default settings module which shall be configurable by XML schema-like language. In such a way, every AXMEDIS Editor module shall specify it’s own settings

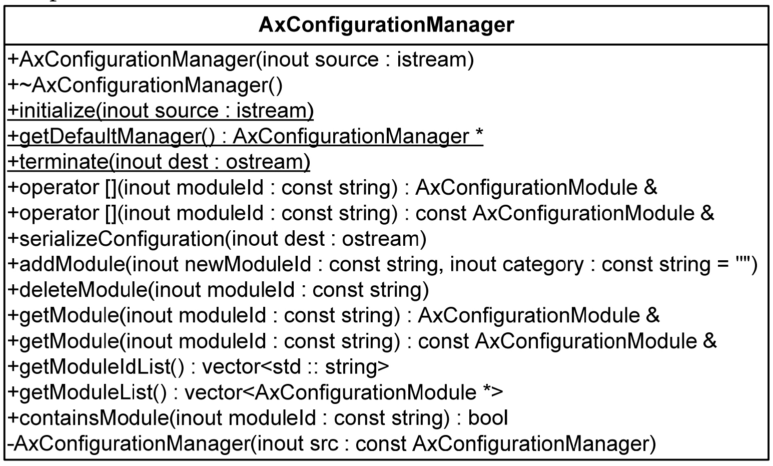
14.2 Module Design in terms of Classes

In this sub-section, the structure of classes to manage the configuration file is presented. A stated in the introduction, these classes has to allow to manage single parameters and set of parameters called module. The class hierarchy reflect the structure of the XML file which contains the configuration parameters which schema is described in the following sub-section.



The Configuration Manager is composed of three main classes:

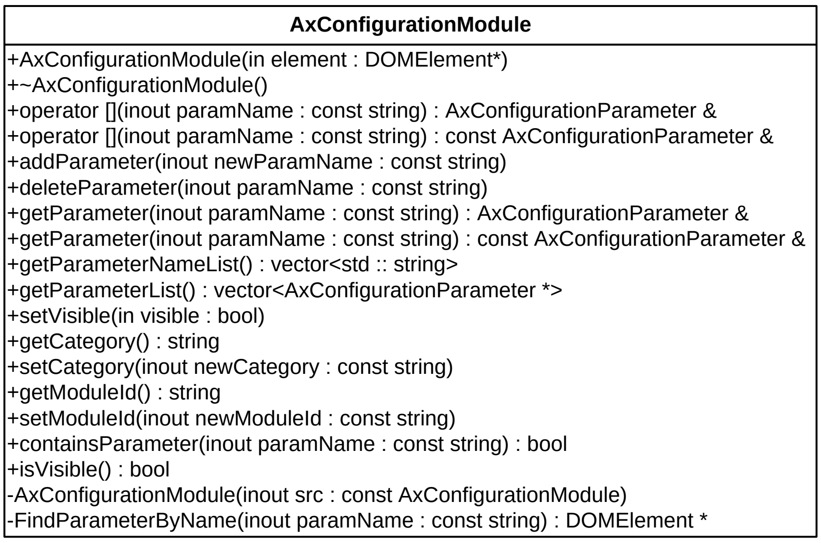
- *AxConfigurationManager* – it is the main class through which all modules and parameters are reachable
- *AxConfigurationModule* – it represents a set of parameters related each others, e.g. all the parameters related to a specific software module
- *AxConfigurationParameter* – it represent one parameter, it provides function to easily manage different types of parameter.



```
AxConfigurationManager
+AxConfigurationManager(inout source : istream)
+~AxConfigurationManager()
+initialize(inout source : istream)
+getDefaultManager() : AxConfigurationManager *
+terminate(inout dest : ostream)
+operator [](inout moduleId : const string) : AxConfigurationModule &
+operator [](inout moduleId : const string) : const AxConfigurationModule &
+serializeConfiguration(inout dest : ostream)
+addModule(inout newModuleId : const string, inout category : const string = "")
+deleteModule(inout moduleId : const string)
+getModule(inout moduleId : const string) : AxConfigurationModule &
+getModule(inout moduleId : const string) : const AxConfigurationModule &
+getModuleIdList() : vector<std :: string>
+getModuleList() : vector<AxConfigurationModule *>
+containsModule(inout moduleId : const string) : bool
-AxConfigurationManager(inout src : const AxConfigurationManager)
```

AxConfigurationManager has been designed as a Singleton (see Design Patterns) in order to allow everyone to reach it within an application. It exposes the following methods to allow management of a set of configurations and parameters:

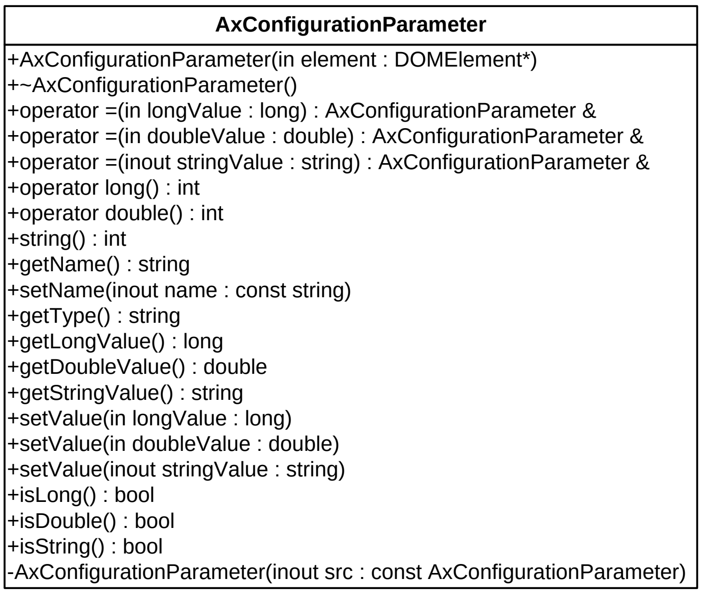
- *AxConfigurationManager* – it creates an instance of *AxConfigurationManager*, while doing that it parses and loads a set of configurations from the given input stream. This constructor can be used to create configuration manager other than the default one.
- *initialize* – it initializes the default configuration manager using the given input stream by calling the provided constructor on a specific file. This function has to be called before any other.
- *getDefaultManager* – it returns the default manager, if it has been initialized.
- *terminate* – it deletes the default manager. This function has to be called after any other.
- *serializeConfiguration* – serializes the whole set of configurations on the given output stream formatted as described in the following sub-section;
- *addModule* – creates a new module of parameters having the given name (*newModuleId*) and the given category (*category*). The module name have to be unique than the function does not create two modules with the same name. The module category could be a string like a file path. In this way, categories and sub-categories can be easily created and managed.
- *containsModule* – checks if the configuration manager contains a module with the given name (*moduleId*).
- *deleteModule* – if exists, removes the module with the given name from the configuration manager;
- *getModule* and *[]* operator – both functions allow to get an instance of *AxConfigurationModule* which represent the module having the given name (*moduleId*). Acting on the obtained instance of *AxConfigurationModule*, it is possible to manage the parameters contained in the corresponding module of the configuration manager.
- *getModuleIdList* – it returns the id list of all modules contained in this manager.
- *getmoduleList* – it returns the list of all modules contained in this manager.



```
AxConfigurationModule
+AxConfigurationModule(in element : DOMElement*)
+~AxConfigurationModule()
+operator [](inout paramName : const string) : AxConfigurationParameter &
+operator [](inout paramName : const string) : const AxConfigurationParameter &
+addParameter(inout newParamName : const string)
+deleteParameter(inout paramName : const string)
+getParameter(inout paramName : const string) : AxConfigurationParameter &
+getParameter(inout paramName : const string) : const AxConfigurationParameter &
+getParameterNameList() : vector<std :: string>
+getParameterList() : vector<AxConfigurationParameter *>
+setVisible(in visible : bool)
+getCategory() : string
+setCategory(inout newCategory : const string)
+getModuleId() : string
+setModuleId(inout newModuleId : const string)
+containsParameter(inout paramName : const string) : bool
+isVisible() : bool
-AxConfigurationModule(inout src : const AxConfigurationModule)
-FindParameterByName(inout paramName : const string) : DOMElement *
```

AxConfigurationModule exposes the following methods to allow management of a module of parameters contained in the owner configuration manager:

- *addParameter* – creates a new parameter having the given name (*newParamName*). The parameter name have to be unique within the module than the function does not create two parameter with the same name.
- *containsParameter* – checks if the module contains a parameter with the given name (*paramName*).
- *deleteParameter* – if exists, removes the parameter with the given name from the module;
- *getParameter* and *[]* operator – both functions allow to get an instance of *AxConfigurationParameter* which represent the parameter having the given name (*paramName*). Acting on the obtained instance of *AxConfigurationParameter*, it is possible to manage the parameter contained in the owner module.
- *getParameterNameList* – it returns the name list of all parameter contained in this module.
- *getParameterNameList* – it returns the list of all parameter contained in this module.
- *setVisible* – it sets the visible attribute of this module.
- *isViisble* – it tests whether the visible attribute of this module is true or not.
- *setCategory* – it sets the category attribute of this module with the given string.
- *getCategory* – it returns the value of the category attribute of this module.
- *setModuleId* – it sets the identifier of this module.
- *getModuleId* – it returns the identifier of this module.

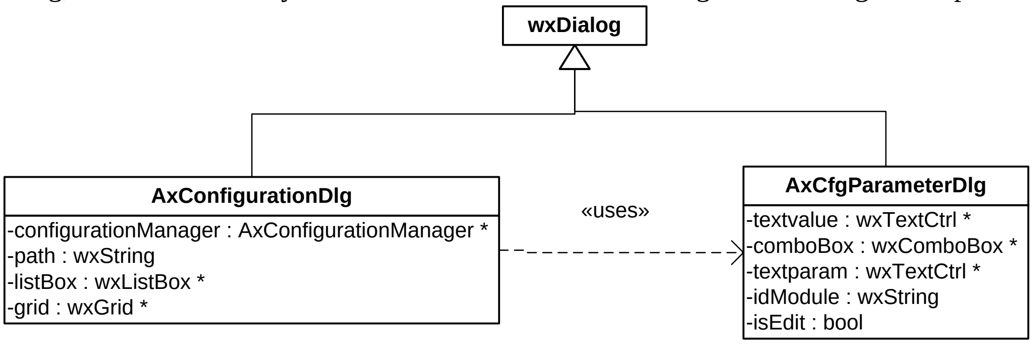


```
AxConfigurationParameter
+AxConfigurationParameter(in element : DOMElement*)
+~AxConfigurationParameter()
+operator =(in longValue : long) : AxConfigurationParameter &
+operator =(in doubleValue : double) : AxConfigurationParameter &
+operator =(inout stringValue : string) : AxConfigurationParameter &
+operator long() : int
+operator double() : int
+string() : int
+getName() : string
+setName(inout name : const string)
+getType() : string
+getLongValue() : long
+getDoubleValue() : double
+getStringValue() : string
+setValue(in longValue : long)
+setValue(in doubleValue : double)
+setValue(inout stringValue : string)
+isLong() : bool
+isDouble() : bool
+isString() : bool
-AxConfigurationParameter(inout src : const AxConfigurationParameter)
```

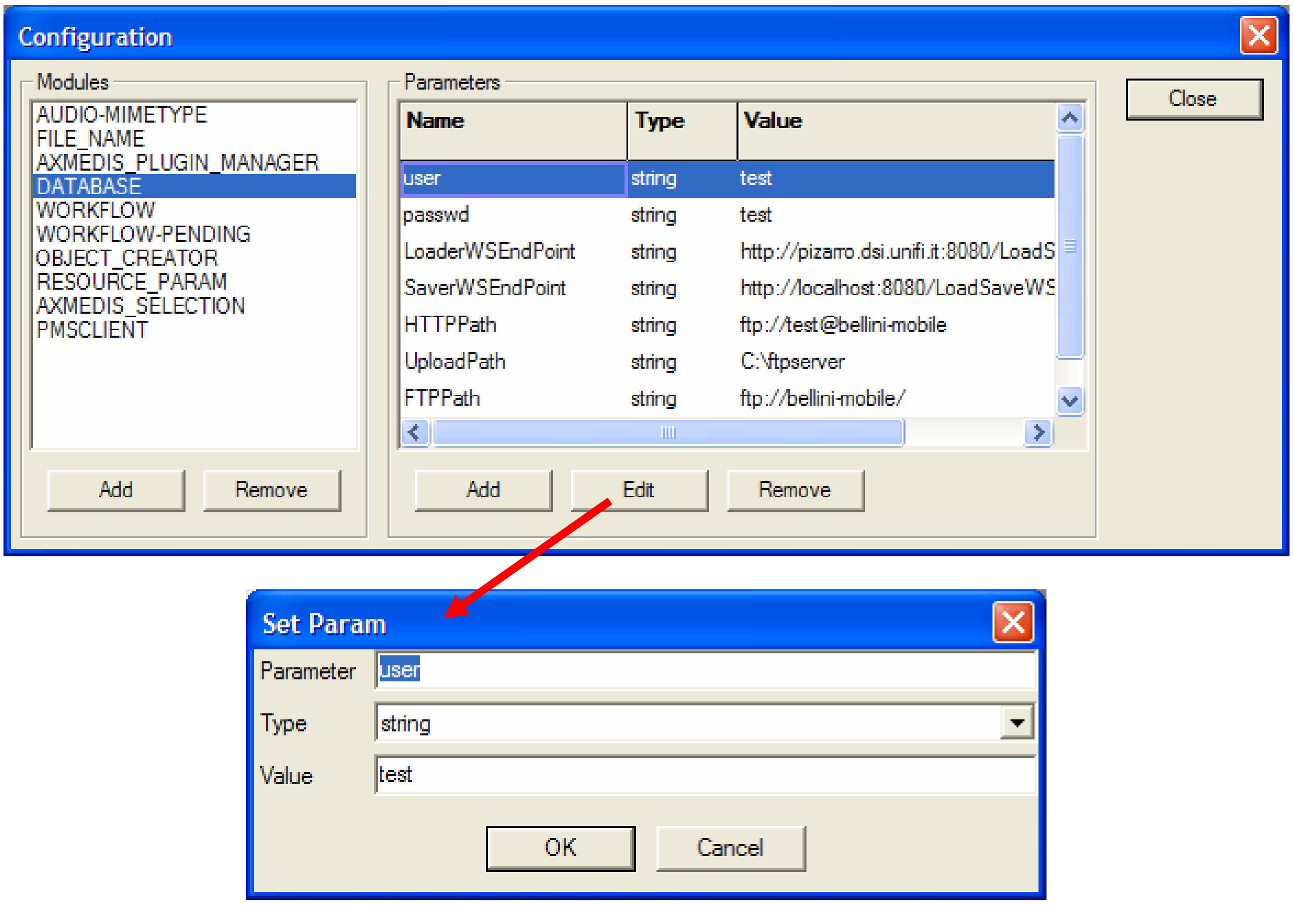
AxConfigurationParameter exposes several functions to allow management of different type of parameter in an easy manner. In particular, it exposes methods to set and get the value of a parameter and to obtain the name and the type of the parameter itself.

14.3 User interface description

In the following, the class hierarchy for the user interface of the Configuration Manger is depicted.



- AXMEDIS Editor Configuration Manager User Interface shall provide an interface to allow development of plug-ins for specific kind (or set) of settings;
- AXMEDIS Editor Configuration Manager User Interface will be capable to correctly display module settings;
- AXMEDIS Editor Configuration Manager shall show all configuration in an user friendly manner, e.g. dived by categories;



Modules:
AUDIO-MIMETYPE
FILE_NAME
AXMEDIS_PLUGIN_MANAGER
DATABASE
WORKFLOW
WORKFLOW-PENDING
OBJECT_CREATOR
RESOURCE_PARAM
AXMEDIS_SELECTION
PMSCLIENT

Name	Type	Value
user	string	test
passwd	string	test
LoaderWSEndPoint	string	http://pizarro.dsi.unifi.it:8080/LoadS
SaverWSEndPoint	string	http://localhost:8080/LoadSaveWS
HTTPPath	string	ftp://test@bellini-mobile
UploadPath	string	C:\ftpserver
FTPPath	string	ftp://bellini-mobile/

Parameter	user
Type	string
Value	test

In the above figure, the user interface of AXMEDIS Configuration Manager is depicted. It is composed of a list box with the modules, and a grid where the parameters of the selected module are displayed.

14.4 Technical and Installation information

References to other major components needed	
Problems not solved	•
Configuration and execution context	

14.5 Examples of usage

The following piece of code shows how Configuration Manager can be used.

```
std::ifstream configuration;
configuration.open("in_configuration.xml");
AxConfigurationManager manager(configuration);
configuration.close();

std::vector<std::string> idList=manager.getModuleIdList();
if(idList[0]=="ID000000")
    std::cout << “Test 1 ok”;
AxConfigurationModule& module1=manager.getModule("ID000000");
if(module1.getModuleId()=="ID000000")
    std::cout << “Test 2 ok”;
if(((long) module1.getParameter("parameterLong"))==1977)
    std::cout << “Test 3 ok”;
if(((double) module1.getParameter("parameterDouble"))==108.5)
    std::cout << “Test 4 ok”;
if(((std::string) module1.getParameter("parameterString"))=="Andrea Vallotti")
    std::cout << “Test 5 ok”;
std::vector<std::string> nameList=module1.getParameterNameList();
if(nameList[0]==" parameterLong")
    std::cout << “Test 6 ok”;
manager.serializeConfiguration(std::cout);
manager.deleteModule("ID000000");
manager.serializeConfiguration(std::cout);
std::cout << std::endl;
manager.addModule("ID001000","foo/footwo");
manager.getModule("ID001000").addParameter("fooLong");
manager.getModule("ID001000")["fooLong"]=1945L;
manager.getModule("ID001000").addParameter("fooDouble");
manager.getModule("ID001000")["fooDouble"]=103.3;
std::ofstream outconf;
outconf.open("out_configuration.txt");
manager.serializeConfiguration(outconf);
outconf.close();
```

14.6 Integration and compilation issues

Currently the Configuration Manager uses the Xerces-C++ library. Therefore, it properly works on Windows and Linux systems. Probably it should be changed in order to be used on system with lower resource, such as Mobile and PDA.

14.7 Configuration Parameters

Config parameter	**Possible values**

14.8 Errors reported and that may occur

Error code	Description and rationales

15 Module - AXMEDIS Editor Plug-in Manager (DSI)

Module/Tool Profile		
AXMEDIS Editor Plug-in Manager		
Responsible Name	Vallotti	
Responsible Partner	DSI	
Status (proposed/approved)		
Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Single thread	
Language of Development	C++	
Platforms supported	Windows, Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/pluginmanager/	
Reference to the AXFW location of the demonstrator executable tool for internal download		
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)		
Usage of the AXMEDIS Error Manager (yes/no)		
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Plugin Description		Section **Errore. L'origine riferimento non è stata trovata.**

Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

15.1 General Description of the Module

AXMEDIS Editor has been thought to be as versatile and flexible as possible. In order to achieve this goal, various AXMEDIS Editor modules need to support plug-in technology. Hence, a AXMEDIS Editor Plug-in Manager is needful, such manager will be able to support installation/registration of plug-ins, to load such plug-ins for AXMEDIS Editor modules which request it and to maintain/manage relationship among plug-ins and related entities or actions, e.g. AXMEDIS Editor Plug-in Manager shall maintain relation among a specific set of metadata and the corresponding production or visualization plug-ins.

- AXMEDIS Editor Plug-in Manager shall provide general mechanisms in order to manage different kind of plug-ins;
- Plug-in Manger shall provide standard interface definition which will be common among all the plug-in families;
- Plug-in Manger shall provide an interface to allow interested AXMEDIS Editor modules to access to associated plug-ins.
- AXMEDIS Editor Plug-in Manager shall store all those information needful to classify and sort plug-ins, such as:
    - Identifier;
    - Provider;
    - Category;
    - Etc…

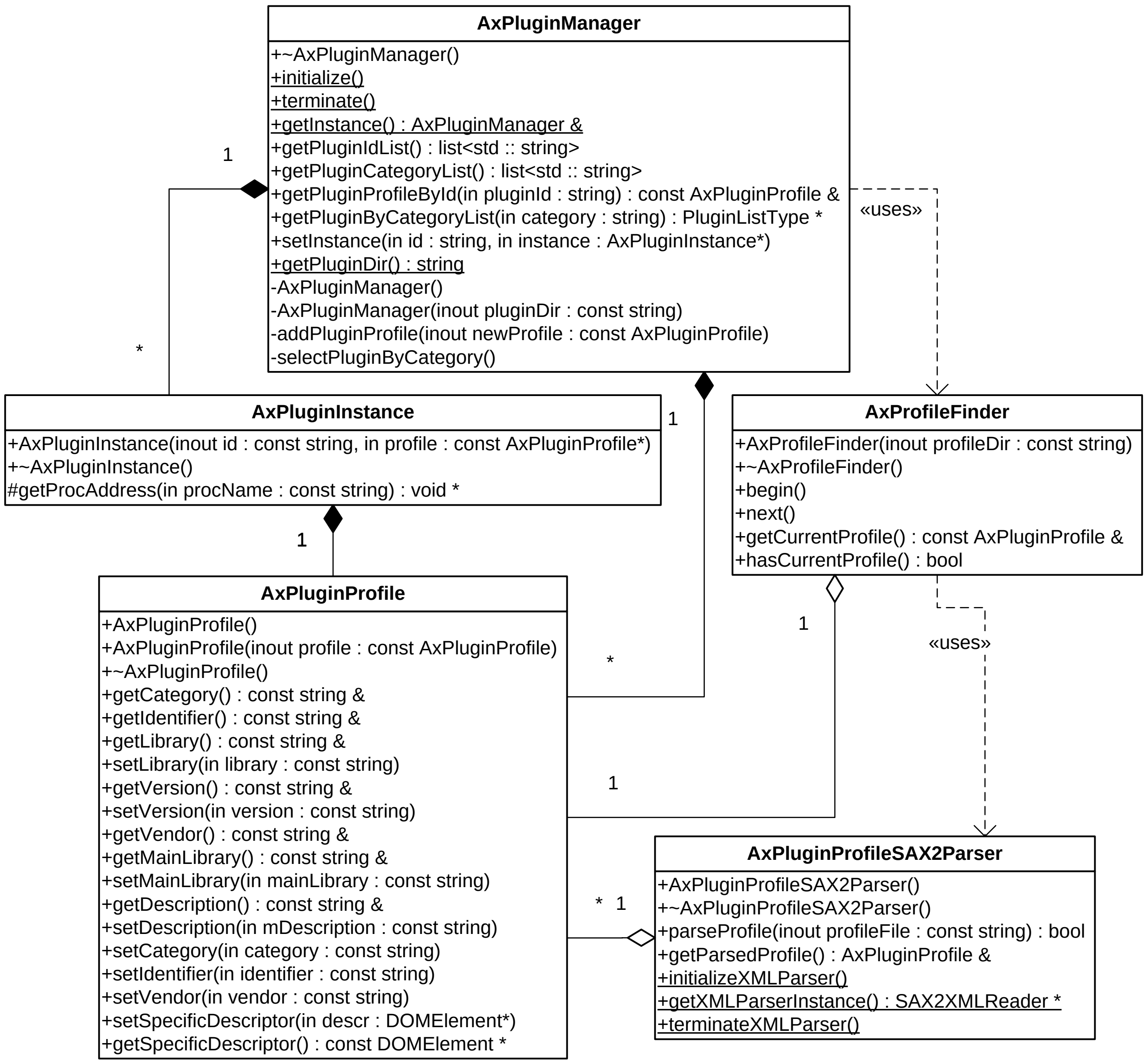
Each plug-in is described by a XML file (namely a profile) which contains the following common information:

- the category of the plug-in, e.g. content processing;

- the unique identifier of the plug-ins, e.g. an URI like a XML namespace;
- the signature of the plug-in evaluated by an AXCS;
- data specific for the kind of plug-in (see below);
- the signature, estimated by the AXCS, of the whole XML file.

The profile shall provide a placeholder for custom information which can vary on the basis of plug-in category. The definition of general plug-in profile schema is reported in section “Formal description of format – Plugins description”.

15.2 Module Design in terms of Classes


AxPluginManager
+~AxPluginManager()
+initialize()
+terminate()
+getInstance() : AxPluginManager &
+getPluginIdList() : list<std :: string>
+getPluginCategoryList() : list<std :: string>
+getPluginProfileById(in pluginId : string) : const AxPluginProfile &
+getPluginByCategoryList(in category : string) : PluginListType *
+setInstance(in id : string, in instance : AxPluginInstance*)
+getPluginDir() : string
-AxPluginManager()
-AxPluginManager(inout pluginDir : const string)
-addPluginProfile(inout newProfile : const AxPluginProfile)
-selectPluginByCategory()
«uses»
AxPluginInstance
+AxPluginInstance(inout id : const string, in profile : const AxPluginProfile*)
+~AxPluginInstance()
#getProcAddress(in procName : const string) : void *
AxProfileFinder
+AxProfileFinder(inout profileDir : const string)
+~AxProfileFinder()
+begin()
+next()
+getCurrentProfile() : const AxPluginProfile &
+hasCurrentProfile() : bool
«uses»
AxPluginProfile
+AxPluginProfile()
+AxPluginProfile(inout profile : const AxPluginProfile)
+~AxPluginProfile()
+getCategory() : const string &
+getIdentifier() : const string &
+getLibrary() : const string &
+setLibrary(in library : const string)
+getVersion() : const string &
+setVersion(in version : const string)
+getVendor() : const string &
+getMainLibrary() : const string &
+setMainLibrary(in mainLibrary : const string)
+getDescription() : const string &
+setDescription(in mDescription : const string)
+setCategory(in category : const string)
+setIdentifier(in identifier : const string)
+setVendor(in vendor : const string)
+setSpecificDescriptor(in descr : DOMElement*)
+getSpecificDescriptor() : const DOMElement *
AxPluginProfileSAX2Parser
+AxPluginProfileSAX2Parser()
+~AxPluginProfileSAX2Parser()
+parseProfile(inout profileFile : const string) : bool
+getParsedProfile() : AxPluginProfile &
+initializeXMLParser()
+getXMLParserInstance() : SAX2XMLReader *
+terminateXMLParser()
1
*
1
1
*
1
1
* 1

In the above diagram the plug-in manager structure is depicted. That diagram provides base classes which allows to construct more complicated architecture. In particular, fundamental element of that approach are the following:

- *AxPluginManager* – it is the main class of the module which links plug-ins with their profiles. Moreover, it allows to access to plug-ins and plug-ins profiles by category.
- *AxPluginProfile* – it allows to retrieve data contained in the plug-in common profile, such as identifier, category, etc.;
- *AxPluginInstance* – it is the super-class of all specific plug-in instance classes. For example, content processing plug-ins are modeled as class (*AxCPPluginInstance*) instances. This class derives from *AxPluginInstance*. In this way, plug-ins instances can be transparently managed by Plug-in Manager

and new plug-in categories can be easily added. *AxPluginInstance* provides basic means to store dynamic library handle and to retrieve methods exposed by the latter;

Moreover, the module contains other two utility classes, that is:

- *AxProfileFinder* – it crawls a given directory looking for available plug-in profiles;
- *AxPluginProfileSAX2parser* – given an XML file which complies the plug-in profile common schema, it loads the data contained in it setting the attribute of an instance of *AxPluginProfile* class.

Plug-in certification functionalities are based on the services provided by the Protection Processor which needs to access to profile of the plug-in.

It has to be pointed out that the functions exposed by a given plug-in depend by the category of the plug-in itself.

15.2.1 Fundamental classes overview

AxPluginManager

It has been designed as a Singleton (see Design Patterns) in order to ensure its uniqueness in the application scope. Moreover, it can be invoked from anywhere supposing it has been already initialized. This class exposes the following methods:

- *initialize* – this static function initializes the Plug-in Manager. It has to be called before any other method calls.
- *terminate* – this static function terminates the Plug-in Manager by deleting all stored data.
- *getInstance* – this static function allows to access to the unique instance of *AxPluginManager*.
- *getPluginIdList* – it returns the id list of all found plug-ins.
- *getPluginCategoryList* – it returns the list of all found plug-in categories.
- *getPluginProfileById* – it returns the general profile of the plug-in with the given identifier.
- *getPluginByCategoryList* – it returns the list of all plug-ins (if already loaded) and their profiles belonging to the given category.
- *setInstance* – it allows to set the plug-in instance (*AxPluginInstance*) corresponding to the given identifier. This functions should be called by who is able to load plug-ins of the appropriate category.

The others are utility functions.

AxPluginProfile

It provides getter and setter methods to access common profile data. The setter methods are used by the profile loader during parsing, while getter methods are used by the rest of the application.

AxPluginInstance

The purpose of this class is twofold. From one side, as stated above, it allows to transparently manage different classes representing plug-in instances. On the other side, it provides and indirection layer between the platform-dependent mechanism for the management of dynamic libraries (e.g. library handle, function retrieving methods, etc…) and the child classes of *AxPluginInstance*. In fact, it mainly provides two methods:

- *AxPluginInstance* – this constructor allows to associate the given profile to this plug-in instance. Moreover, using the profile and the given working directory, it loads the related dynamic library.
- *getProcAddress* – it is an utility function which allows to retrieve pointers to functions exposed by the dynamic library the plug-in instance refers to. That function avoids the usage of platform-dependent mechanism, i.e. it acts as an indirection layer.

15.3 Integration and compilation issues

This modules needs Xerces-C++ library in order to parse plug-in profiles. Moreover, it needs the common library since it uses the Configuration Manager. These two libraries are platform independent, therefore they can be used on Windows and Linux systems. Moreover, this module uses platform-dependant libraries in order to manage dynamic libraries.

15.4 Configuration Parameters

Config parameter	Possible values
`AXMEDIS_PLUGIN_MANAGER - PLUGINS_PATH`	Any directory containing plug-ins and their profiles

15.5 Errors reported and that may occur

Error code	Description and rationales
0	Unable to load the dynamic library related to the given plug-in profile
1	Unable to unload the dynamic library related to the given plug-in profile
2	Plug-in Manager has been already initialized
3	Plug-in Manager has not been initialized
4	Any plug-in belonging to this category has been found
5	Plug-in not found
6	Plug-in identifier already present
7	Any parsed profile: *parseProfile* has not been called, or an error occurred during parsing
8	*next* function has been called before begin function or the current search is invalid

16 Module - AXOM Content processing (DSI)

Module/Tool Profile		
AXOM Content processing		
Responsible Name	Vallotti	
Responsible Partner	DSI	
Status (proposed/approved)		
Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Single thread	
Language of Development	C++	
Platforms supported	Windows, Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/contentprocessing/	
Reference to the AXFW location of the demonstrator executable tool for internal download		
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)		
Usage of the AXMEDIS Error Manager (yes/no)		
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Content Processing Plug-ins specific description		Section 29
Parameter description		Section 30

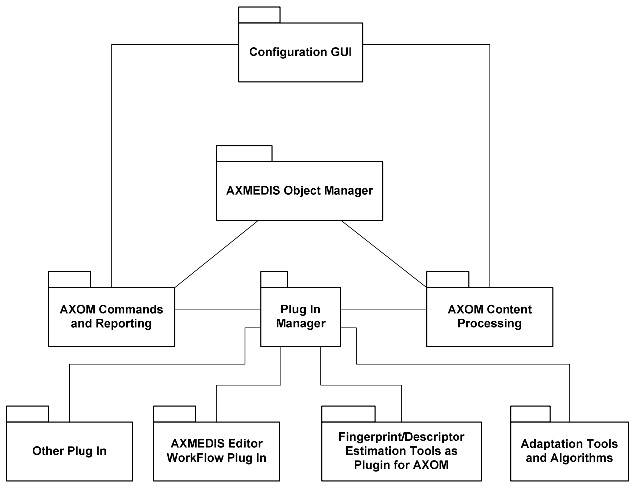
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

16.1 General Description of the Module

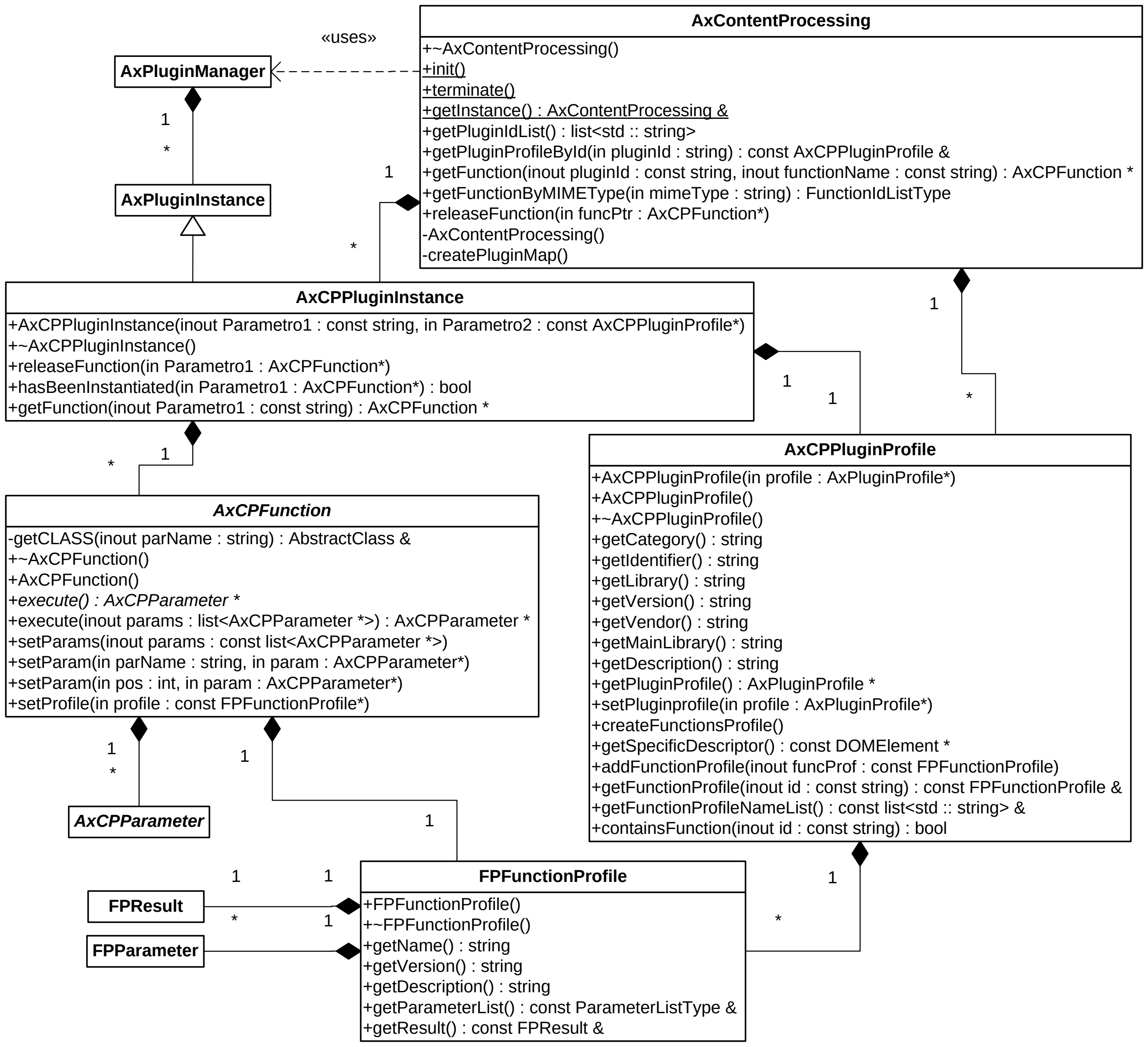
AXOM Content Processing is the interface the AXOM uses to call dynamic functions for content processing. Content processing function belongs to the following categories: Fingerprint estimation, Descriptor estimation, Adaptation algorithm, etc…

AXOM Content Processing is in charge of managing plug-ins for processing resources. It uses the plug-in manager in order to retrieve all plug-ins belonging to content processing category. In that way, content processing can manage and arrange the content processing functions on the base of relevant information such as, for example, MIME type of the resource to which the function applies, etc..

AXOM uses AXOM Content Processing function to access functions on the base of its needs and of the requests it receives from the user.


Configuration GUI
AXMEDIS Object Manager
AXOM Commands and Reporting
Plug In Manager
AXOM Content Processing
Other Plug In
AXMEDIS Editor WorkFlow Plug In
Fingerprint/Descriptor Estimation Tools as Plugin for AXOM
Adaptation Tools and Algorithms

16.2 Module Design in terms of Classes

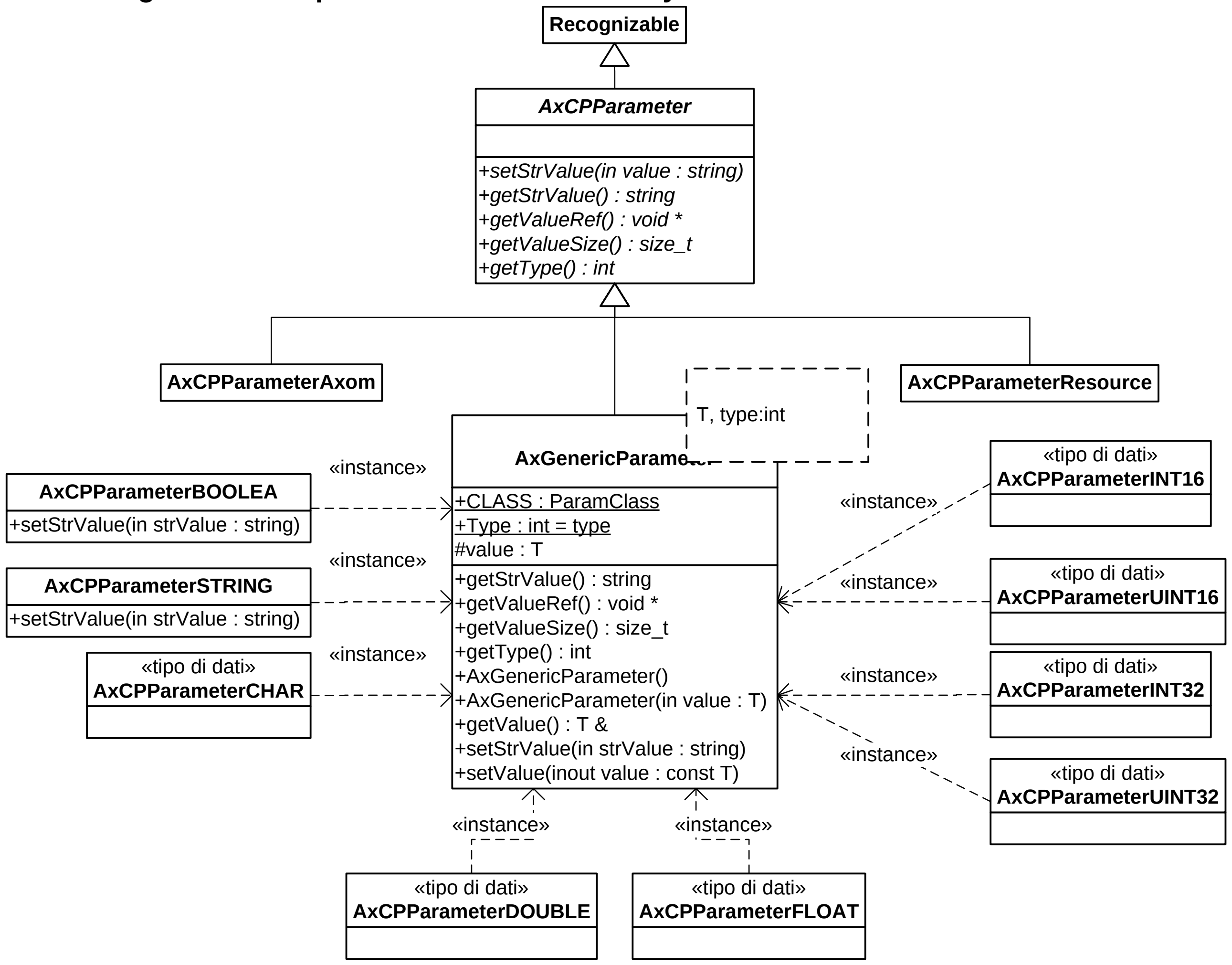

«uses»
AxPluginManager
AxPluginInstance
AxContentProcessing
+~AxContentProcessing()
+init()
+terminate()
+getInstance() : AxContentProcessing &
+getPluginIdList() : list<std :: string>
+getPluginProfileById(in pluginId : string) : const AxCPPluginProfile &
+getFunction(inout pluginId : const string, inout functionName : const string) : AxCPFunction *
+getFunctionByMIMEType(in mimeType : string) : FunctionIdListType
+releaseFunction(in funcPtr : AxCPFunction*)
-AxContentProcessing()
-createPluginMap()
AxCPPluginInstance
+AxCPPluginInstance(inout Parametro1 : const string, in Parametro2 : const AxCPPluginProfile*)
+~AxCPPluginInstance()
+releaseFunction(in Parametro1 : AxCPFunction*)
+hasBeenInstantiated(in Parametro1 : AxCPFunction*) : bool
+getFunction(inout Parametro1 : const string) : AxCPFunction *
AxCPPluginProfile
+AxCPPluginProfile(in profile : AxPluginProfile*)
+AxCPPluginProfile()
+~AxCPPluginProfile()
+getCategory() : string
+getIdentifier() : string
+getLibrary() : string
+getVersion() : string
+getVendor() : string
+getMainLibrary() : string
+getDescription() : string
+getPluginProfile() : AxPluginProfile *
+setPluginprofile(in profile : AxPluginProfile*)
+createFunctionsProfile()
+getSpecificDescriptor() : const DOMElement *
+addFunctionProfile(inout funcProf : const FPFunctionProfile)
+getFunctionProfile(inout id : const string) : const FPFunctionProfile &
+getFunctionProfileNameList() : const list<std :: string> &
+containsFunction(inout id : const string) : bool
AxCPFunction
-getCLASS(inout parName : string) : AbstractClass &
+~AxCPFunction()
+AxCPFunction()
+execute() : AxCPParameter *
+execute(inout params : list<AxCPParameter *>) : AxCPParameter *
+setParams(inout params : const list<AxCPParameter *>)
+setParam(in parName : string, in param : AxCPParameter*)
+setParam(in pos : int, in param : AxCPParameter*)
+setProfile(in profile : const FPFunctionProfile*)
AxCPParameter
FPResult
FPParameter
FPFunctionProfile
+FPFunctionProfile()
+~FPFunctionProfile()
+getName() : string
+getVersion() : string
+getDescription() : string
+getParameterList() : const ParameterListType &
+getResult() : const FPResult &
1
*

As shown in the class diagram, the Content Processing module is composed by the following classes:

- *AxContentProcessing* – it is the main class of the module. It provides methods allowing to access dynamic processing functions. In particular, it manages plug-ins belonging to "ContentProcessing" category.

- *AxCPPluginInstance* – it is an extension of *AxPluginInstance*. An instance of this class represents a loaded content processing plug-in. This class allows exploiting specific functions of this kind of plug-ins (e.g. create and release Content Processing functions).
- *AxCPFunction* – it represent Content Processing Function. It provides method: to set function parameters; to execute the function; and to obtain the result. It checks the given parameter verifying they respect what is stated in the relate function profile.
- *AxCPParameter* – it is the common class of all parameter classes which can be passed as argument to Content Processing Functions (see next section).
- *AxCPPluginProfile* – it wraps *AxPluginProfile* in order to allow access to generic and specific data in the plug-in description. In particular, it provides methods to get the name of contained functions and to get the description of any of them (given the related name).
- *FPFunctionProfile* – it provides methods to access information about the related function, see section 29 for more details about function profile data.

16.2.1 Plug-in function parameters class hierarchy


Recognizable
AxCPParameter
+setStrValue(in value : string)
+getStrValue() : string
+getValueRef() : void *
+getValueSize() : size_t
+getType() : int
AxCPParameterAxom
AxCPParameterResource
T, type:int
AxGenericParameter
+CLASS : ParamClass
+Type : int = type
#value : T
+getStrValue() : string
+getValueRef() : void *
+getValueSize() : size_t
+getType() : int
+AxGenericParameter()
+AxGenericParameter(in value : T)
+getValue() : T &
+setStrValue(in strValue : string)
+setValue(inout value : const T)
«instance»
AxCPParameterBOOLEA
+setStrValue(in strValue : string)
AxCPParameterSTRING
+setStrValue(in strValue : string)
«tipo di dati» AxCPParameterCHAR
«tipo di dati» AxCPParameterINT16
«tipo di dati» AxCPParameterUINT16
«tipo di dati» AxCPParameterINT32
«tipo di dati» AxCPParameterUINT32
«tipo di dati» AxCPParameterDOUBLE
«tipo di dati» AxCPParameterFLOAT

For each parameter type defined in parameter description schema (see section 30), a specific class has to be used in the programming language. Those classes derive from a common base class (*AxCPParameter*) which provides a basic interface for all possible parameter types. Moreover *AxCPParameter* derives from *Recognizable* which is an interface which allows to perform type-related operations on the objects (e.g. something like reflection in Java and C#).
The associations among class and parameter types is depicted below:

- *AxGenericParameter* – it is a template class which have been used, through a set of *typedef* definition, to wrap the basic data type, that is: UNIT16, INT16, UINT32, INT32, CHAR, BOOLEAN, FLOAT, DOUBLE and STRING.

- *AxCPParameterResource* – it represents a RESOURCE. It exposes functions to access the related digital resource. By "access" is meant to read and write the resource by means of streams.
- *AxCPParameterAxom* – it represents a parameter of type AXOM.

It is worth to point out that a Content Processing Plug-in have to export the following functions:

```
extern "C" AxCPFunction* GetPluginFunction(std::string funcName)
extern "C" void releasePluginFunction(AxCPFunction* function)
```

`GetPluginFunction` allows creating Content Processing Function instances by providing the function name (`funcName`).
`releasePluginFunction` releases the given `function` instance. This function has been introduced for the function instances are created by the dynamic library and they have to be deleted by it.

16.3 Errors reported and that may occur

Error code	Description and rationales
0	Content Processing has been already initialized
1	Content processing has not been initialized
2	Plug-in not found
3	The specified parameter is not present in function profile
4	Parameter is not of the right type
5	Unable to identify the requested type
6	Result has never been initialized
7	The parameter is mandatory therefore it cannot have a default value
8	The given function has not been instantiated by this plug-in

17 Module - AXOM Commands and Reporting (DSI)

Module/Tool Profile	
AXOM Commands and Reporting	
Responsible Name	Rogai
Responsible Partner	DSI
Status (proposed/approved)	
Implemented/not implemented	Implemented
Status of the implementation	
Executable or Library/module (Support)	Library
Single Thread or Multithread	Multithread
Language of Development	C++
Platforms supported	Windows, linux
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/commandreporting
Reference to the AXFW location of the demonstrator executable tool for internal download	
Reference to the AXFW location of the demonstrator executable tool for public download	
Address for accessing to WebServices if any, add accession information (user and Passwd) if any	

Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)		
Usage of the AXMEDIS Error Manager (yes/no)		
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

17.1 General Description of the Module

AXOM Commands and Reporting has the same role as AXOM Content Processing instead it is responsible to manage dynamic functions related to workflow system. It is implemented in the same way of AXOM Content Processing and it is the intermediate between AXOM and Plug-in Manager. In fact, it is a common service module in order to hide low-level plug-in manager functionalities, while providing an easy mechanism in order to allow different AXMEDIS tool (e.g. Editor, AXCP Rule Editor, AXCP Rule Scheduler) to load the suitable plug-ins enabling a workflow-based control and notification.

17.2 Module Design in terms of Classes

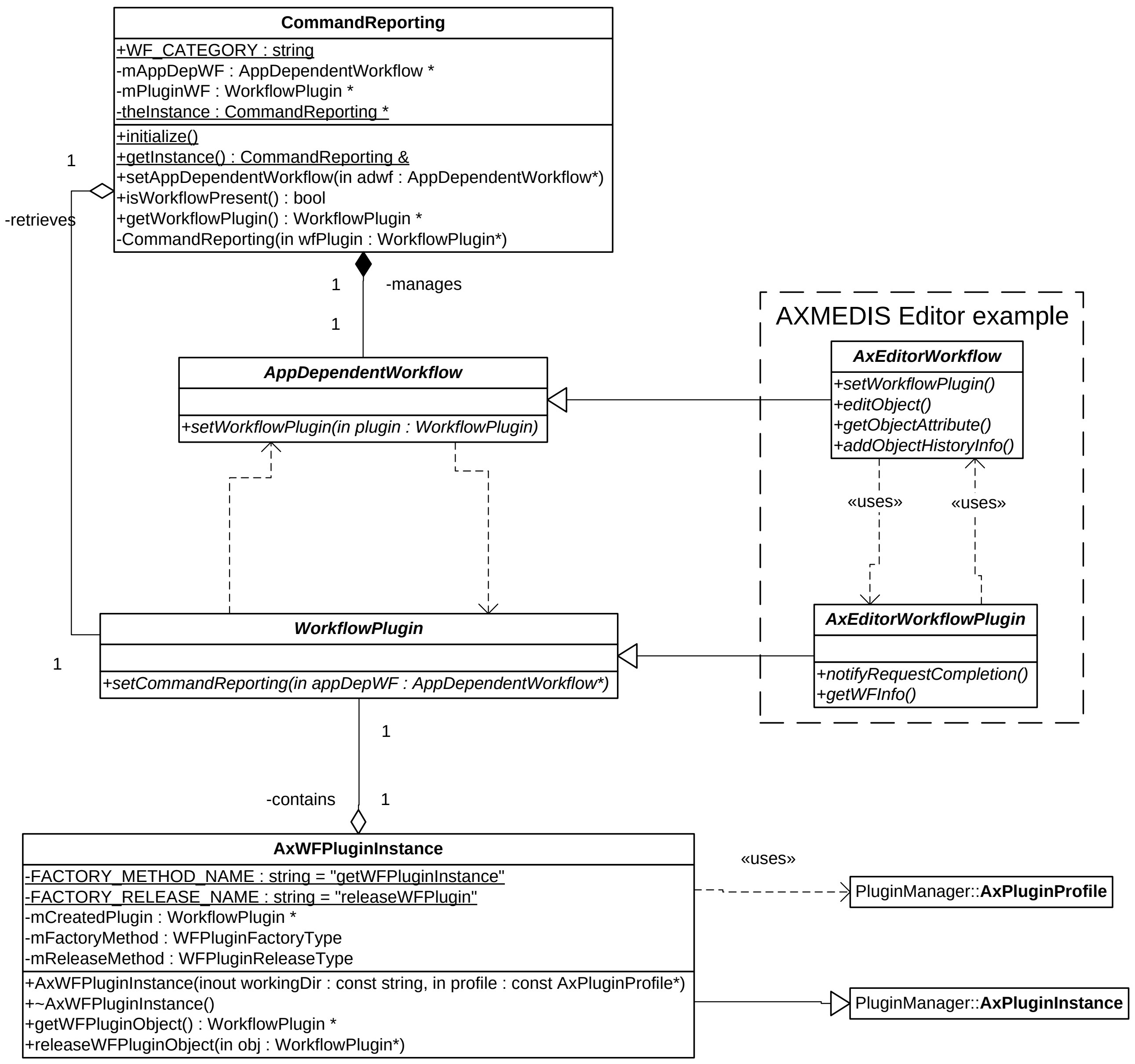
The main class in terms of exposed APIs is CommandReporting. This class implements a singleton capable of retrieving the workflow plug-in. This entity is responsible of connecting the workflow counter-parts (application and plug-in side). When it is initialized, it uses the AxPluginManager services in order to explore the available plug-ins finding to suitable one. After it has located the plug-in component it stores informations needed for an on-demand instantiation.

Two classes for workflow counterparts have been designed:

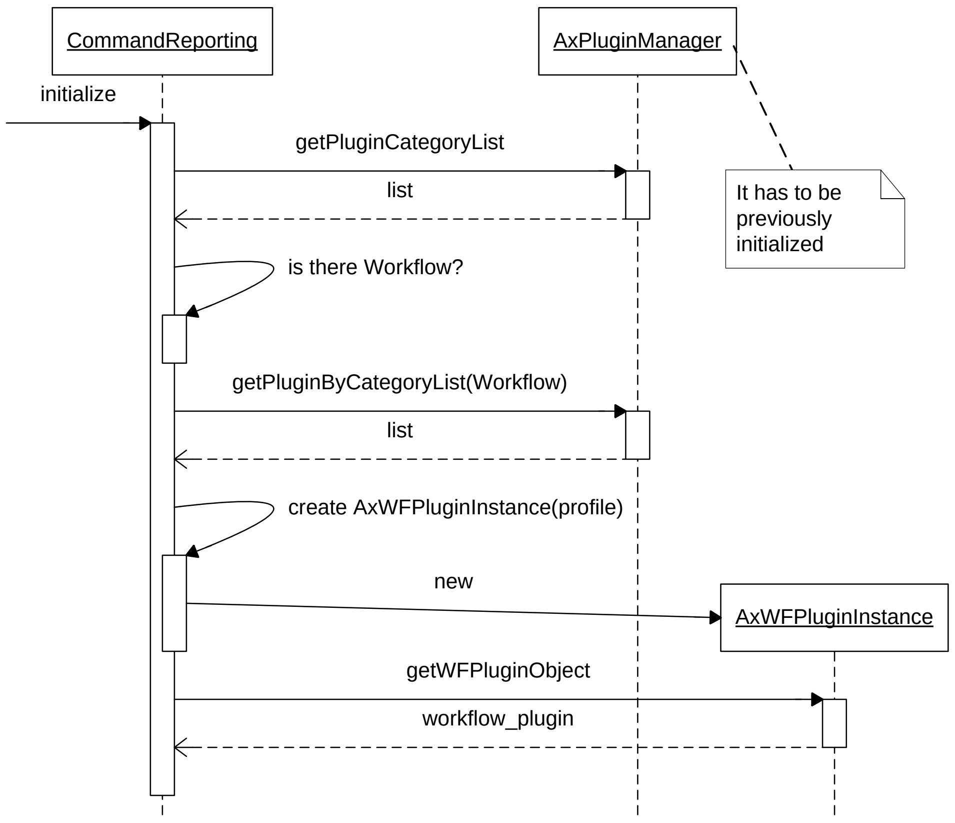
- AppDependentWorkflow: it models a generic workflow handler in the application (i.e. who is responsible of handling the request coming from the plug-in in the application context). It is an abstract class to be inherited by any AXMEDIS application. It has a general dependency to the general WorkflowPlugin.
- WorkflowPlugin: it models a generic workflow plug-in it exposes a unique link method that is known by the CommandReporting. Any plug-in component can implement a factory method to provide a WorkflowPlugin inherited istance.

Responsible of creating a WorkflowPlugin is a WFPluginInstance, which is obtained by AxPluginManager. The information to retrieve such a plug-in module are expressed by an XML profile, parsed and readed by AxPluginProfile.

For example, in the class diagram has been depicted the new classes for definining the AxEditor workflow. A specific AxEditorWorkflow inherited class has to be created where the “actual” action to be performed in the application context after a workflow request for the AXMEDIS Editore are implemented.

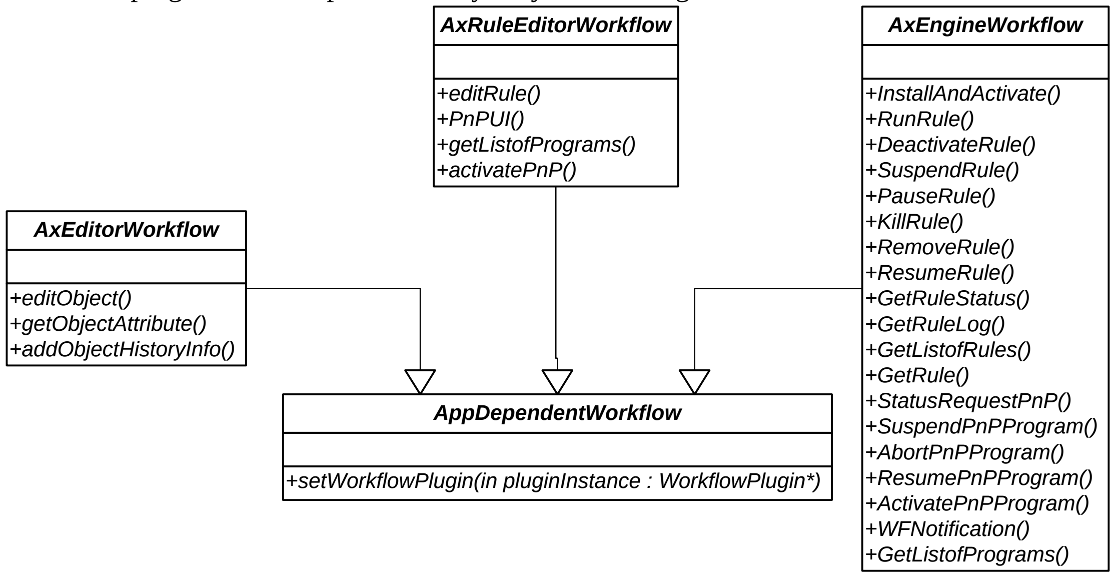

CommandReporting
+WF_CATEGORY : string
-mAppDepWF : AppDependentWorkflow *
-mPluginWF : WorkflowPlugin *
-theInstance : CommandReporting *
+initialize()
+getInstance() : CommandReporting &
+setAppDependentWorkflow(in adwf : AppDependentWorkflow*)
+isWorkflowPresent() : bool
+getWorkflowPlugin() : WorkflowPlugin *
-CommandReporting(in wfPlugin : WorkflowPlugin*)
1
-retrieves
1
-manages
1
1
AppDependentWorkflow
+setWorkflowPlugin(in plugin : WorkflowPlugin)
AXMEDIS Editor example
AxEditorWorkflow
+setWorkflowPlugin()
+editObject()
+getObjectAttribute()
+addObjectHistoryInfo()
«uses»
«uses»
AxEditorWorkflowPlugin
+notifyRequestCompletion()
+getWFInfo()
WorkflowPlugin
+setCommandReporting(in appDepWF : AppDependentWorkflow*)
1
1
-contains
1
AxWFPluginInstance
-FACTORY_METHOD_NAME : string = "getWFPluginInstance"
-FACTORY_RELEASE_NAME : string = "releaseWFPlugin"
-mCreatedPlugin : WorkflowPlugin *
-mFactoryMethod : WFPluginFactoryType
-mReleaseMethod : WFPluginReleaseType
+AxWFPluginInstance(inout workingDir : const string, in profile : const AxPluginProfile*)
+~AxWFPluginInstance()
+getWFPluginObject() : WorkflowPlugin *
+releaseWFPluginObject(in obj : WorkflowPlugin*)
«uses»
PluginManager::AxPluginProfile
PluginManager::AxPluginInstance

In the following diagram the initialization sequence of CommandReporting has been depicted. Please note that the AxPluginManager is invoked in order to have the list of the plugin which are present in the plugin directory.

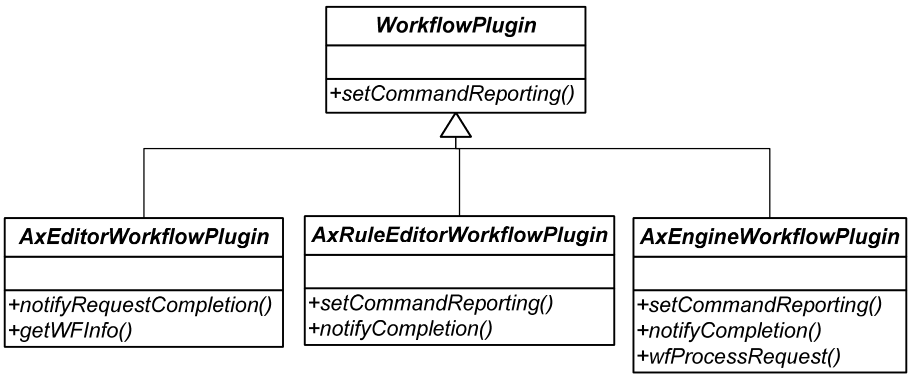


17.2.1 Different application dependent workflows interfaces

For each AXMEDIS Content Production tool a workflow interface has been defined. Each interface has to support the basic nature of AppDependentWorkflow adding to it several possible actions that can be invoked through the corresponding plugin. AXMEDIS Editor implementation has to provide a suitable AxEditorWorkflow inherited class with the desired implementation. On the other side a AXMEDIS Editor Workflow plugin can be implemented by only considering the standard AxEditorWorkflow abstract class.



An abstract class for each different type of plug-in has been defined as well. In this case the Application Workflow can use plug-in functionalities in order to output to workflow notification or other reports.



17.3 Technical and Installation information

References to other major components needed	
Problems not solved	•
Configuration and execution context	

17.4 Draft User Manual

see examples of usage

17.5 Examples of usage

In the following example it has been depicted an example of how to implement an editor initialization of workflow infrastructure via CommandReporting. After initialization the singleton can be obtained and the presence of a workflow plug.in can be checked.
In the last line the specific application workflow implementation is registered to the CommandReporting. After this action the workflow is ready to operate in the running application.

```
 CommandReporting::initialize();
CommandReporting& cr = CommandReporting::getInstance();

 if(!cr.isWorkflowPresent())
     printf("Plugin not present");
AxEditorWorkflowImpl *editorWF = new AxEditorWorkflowImpl();
cr.setAppDependentWorkflow(editorWF);
```

17.6 Integration and compilation issues

none

17.7 Configuration Parameters

Config parameter	**Possible values**

17.8 Errors reported and that may occur

Error code	Description and rationales

18 Module - Internal Audio Player (DSI)

Module/Tool Profile		
Internal Audio Player		
Responsible Name	Bellini	
Responsible Partner	DSI	
Status (proposed/approved)		
Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/axeditor/resource_editor	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/axeditor/bin	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)		
Usage of the AXMEDIS Error Manager (yes/no)		
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section

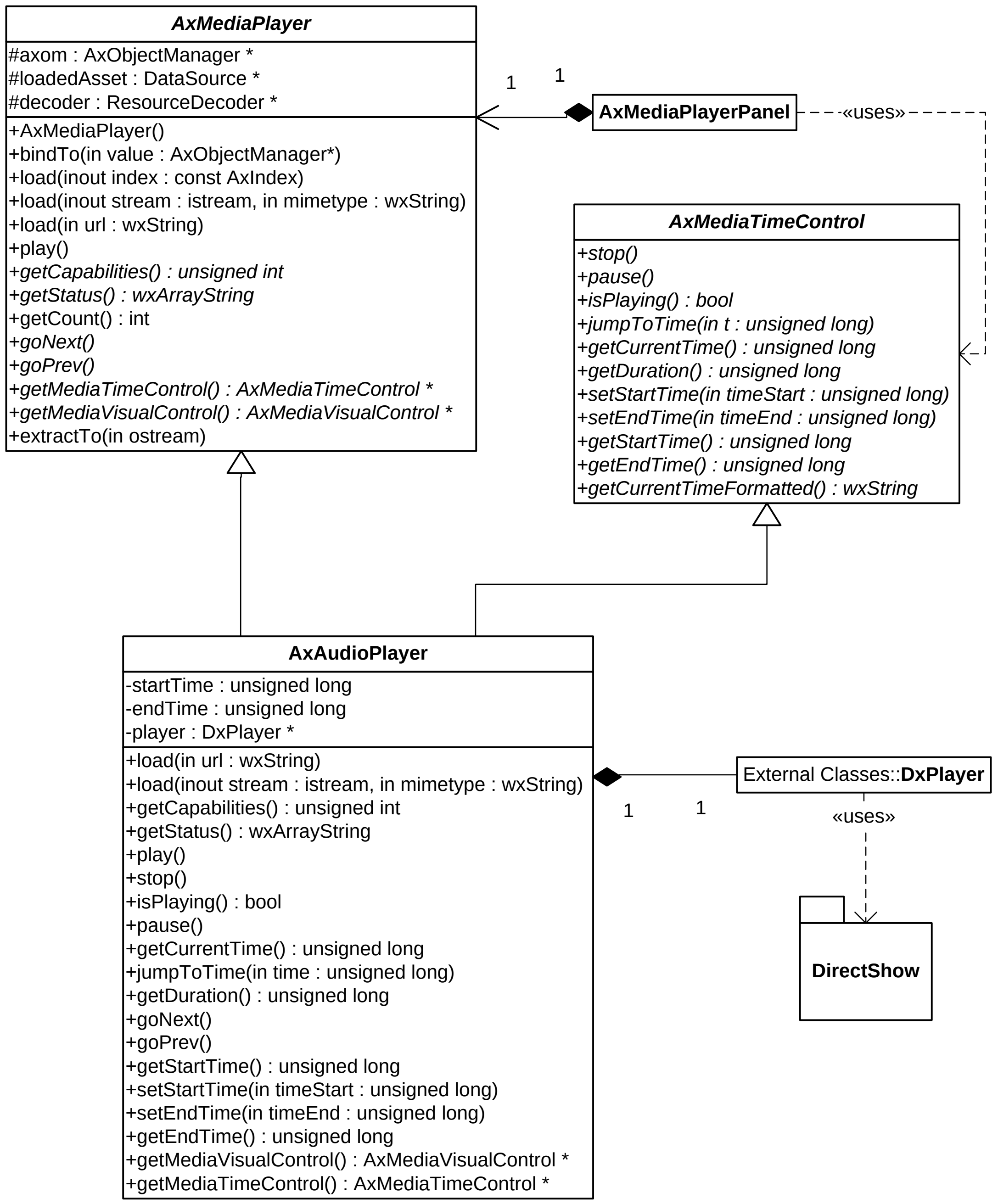
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets	wxWidgets 2.4.2	LGPL
MS- DIRECTX		

18.1 General Description of the Module

The Internal Audio Player will allow to listen to audio files encoded in WAV, MP3 and possibly AAC and WMA formats.
The functionalities supported will allow to:

- load a file directly from file system or from an AXMEDIS Object
- control the playback with play, pause, stop functions
- control the volume
- get status (current time, duration, information on the played resource, …)
- set the start/end time for playing and for extraction
- use content processing plugins

18.2 Module Design in terms of Classes

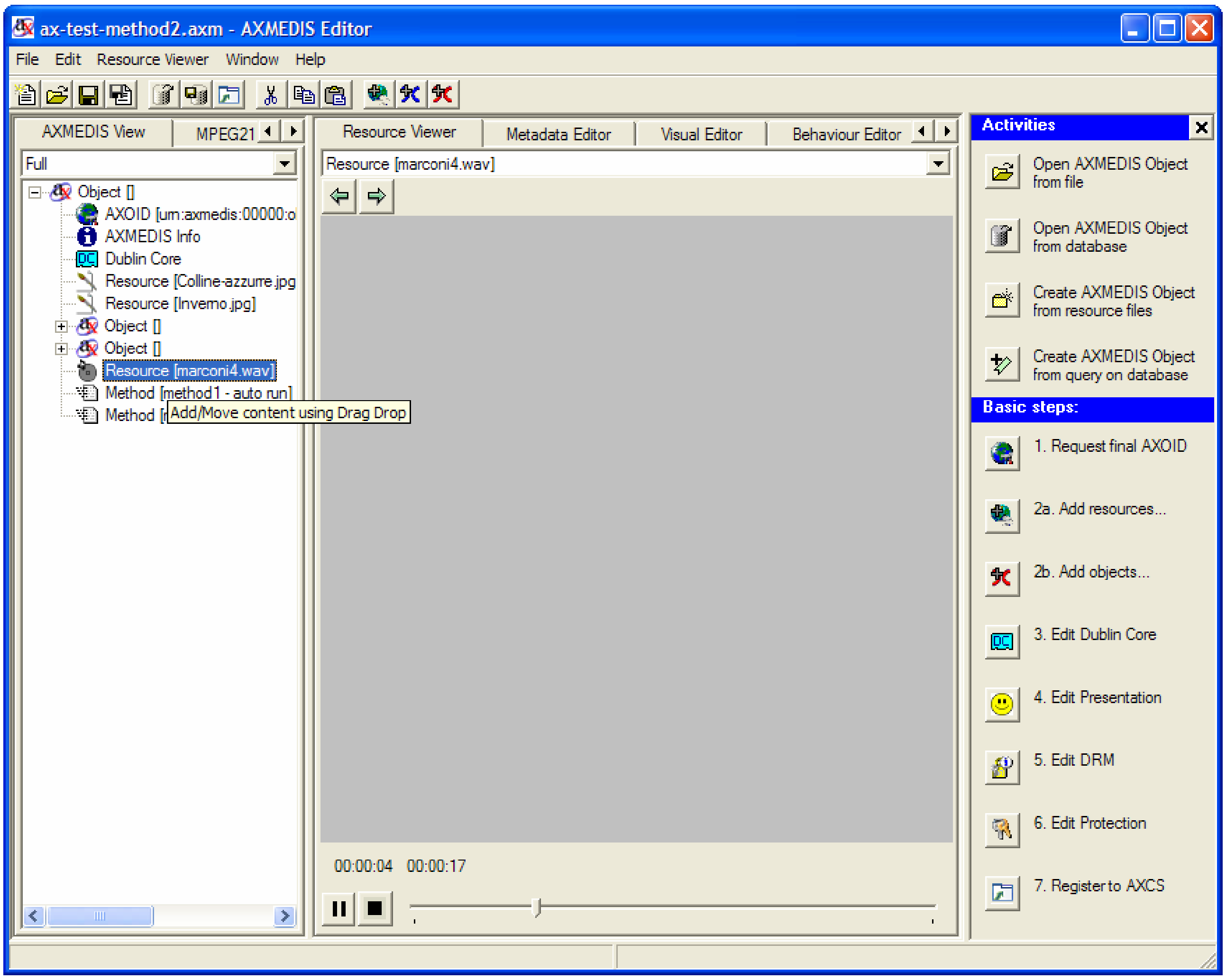


AxMediaPlayer
#axom : AxObjectManager *
#loadedAsset : DataSource *
#decoder : ResourceDecoder *
+AxMediaPlayer()
+bindTo(in value : AxObjectManager*)
+load(inout index : const AxIndex)
+load(inout stream : istream, in mimetype : wxString)
+load(in url : wxString)
+play()
+getCapabilities() : unsigned int
+getStatus() : wxArrayString
+getCount() : int
+goNext()
+goPrev()
+getMediaTimeControl() : AxMediaTimeControl *
+getMediaVisualControl() : AxMediaVisualControl *
+extractTo(in ostream)
1
1
AxMediaPlayerPanel
«uses»
AxMediaTimeControl
+stop()
+pause()
+isPlaying() : bool
+jumpToTime(in t : unsigned long)
+getCurrentTime() : unsigned long
+getDuration() : unsigned long
+setStartTime(in timeStart : unsigned long)
+setEndTime(in timeEnd : unsigned long)
+getStartTime() : unsigned long
+getEndTime() : unsigned long
+getCurrentTimeFormatted() : wxString
AxAudioPlayer
-startTime : unsigned long
-endTime : unsigned long
-player : DxPlayer *
+load(in url : wxString)
+load(inout stream : istream, in mimetype : wxString)
+getCapabilities() : unsigned int
+getStatus() : wxArrayString
+play()
+stop()
+isPlaying() : bool
+pause()
+getCurrentTime() : unsigned long
+jumpToTime(in time : unsigned long)
+getDuration() : unsigned long
+goNext()
+goPrev()
+getStartTime() : unsigned long
+setStartTime(in timeStart : unsigned long)
+setEndTime(in timeEnd : unsigned long)
+getEndTime() : unsigned long
+getMediaVisualControl() : AxMediaVisualControl *
+getMediaTimeControl() : AxMediaTimeControl *
External Classes::DxPlayer
1
1
«uses»
DirectShow

Class **AxMediaPlayerPanel** is a panel containing the basic user interface for a media player allowing the basic operations on media files (play/pause, stop, seek).
Class **AxMediaPlayer** is an abstract class containing the basic functionalities that should be implemented by a media viewer, other functionalities for time control and for visual control are in two specific interfaces (*AxMedisTimeControl*, *AxMediaVisualControl*) that the class derived from *AxMediaPlayer* may on may not implement. Class *AxMediaPlayer* is specialized in **AxAudioPlayer, AxVideoPlayer** and **AxImageViewer.**
Class **AxAudioPlayer** implements the functionalities for audio using DxPlayer that is used from VideoPlayer.

18.3 User interface description

The following is the Internal Audio Player used inside a MediaPlayerPanel as Resource View:



18.4 Technical and Installation information

References to other major components needed	
Problems not solved	•
Configuration and execution context	

18.5 Draft User Manual

See section 19.3 User Interface Description above and for more details, see AXMEDIS User Manual DE5-0-1-2 for AXTools.

18.6 Examples of usage

18.7 Integration and compilation issues

None

18.8 Configuration Parameters

Config parameter	Possible values

18.9 Errors reported and that may occur

Error code	Description and rationales

19 Module - Internal Image Viewer (DSI)

Module/Tool Profile	
Internal Image Viewer	
Responsible Name	Bellini
Responsible Partner	DSI
Status (proposed/approved)	approved
Implemented/not implemented	Implemented
Status of the implementation	90%
Executable or Library/module (Support)	Library
Single Thread or Multithread	Multithread
Language of Development	C++
Platforms supported	Windows/Linux
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/axeditor/resource_editor
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/axeditor/bin
Reference to the AXFW location of the demonstrator executable tool for public download	
Address for accessing to WebServices if any, add accession information (user and Passwd) if any	
Test cases (present/absent)	
Test cases location	
Usage of the AXMEDIS configuration manager	yes

(yes/no)		
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
	C++	wxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets	wxWidgets 2.4.2	LGPL
wxImagick		LGPL
ImageMagick		LGPL

19.1 General Description of the Module

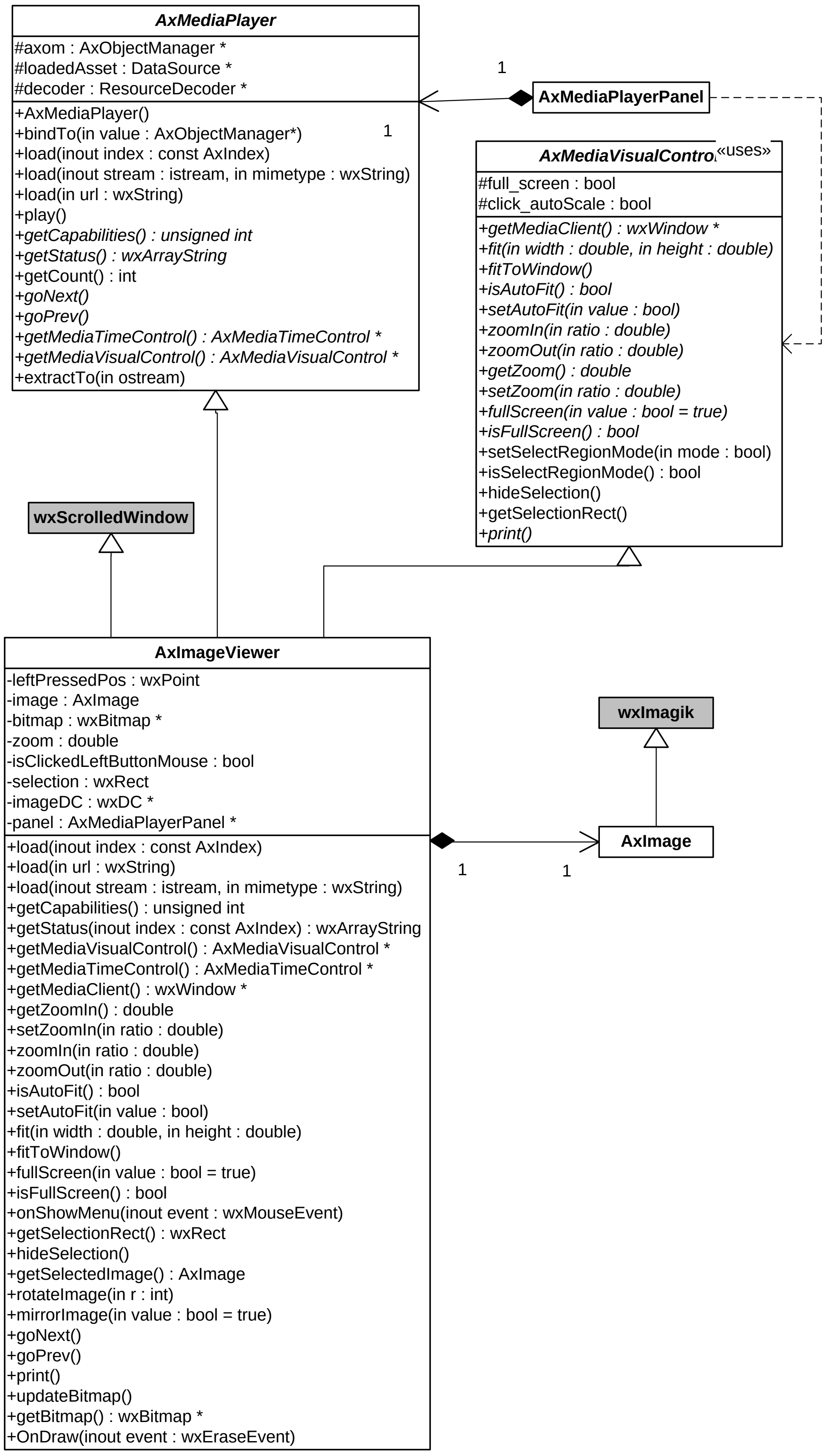
The Internal Image Viewer will allow to display images, it will support the image formats supported by the ImageMagick library.

Functionalities provided will allow to:

- zoom the image

- fit the image within a size
- select a region and save/copy it
- change the image orientation (rotate, flip)
- save the modified image back in the AXMEDIS Object

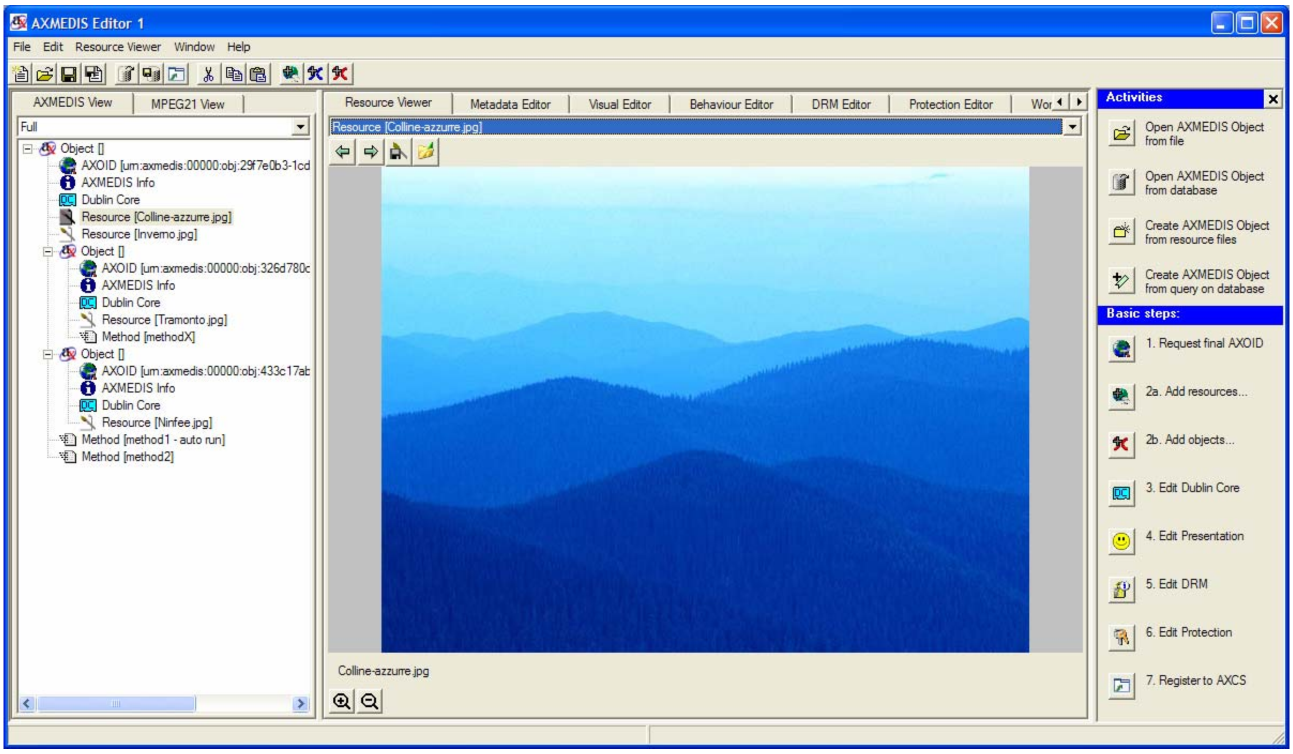
19.2 Module Design in terms of Classes

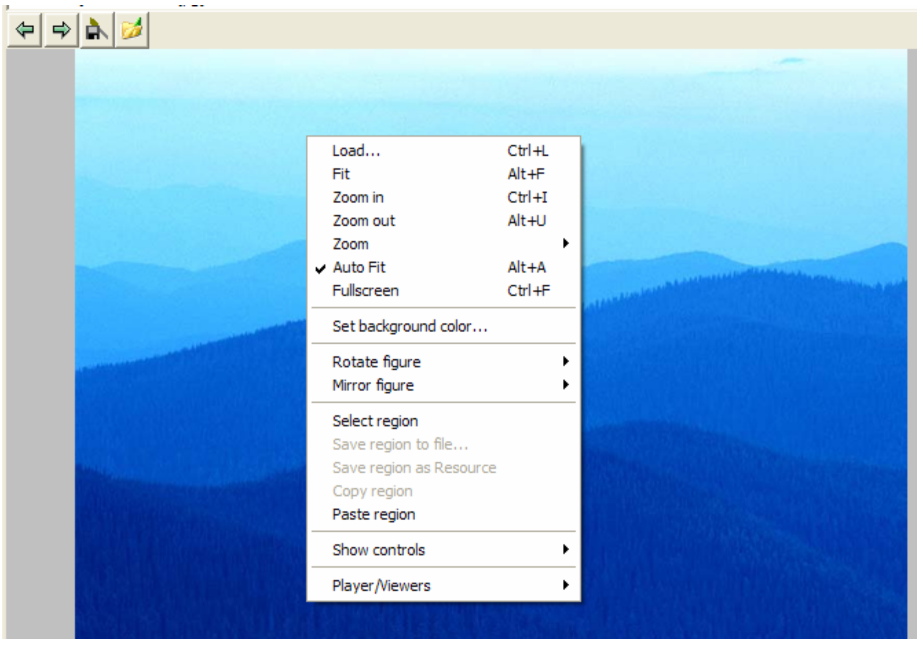

AxMediaPlayer
#axom : AxObjectManager *
#loadedAsset : DataSource *
#decoder : ResourceDecoder *
+AxMediaPlayer()
+bindTo(in value : AxObjectManager*)
+load(inout index : const AxIndex)
+load(inout stream : istream, in mimetype : wxString)
+load(in url : wxString)
+play()
+getCapabilities() : unsigned int
+getStatus() : wxArrayString
+getCount() : int
+goNext()
+goPrev()
+getMediaTimeControl() : AxMediaTimeControl *
+getMediaVisualControl() : AxMediaVisualControl *
+extractTo(in ostream)
1
1
AxMediaPlayerPanel
«uses»
AxMediaVisualControl
#full_screen : bool
#click_autoScale : bool
+getMediaClient() : wxWindow *
+fit(in width : double, in height : double)
+fitToWindow()
+isAutoFit() : bool
+setAutoFit(in value : bool)
+zoomIn(in ratio : double)
+zoomOut(in ratio : double)
+getZoom() : double
+setZoom(in ratio : double)
+fullScreen(in value : bool = true)
+isFullScreen() : bool
+setSelectRegionMode(in mode : bool)
+isSelectRegionMode() : bool
+hideSelection()
+getSelectionRect()
+print()
wxScrolledWindow
AxImageViewer
-leftPressedPos : wxPoint
-image : AxImage
-bitmap : wxBitmap *
-zoom : double
-isClickedLeftButtonMouse : bool
-selection : wxRect
-imageDC : wxDC *
-panel : AxMediaPlayerPanel *
+load(inout index : const AxIndex)
+load(in url : wxString)
+load(inout stream : istream, in mimetype : wxString)
+getCapabilities() : unsigned int
+getStatus(inout index : const AxIndex) : wxArrayString
+getMediaVisualControl() : AxMediaVisualControl *
+getMediaTimeControl() : AxMediaTimeControl *
+getMediaClient() : wxWindow *
+getZoomIn() : double
+setZoomIn(in ratio : double)
+zoomIn(in ratio : double)
+zoomOut(in ratio : double)
+isAutoFit() : bool
+setAutoFit(in value : bool)
+fit(in width : double, in height : double)
+fitToWindow()
+fullScreen(in value : bool = true)
+isFullScreen() : bool
+onShowMenu(inout event : wxMouseEvent)
+getSelectionRect() : wxRect
+hideSelection()
+getSelectedImage() : AxImage
+rotateImage(in r : int)
+mirrorImage(in value : bool = true)
+goNext()
+goPrev()
+print()
+updateBitmap()
+getBitmap() : wxBitmap *
+OnDraw(inout event : wxEraseEvent)
wxImagik
AxImage
1
1

Class ***AxImageViewer*** implements the functionalities of *axMediaPlayer* for viewing an image. It uses the wxImagick library to view the images. wxImagick uses the ImageMagick library for encoding/decoding images (multi platform) but the visualization works only under Windows.

19.3 User interface description

The following is the user interface to view images, the AxMediaPlayerPanel (inside the Resource View Tab) contains the AxImageViewer





19.4 Technical and Installation information

References to other major	

components needed	
Problems not solved	•
Configuration and execution context	

19.5 Draft User Manual

See section 20.3 User Interface Description above and for more details, see AXMEDIS User Manual DE5-0-1-2 for AXTools.

19.6 Integration and compilation issues

None

19.7 Configuration Parameters

Config parameter	Possible values

19.8 Errors reported and that may occur

Error code	Description and rationales

20 Module - Internal Video Player (DSI)

Module/Tool Profile	
Internal Video Player	
Responsible Name	Bellini
Responsible Partner	DSI
Status (proposed/approved)	approved
Implemented/not implemented	Implemented
Status of the implementation	90%
Executable or Library/module (Support)	Library
Single Thread or Multithread	Single thread
Language of Development	C++
Platforms supported	windows
Reference to the AXFW	https://cvs.axmedis.org/newrepos/Framework/source/axeditor/resource_editor

location of the source code demonstrator		
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/axeditor/bin	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	yes	
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
	C++	wxWidgets

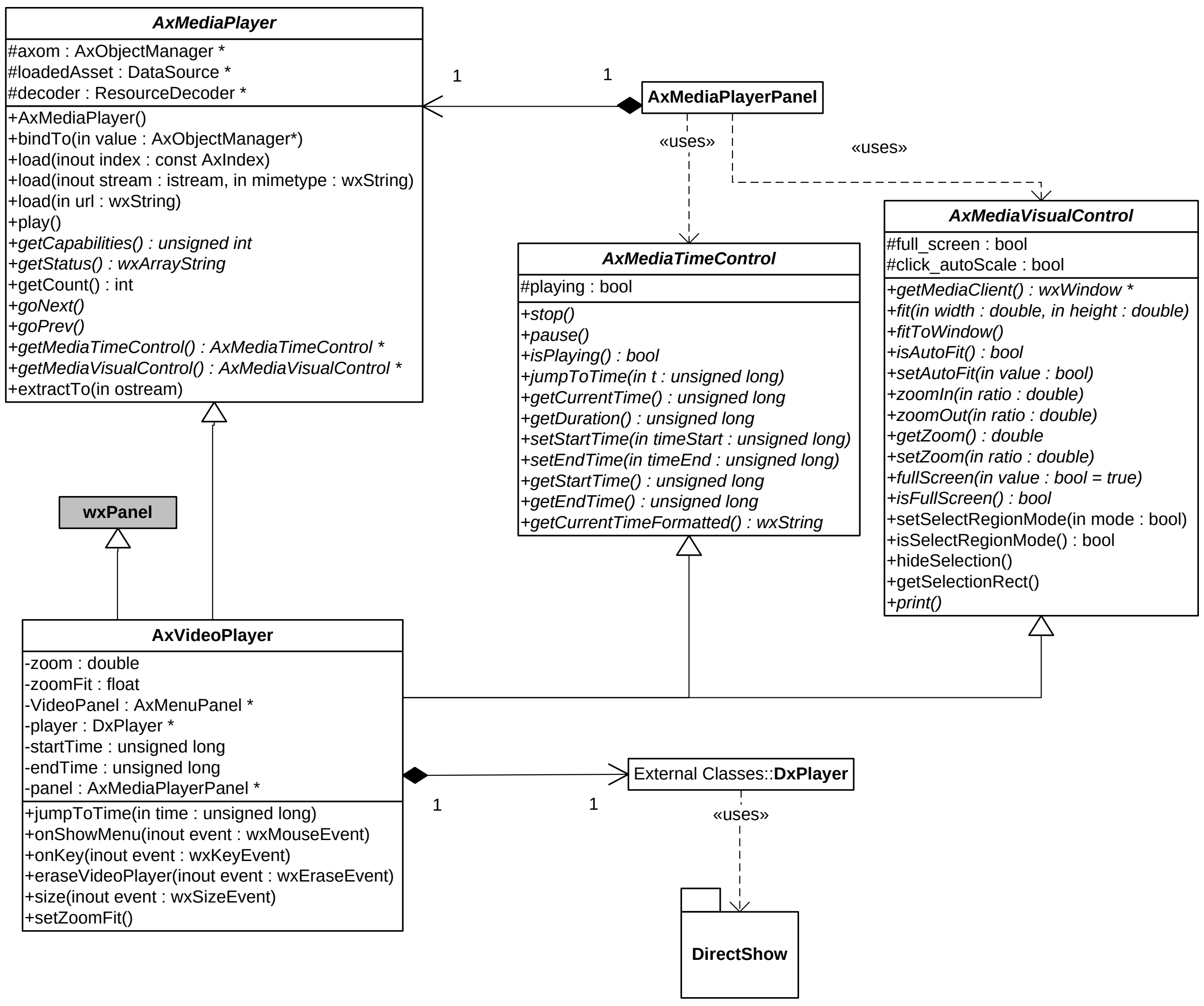
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets	wxWidgets 2.4.2	LGPL
DirectX SDK	DirectX v.9	

20.1 General Description of the Module

The Internal Video Player will be used to display video resources. It will provide functionalities to:

- control execution play/stop/pause
- seek to a time position
- fit the video frame in a dimension,
- zoom in or out
- go in full screen mode

20.2 Module Design in terms of Classes

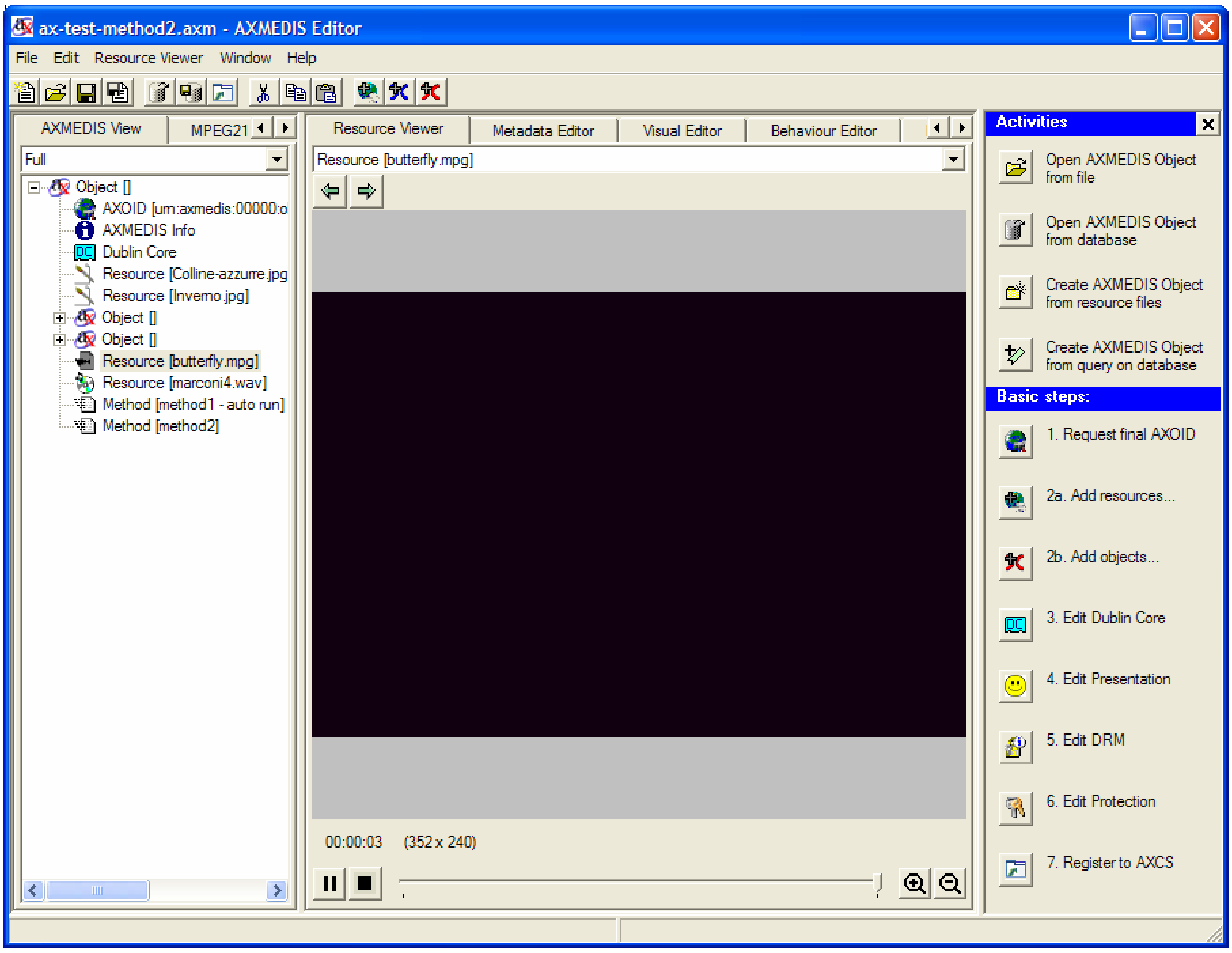

AxMediaPlayer
#axom : AxObjectManager *
#loadedAsset : DataSource *
#decoder : ResourceDecoder *
+AxMediaPlayer()
+bindTo(in value : AxObjectManager*)
+load(inout index : const AxIndex)
+load(inout stream : istream, in mimetype : wxString)
+load(in url : wxString)
+play()
+getCapabilities() : unsigned int
+getStatus() : wxArrayString
+getCount() : int
+goNext()
+goPrev()
+getMediaTimeControl() : AxMediaTimeControl *
+getMediaVisualControl() : AxMediaVisualControl *
+extractTo(in ostream)
1
1
AxMediaPlayerPanel
«uses»
«uses»
AxMediaTimeControl
#playing : bool
+stop()
+pause()
+isPlaying() : bool
+jumpToTime(in t : unsigned long)
+getCurrentTime() : unsigned long
+getDuration() : unsigned long
+setStartTime(in timeStart : unsigned long)
+setEndTime(in timeEnd : unsigned long)
+getStartTime() : unsigned long
+getEndTime() : unsigned long
+getCurrentTimeFormatted() : wxString
AxMediaVisualControl
#full_screen : bool
#click_autoScale : bool
+getMediaClient() : wxWindow *
+fit(in width : double, in height : double)
+fitToWindow()
+isAutoFit() : bool
+setAutoFit(in value : bool)
+zoomIn(in ratio : double)
+zoomOut(in ratio : double)
+getZoom() : double
+setZoom(in ratio : double)
+fullScreen(in value : bool = true)
+isFullScreen() : bool
+setSelectRegionMode(in mode : bool)
+isSelectRegionMode() : bool
+hideSelection()
+getSelectionRect()
+print()
wxPanel
AxVideoPlayer
-zoom : double
-zoomFit : float
-VideoPanel : AxMenuPanel *
-player : DxPlayer *
-startTime : unsigned long
-endTime : unsigned long
-panel : AxMediaPlayerPanel *
+jumpToTime(in time : unsigned long)
+onShowMenu(inout event : wxMouseEvent)
+onKey(inout event : wxKeyEvent)
+eraseVideoPlayer(inout event : wxEraseEvent)
+size(inout event : wxSizeEvent)
+setZoomFit()
1
1
External Classes::DxPlayer
«uses»
DirectShow

Class **AxVideoPlayer** implements the functionalities of **AxMediaPlayer, AxMediaTimeControl** and **AxMediaVisualControl** for playing a video.
Class ***AxVideoPlayer*** has been implemented using **DirectShow** under Windows. For other platforms (Linux/MAC) the use of cross platform library will be investigated (like SDL – Simple DirectMedia Layer, http://www.libsdl.org).
The main issue is on how to access a protected video without writing it in clear as file. In DirectX it can be done writing a custom *AsyncSource* node to be used in the decoding Graph. Have to be noted that this node should not to be deployed as DLL otherwise a malicious user can build a Graph allowing to save in clear the whole video.

Note: The DxPlayer has been enhanced to work with C++ streams, however it work only for MP3 audio files and MPEG video files.

20.3 User interface description

The following may be the user interface of the Internal Video Player:



20.4 Technical and Installation information

References to other major components needed	
Problems not solved	• works only getting data from C++ streams with MP3 and MPEG video files
Configuration and execution context	

20.5 Draft User Manual

See section 21.3 User Interface Description above and for more details, see AXMEDIS User Manual DE5-0-1-2 for AXTools.

20.6 Integration and compilation issues

None

20.7 Configuration Parameters

Config parameter	Possible values

20.8 Errors reported and that may occur

Error code	Description and rationales

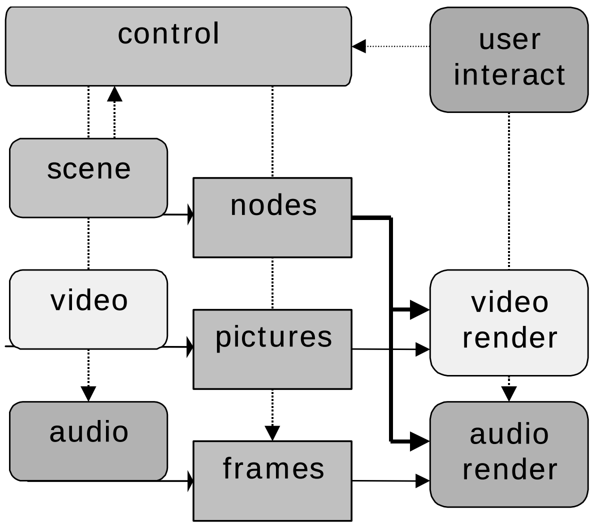
21 Module - Internal MPEG4 Player (EPFL)

Module/Tool Profile		
Internal MPEG4 Player		
Responsible Name	Marco Mattavelli	
Responsible Partner	EPFL	
Status (proposed/approved)	Approved	
Implemented/not implemented	Partially implemented	
Status of the implementation	100%	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/mpeg4player	
Reference to the AXFW location of the demonstrator executable tool for internal download		
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)		
Usage of the AXMEDIS Error Manager (yes/no)		
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
mp4 (MPEG-4 File Format)		

Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
DirectX / DirectSound		MS Visual C++ (6, .NET 2002)
OpenGL		
OpenAL		
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
• faac (L-GPL) • im1_dmif_mp4 • im1_dmif_trif • im1_dmif_remote • im1_dmifclientfilter other players to be negotiated or changed (see below)	 MPEG-4 r.s. MPEG-4 r.s. MPEG-4 r.s. MPEG-4 r.s.	L-GPL ISO ISO ISO ISO
	r.s. = reference software	
MPEG-4 Player		The OSMO player contained into Gpac 0.4.0 has been chosen as MPEG-4 Systems implementation.

21.1 General Description of the Module

The MPEG-4 internal player constitutes a slightly different case of Media Player for AXMEDIS. In fact MPEG-4 itself not only support 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 this is 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):



Management of specific protection rules is also possible in relevant points of the above diagram according to the MPEG IPMPX specification.

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.
The MPEG-4 Player can directly extract the mp-4 resource from AXMEDIS object without need to saving it as a temporary file on the disk.

21.2 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.

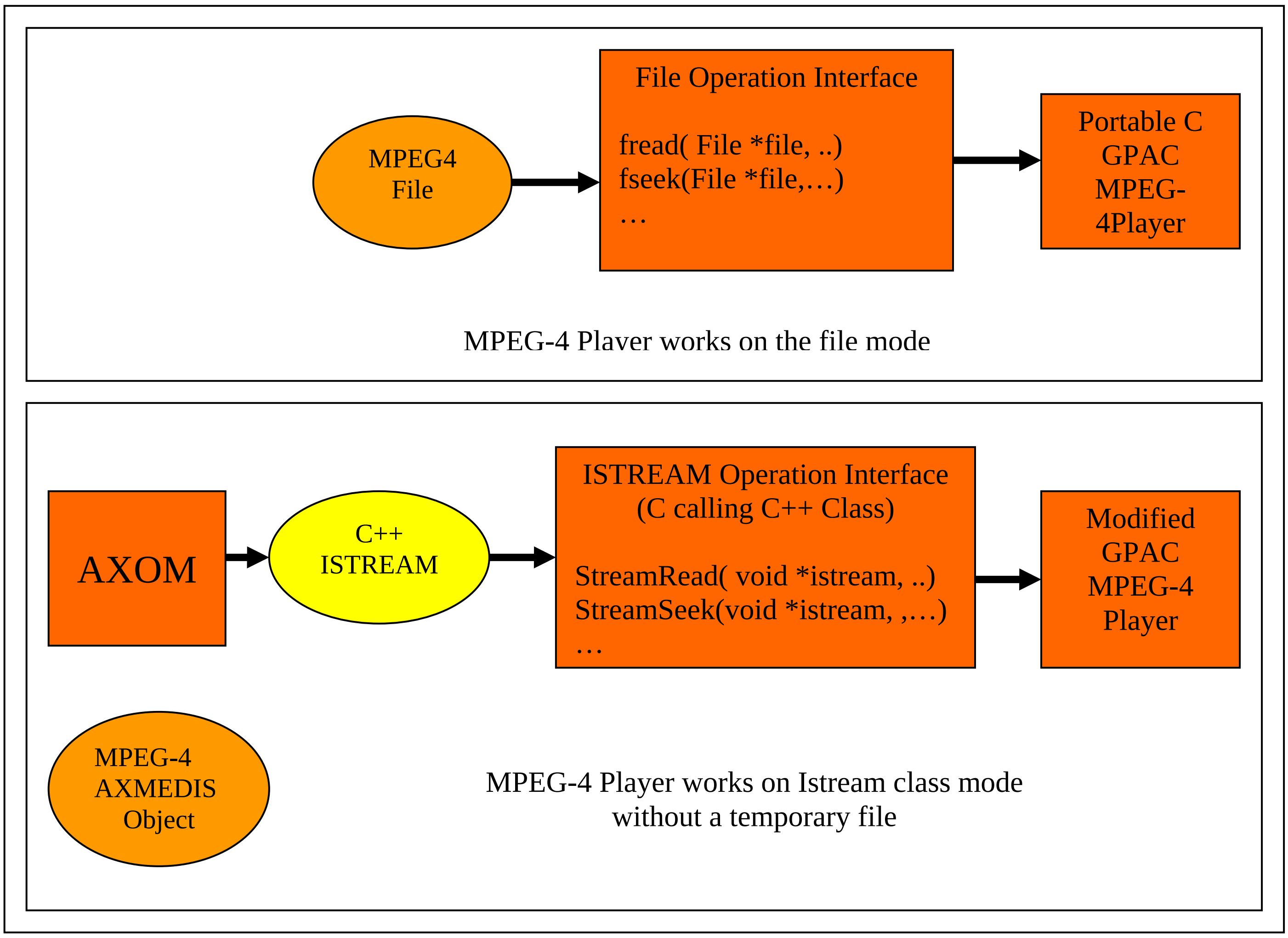

MPEG4 File
File Operation Interface
fread(File *file, ..)
fseek(File *file,…)
…
Portable C GPAC MPEG-4Player
MPEG-4 Player works on the file mode
AXOM
C++ ISTREAM
ISTREAM Operation Interface (C calling C++ Class)
StreamRead(void *istream, ..)
StreamSeek(void *istream, ,…)
…
Modified GPAC MPEG-4 Player
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

22 Module - Internal SMIL Player (EPFL)

Module/Tool Profile		
Internal SMIL Player		
Responsible Name	Beilu Shao, Samuel Keller	
Responsible Partner	EPFL	
Status (proposed/approved)	approved	
Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows,	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Fra	
Reference to the AXFW location of the demonstrator executable tool for internal download		
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	no	
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section

Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
	C++	wxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets	wxWidgets 2.4.2	LGPL
Ambulant	Ambulant 1.6.2	LGPL

22.1 General Description of the Module

SMIL is an XML language for choreographing multimedia presentations where audio, video, text and graphics are combined in real time. The language, the Synchronized Multimedia Integration Language (SMIL, pronounced, "smile") is written as an XML application and is currently a W3C Recommendation. Simply put, it enables authors to specify what should be presented when, enabling them to control the precise time that a sentence is spoken and make it coincide with the display of a given image appearing on the screen.

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 implementation

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.

The bottom line is that we have to modify the datasource interface: to generate media data from AxIndex instead of URL.

It can also display the HTML format resource.

22.2 Module Design in terms of Classes

The AMBULANT player can be used to play SMIL-compatible documents from AXMEDIS objects. A SMIL player has to be able to render different kinds of media objects (text, audio, images, video...). Currently the AMBULANT player delegates the rendering of images, video, or audio to third-party specialized libraries. In case we wanted the SMIL player to be able to use the AXMEDIS internal MPEG-4 player the MPEG-4 player would have to implement the **playable** interface (see UML diagram below). The **playable** interface is used by the SMIL player to control the objects being scheduled (renderers, animations, timelines, transitions. There is a corresponding interface **playable_notification** that implementations of **playable** will use to communicate back: things like media end reached, user clicked the mouse, etc.

The AXMEDIS Editor would control the SMIL player through the **player** interface. The **player** interface – see C++ abstract class below- is the one used by embedding programs that want to control the SMIL player.
In AXMEDIS, all the players have to implement the **axMediaPlayer** interface. The AMBULANT player does not implement this interface but it implements the **player** interface instead. To use the AMBULANT player in AXMEDIS the player interface will have to be extended until it matches the **axMediaPlayer** interface.

Some functions of the **axMediaPlayer** interface and the SMIL **player** interface are the same and will not need to be added. Some other functions of the SMIL Player will have to be slightly modified, for instance the Play function will need, as input parameter, the index of the resource in the AXMEDIS document. Some other functions are missing in the SMIL **play** interface and will need to be added. These are the most important functions to be added to the AMBULANT player to use it, in AXMEDIS, as an internal player:

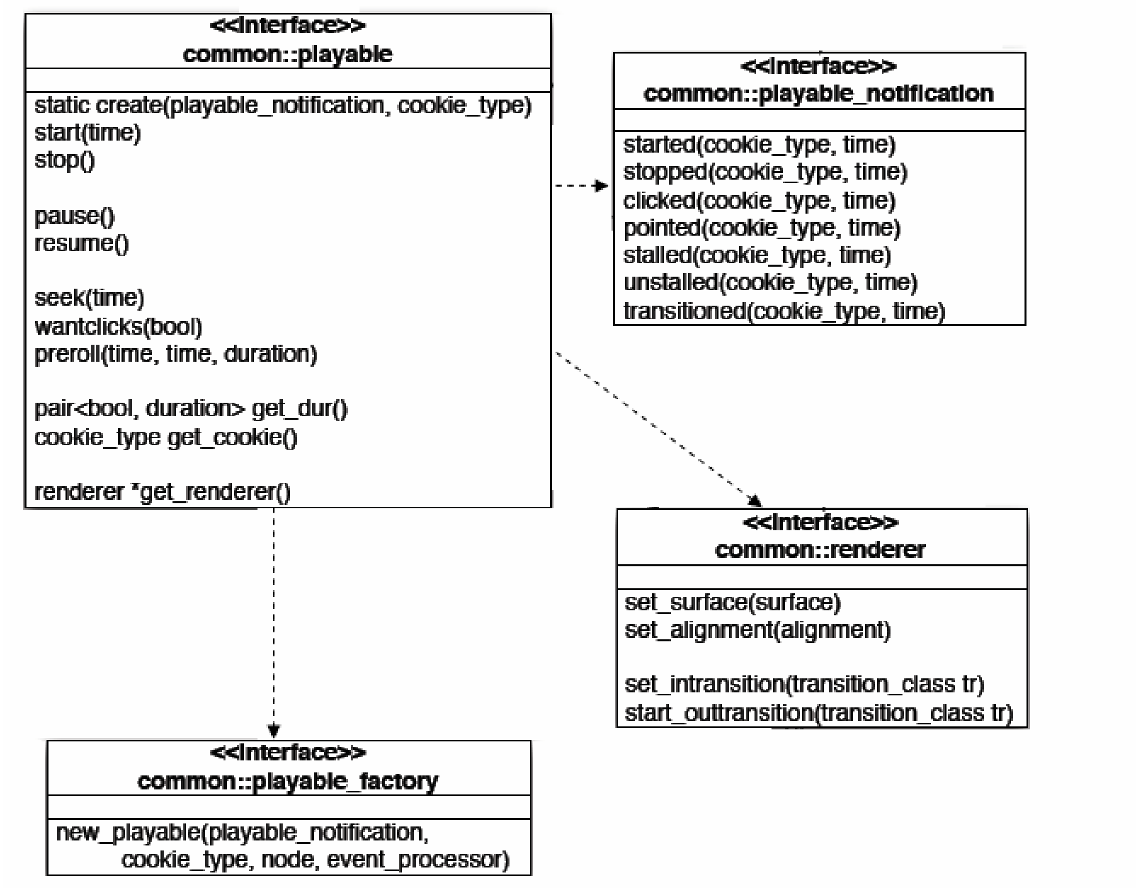
- bindTo(in axom: axObjectManager)
- load(in axoid)
- getMediaClient(): wxWindow

The **bindTo** function will be called on the SMIL player to attach it an AXMEDIS Object Manager. The SMIL player needs a reference to an Object Manager because the player does not have direct access to the AXMEDIS document. The SMIL player will use the reference to the Object Manager to read the AXMEDIS document.
The **load** function will be used to load from the AXMEDIS document a resource with identifier **axoid** -the parameter of the function.
The **getMediaClient** function returns a wxWindow reference. This poses a problem to the AMBULANT player because, in its Windows version, it does not use wxWidgets but MFC. There is no easy conversion from an MFC window object to a wxWidget window object. One solution could be adding another function to the **axMediaPlayer** interface **getMediaClientMM**() which could return a reference to a custom object **axmedisWindow**. The **axmedisWindow** object should implement the set of functions from **wxWindow** that would be needed, for instance:

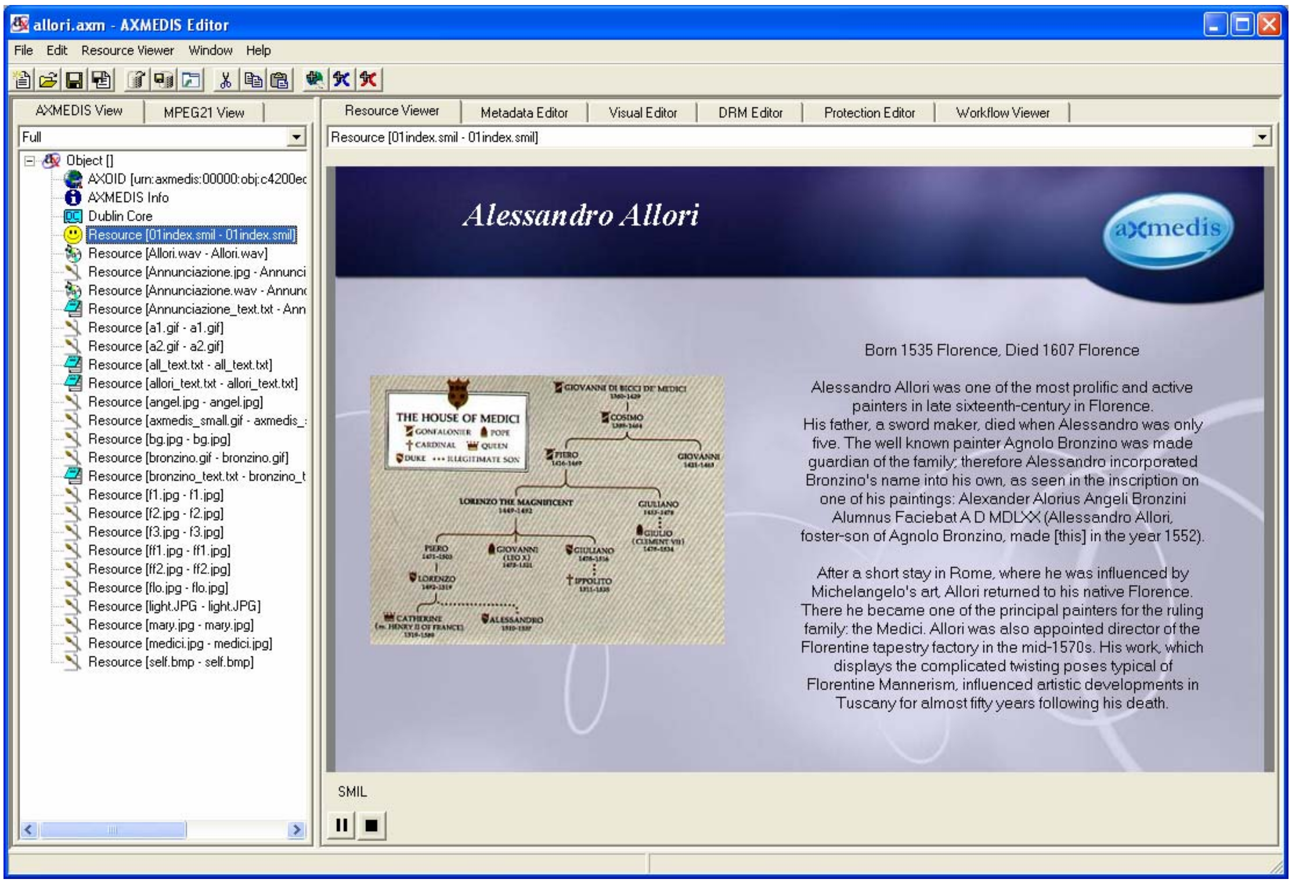
- SetSize(...)
- SetTitle(...)
- Show(...)
- Hide(...)

NB: the same problem occurs in the MPEG-4 Player above, since the management of windows is already implemented either using MS API or OpenGL. Adapting in the proposed way may solve both.

```
/// This is the API an embedding program would use to control the
/// player, to implement things like the "Play" command in the GUI.
class player {
  public:
      virtual ~player() {};

      /// Return the timer this player uses.
      virtual lib::timer* get_timer() = 0;
      /// Return the event_processor this player uses.
      virtual lib::event_processor* get_evp() = 0;
      /// Start playback.
      virtual void start() = 0;
      /// Stop playback.
      virtual void stop() = 0;
      /// Pause playback.
      virtual void pause() = 0;
      /// Undo the effect of pause.
      virtual void resume() = 0;
      /// Return true if player is playing.
      virtual bool is_playing() const { return false;}
      /// Retirn true if player is paused.
      virtual bool is_pausing() const { return false;}
      /// Return true if player has finished.
      virtual bool is_done() const { return false;}
      /// Return index of desired cursor (arrow or hand).
      virtual int get_cursor() const { return 0; }
      /// Set desired cursor.
      virtual void set_cursor(int cursor) {}
//    void set_speed(double speed);
//    double get_speed() const;
};
```


<<Interface>>
common::playable
static create(playable_notification, cookie_type)
start(time)
stop()
pause()
resume()
seek(time)
wantclicks(bool)
preroll(time, time, duration)
pair<bool, duration> get_dur()
cookie_type get_cookie()
renderer *get_renderer()
<<Interface>>
common::playable_notification
started(cookie_type, time)
stopped(cookie_type, time)
clicked(cookie_type, time)
pointed(cookie_type, time)
stalled(cookie_type, time)
unstalled(cookie_type, time)
transitioned(cookie_type, time)
<<Interface>>
common::renderer
set_surface(surface)
set_alignment(alignment)
set_intransition(transition_class tr)
start_outtransition(transition_class tr)
<<Interface>>
common::playable_factory
new_playable(playable_notification,
cookie_type, node, event_processor)

22.3 User interface description



Object []
AXOID [urn:axmedis:00000:obj:c4200ec
AXMEDIS Info
Dublin Core
Resource [01index.smil - 01index.smil]
Resource [Allori.wav - Allori.wav]
Resource [Annunciazione.jpg - Annunci
Resource [Annunciazione.wav - Annunc
Resource [Annunciazione_text.txt - Ann
Resource [a1.gif - a1.gif]
Resource [a2.gif - a2.gif]
Resource [all_text.txt - all_text.txt]
Resource [allori_text.txt - allori_text.txt]
Resource [angel.jpg - angel.jpg]
Resource [axmedis_small.gif - axmedis_:
Resource [bg.jpg - bg.jpg]
Resource [bronzino.gif - bronzino.gif]
Resource [bronzino_text.txt - bronzino_t
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.JPG]
Resource [mary.jpg - mary.jpg]
Resource [medici.jpg - medici.jpg]
Resource [self.bmp - self.bmp]

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.

22.4 Technical and Installation information

References to other major components needed	
Problems not solved	•
Configuration and execution context	

22.5 Draft User Manual

22.6 Examples of usage

22.7 Integration and compilation issues

None

22.8 Configuration Parameters

Config parameter	Possible values

22.9 Errors reported and that may occur

Error code	Description and rationales

23 Module - Internal Document Viewer (SEJER)

Module/Tool Profile		
Internal Document Viewer		
Responsible Name	Jacquet	
Responsible Partner	SEJER	
Status (proposed/approved)	approved	
Implemented/not implemented	Implemented	
Status of the implementation	50%	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Single thread	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/axeditor/resourceeditor	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/axeditor/bin	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	yes	
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section

Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
	C++	wxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets	wxWidgets 2.4.2	LGPL
wxActiveX		LGPL
wxMozilla		

23.1 General Description of the Module

The document viewer will support the visualization of documents like:

- HTML
- PDF
- MSWord Documents
- Postscript

The functionalities provided will be:

- go to next page, go to previous page, go to page
- zoom the page
- fit the page in the view
- print
- scroll the page within the view

To realize these functionalities different approaches can be used for each type of document

23.1.1 HTML

To view HTML files two possibilities are available:

- use the Internet Explorer ActiveX (only under Windows)
- use mozilla embedded inside the application (multi platform)

On one hand, wxActiveX (http://sourceforge.net/projects/wxactivex) is a library providing a wrapper ckass to host ActiveX controls inside wxWidgets applications.
This library also provides an interface to host the Internet Explorer's ActiveX.

The class wxIEHtmlWin can be hosted as a client of any other wx component (like wxFrame, wxNotebook, etc.) it provides functionalities to:

- load from a URL, wxString, a stream object like wxInputStream or std::istream
- set the charset
- get/set the edit mode
- get the selected text inside the IE control (as HTML or not)
- get the text of the page shown in the IE control (as HTML or not)
- go back, go forward, go home
- go search
- refresh the window
- stop the page loading

Moreover it allows catching various events generated by the IE ActiveX like:

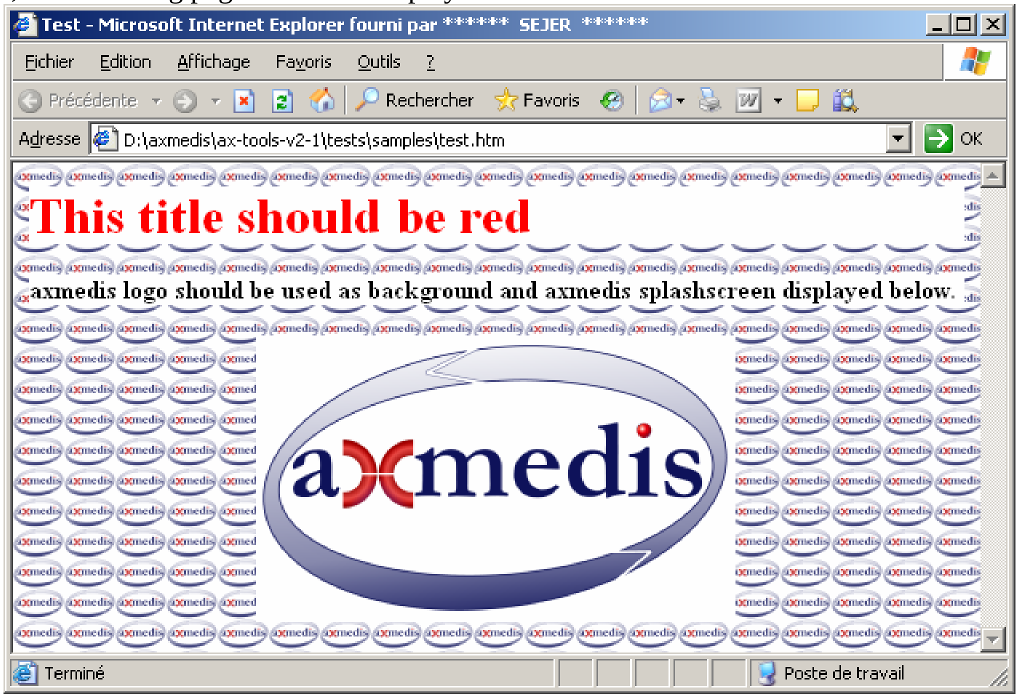
- when status text is changed
- before the connection to an URL
- when the title is changed
- when a new window is opened
- when progress during download changed

To access to protected content the use of streams will avoid saving the HTML files in clear on disk however it is not clear how to provide content referred from the protected html (e.g. images).
Some restrictions could be put to avoid the possibility of copy to clipboard selected text or to view the HTML. To avoid this some filters on events could be set to filter the Ctrl+C and to block the right click in case of protected content.

On the other hand, wxMozilla (http://wxmozilla.sourceforge.net/) is a wrapper of mozilla for wxWidgets applications. It does not use the ActiveX technology thus allowing using it also under other platforms (Linux, MacOS).

In order to control the data displayed not only the HTML but also the images or the style sheets, both browsers use a similar mechanism, called Asynchronous Pluggable Protocol for Internet Explorer and Protocol Handler for Mozilla.

For instance, the following page should be displayed in the same manner in the document viewer:



D:\axmedis\ax-tools-v2-1\tests\samples\test.htm

This title should be red

axmedis logo should be used as background and axmedis splashscreen displayed below.

The splash screen image is declared in a <IMG> tag, the color of the title and the url of the background picture in the CSS file.

The basic principle is to register for a specific URL protocol, such as axmedis://, an object which responsibility is to retrieve data for a certain URL.
Also, AXMEDIS object identifier are URN but we can map them to URL using the following conversion:
urn:axmedis:<yyyyy>:obj:<uuid> => axmedis://<yyyyy>.obj.<uuid>/

For more information on Asynchronous Pluggable Protocols, see http://msdn.microsoft.com/workshop/networking/pluggable/overview/overview.asp and for more information on Protocol Handlers, see http://www.mozilla.org/projects/netlib/new-handler.html.

In terms of configuration, Internet Explorer will be associated by default to the mime-type text/html on PC, while Mozilla will be associated to application/xhtml+xml. On other platforms, Mozilla would associate to both mime-types.

23.1.2 MSWord Documents

To display MSWord Documents the Internet Explorer ActiveX can be used, however protection of such content can be done only by filtering events (like Ctrl+C and right click).

23.1.3 PDF

To display PDF files the following possibilities are available:

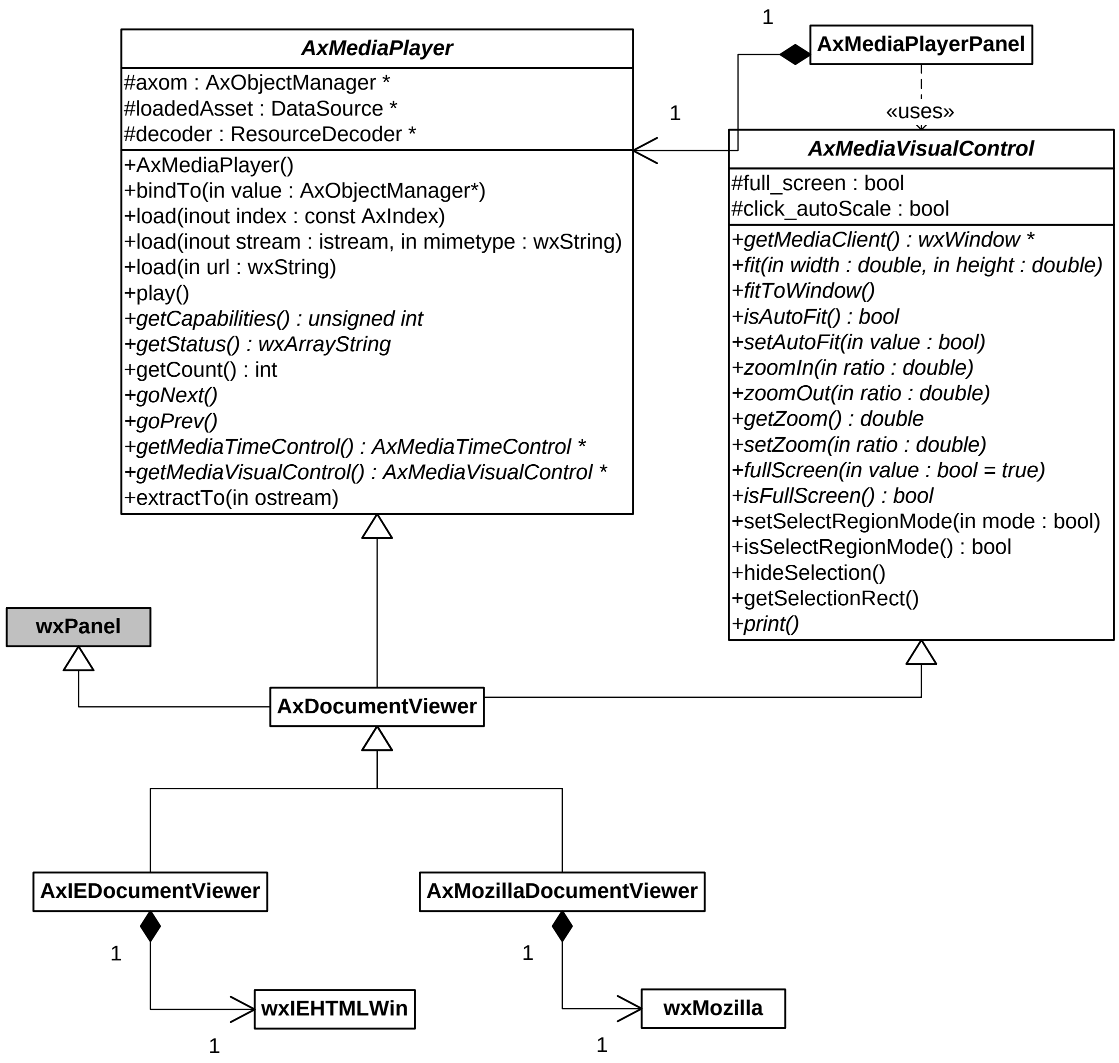
1. use the InternetExplorer ActiveX to display PDF files *(it uses the Acrobat ActiveX)*
2. use directly the Acrobat ActiveX
3. *use the embedded Mozilla to display PDF files (it uses the Acrobat ActiveX)*
4. use ghostscript and *Imagick libraries*

The first three solution use directly on indirectly the Acrobat ActiveX control:
The wxActiveX library could used to host the Acrobat ActiveX inside a wx application.
The interface provided by the Actrobat ActiveX is very simple; basically it allows opening a file from the file system. In case of protected content no way was found to load the PDF file from a stream object avoiding to store the file in clear on the file system. However to protect pdf content the Acrobat DRM can be used and customized for the AXMEDIS needs.
In particular Acrobat supports protection plug-in allowing defining the key to encrypt the PDF file. This key may be generated and stored with the PDF file on the file system. When the PDF is put inside an AXMEDIS object also this key is put inside and protected. When the protected PDF file is opened it is stored on disk (unusable without the key) and the Acrobat ActiveX is used to open it, the AXMEDIS specific protection plug-in for Acrobat will acquire the key from the AXEditor (e.g. via a protected channel) allowing it to display the document. Note that the use of the protection plugin inside AcrobatReader is not free and it has a not negligible cost.
Another solution is to use and adapt some free library for PDF rendering (Xpdf, ghostscript etc.) and to handle protection mechanisms in a more effective way. However these libraries may not support all the features of PDF files.

23.1.4 Postscript

Postscript documents may be visualized using ghostscript and the ImageMagick library (in the same way as PDF files can be), regarding protection aspects the possibility of ghostscript to get content from a stream object rather than from direct file access has to be investigated.

23.2 Module Design in terms of Classes


AxMediaPlayer
#axom : AxObjectManager *
#loadedAsset : DataSource *
#decoder : ResourceDecoder *
+AxMediaPlayer()
+bindTo(in value : AxObjectManager*)
+load(inout index : const AxIndex)
+load(inout stream : istream, in mimetype : wxString)
+load(in url : wxString)
+play()
+getCapabilities() : unsigned int
+getStatus() : wxArrayString
+getCount() : int
+goNext()
+goPrev()
+getMediaTimeControl() : AxMediaTimeControl *
+getMediaVisualControl() : AxMediaVisualControl *
+extractTo(in ostream)
AxMediaPlayerPanel
«uses»
AxMediaVisualControl
#full_screen : bool
#click_autoScale : bool
+getMediaClient() : wxWindow *
+fit(in width : double, in height : double)
+fitToWindow()
+isAutoFit() : bool
+setAutoFit(in value : bool)
+zoomIn(in ratio : double)
+zoomOut(in ratio : double)
+getZoom() : double
+setZoom(in ratio : double)
+fullScreen(in value : bool = true)
+isFullScreen() : bool
+setSelectRegionMode(in mode : bool)
+isSelectRegionMode() : bool
+hideSelection()
+getSelectionRect()
+print()
wxPanel
AxDocumentViewer
AxIEDocumentViewer
AxMozillaDocumentViewer
wxIEHTMLWin
wxMozilla
1
1
1
1
1
1

23.2.1 Protocol handlers

wxMozilla provide a set of classes and functions to ease the implementation and registering of protocol handlers: wxProtocolHandler, wxProtocolChannel.
Mozilla separates the handling of the protocol handler and the retrieval of the data in two classes, while Internet Explorer queries for two interfaces on the registered protocol handler.
Therefore, the logic will be implemented in AxProtocolHandler, which derived from wxProtocolHandler, for all url management aspects, and AxProtocolChannel, which derived from wxProtocolChannel, for actual data retrieval of the AXMEDIS resources.
The AxIEDocumentViewer will implement IInternetProtocolRoot, IInternetProtocol and IInternetProtocolInfo and delegates the call to AxProtocolHandler and AxProtocolChannel.
For internet explorer, as the protocol handler is a COM Object, the AXIEDocumentViewer will implement also a class factory (IClasFactory) in order to be able to instanciate the protocol handler without having to register it in the windows registry.

23.2.2 AxIEDocumentViewer and AxMozillaDocumentViewer

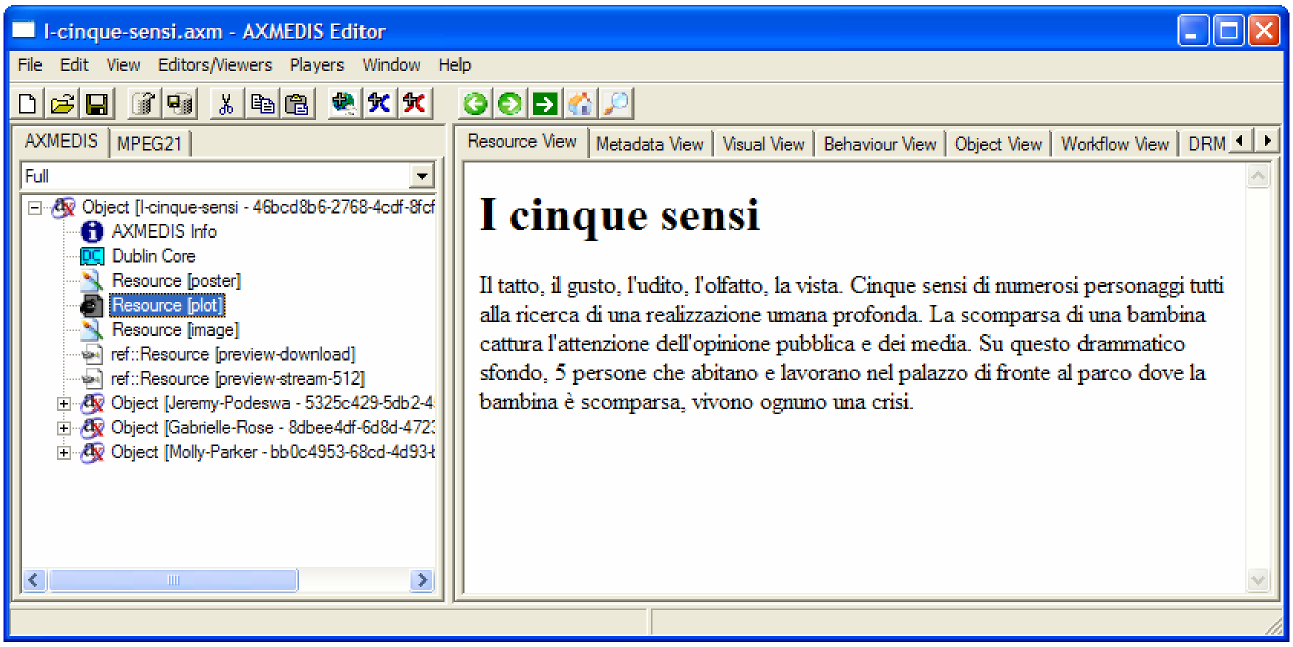
As these two document viewer inherit from AxDocumentViewer, they must implement AxMediaPlayer and AxMediaVisualControl interfaces but some of the function are meaningless for an HTML document:

- play
- getCount
- goNext
- goPrevious

- fit
- zoom
- getSelectionRect

The zoom can be emulated by changing the size of the base font.
Still, some of these functions are needed for other document's type, such as PDF document in which you can go to the next page, the previous page , fit the page to the screen, zoom in and out, etc.

23.3 User interface description


I-cinque-sensi.axm - AXMEDIS Editor
File
Edit
View
Editors/Viewers
Players
Window
Help
AXMEDIS
MPEG21
Full
Object [I-cinque-sensi - 46bcd8b6-2768-4cdf-8fcf
AXMEDIS Info
Dublin Core
Resource [poster]
Resource [plot]
Resource [image]
ref::Resource [preview-download]
ref::Resource [preview-stream-512]
Object [Jeremy-Podeswa - 5325c429-5db2-4
Object [Gabrielle-Rose - 8dbee4df-6d8d-472
Object [Molly-Parker - bb0c4953-68cd-4d93
Resource View
Metadata View
Visual View
Behaviour View
Object View
Workflow View
DRM

I cinque sensi

Il tatto, il gusto, l'udito, l'olfatto, la vista. Cinque sensi di numerosi personaggi tutti alla ricerca di una realizzazione umana profonda. La scomparsa di una bambina cattura l'attenzione dell'opinione pubblica e dei media. Su questo drammatico sfondo, 5 persone che abitano e lavorano nel palazzo di fronte al parco dove la bambina è scomparsa, vivono ognuno una crisi.

23.4 Technical and Installation information

References to other major components needed	
Problems not solved	•
Configuration and execution context	

23.5 Draft User Manual

23.6 Examples of usage

The document viewer can be used to display protected HTML pages, with rich layout and including various media, such pictures. It combines both the ease of creating HTML pages with the protection aspect of AXMEDIS objects.

23.7 Integration and compilation issues

None

23.8 Configuration Parameters

Config parameter	Possible values

23.9 Errors reported and that may occur

Error code	Description and rationales

24 Module - Method Player (DSI)

Module/Tool Profile		
Method Player		
Responsible Name	Bellini	
Responsible Partner	DSI	
Status (proposed/approved)	approved	
Implemented/not implemented	Implemented	
Status of the implementation	10%	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Single thread	
Language of Development	C++	
Platforms supported	windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/axeditor/resource_editor	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/axeditor/bin	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	yes	
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section

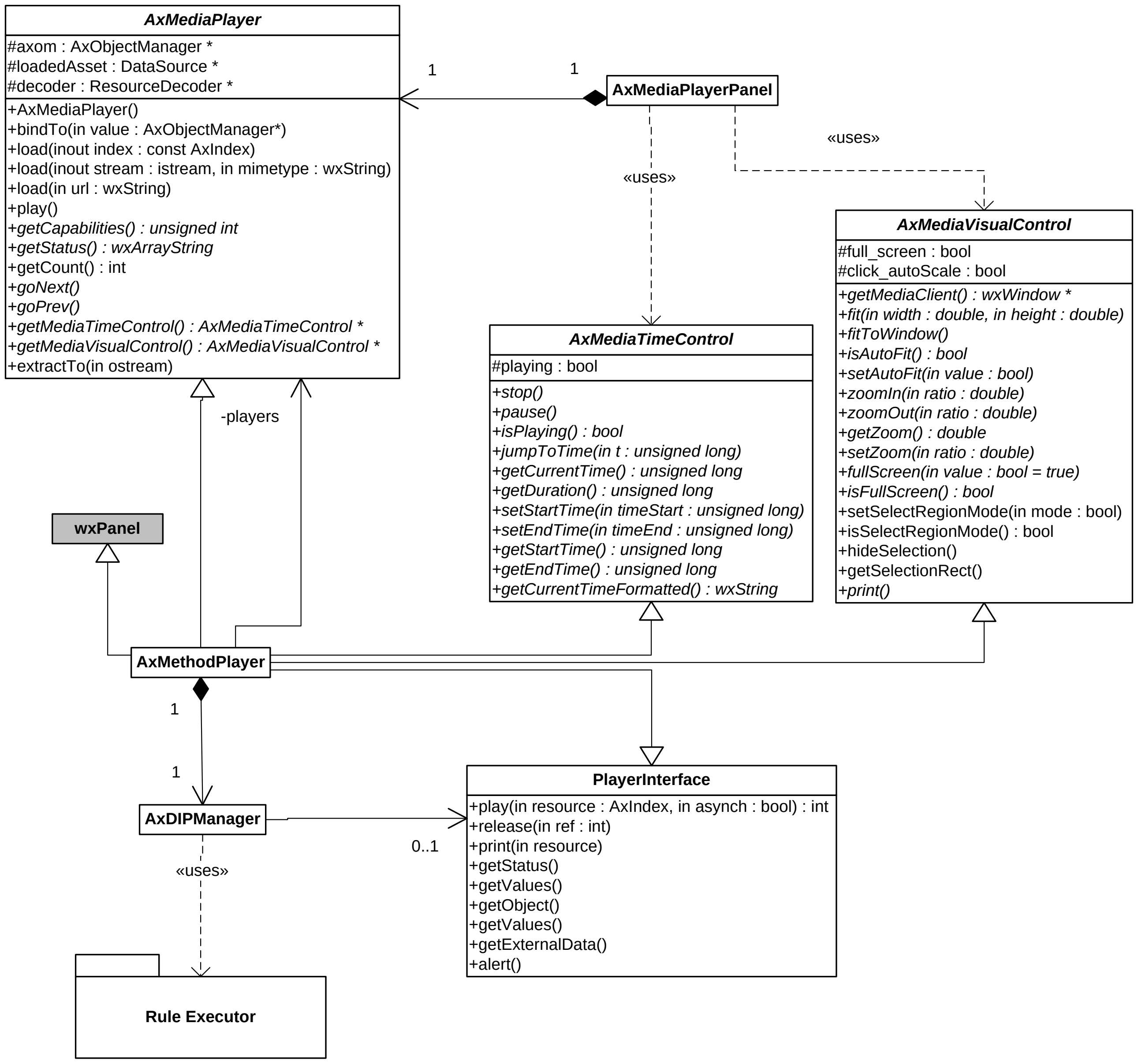
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
	C++	wxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets	wxWidgets 2.4.2	LGPL
spider monkey	Spider Monkey 1.6	LGPL

24.1 General Description of the Module

The Method Player will be used to run MPEG21 DIP methods that can be present inside an AXMEDIS Object. It will provide functionalities to:

- run a method
- implement the basic functionalities of DIP player

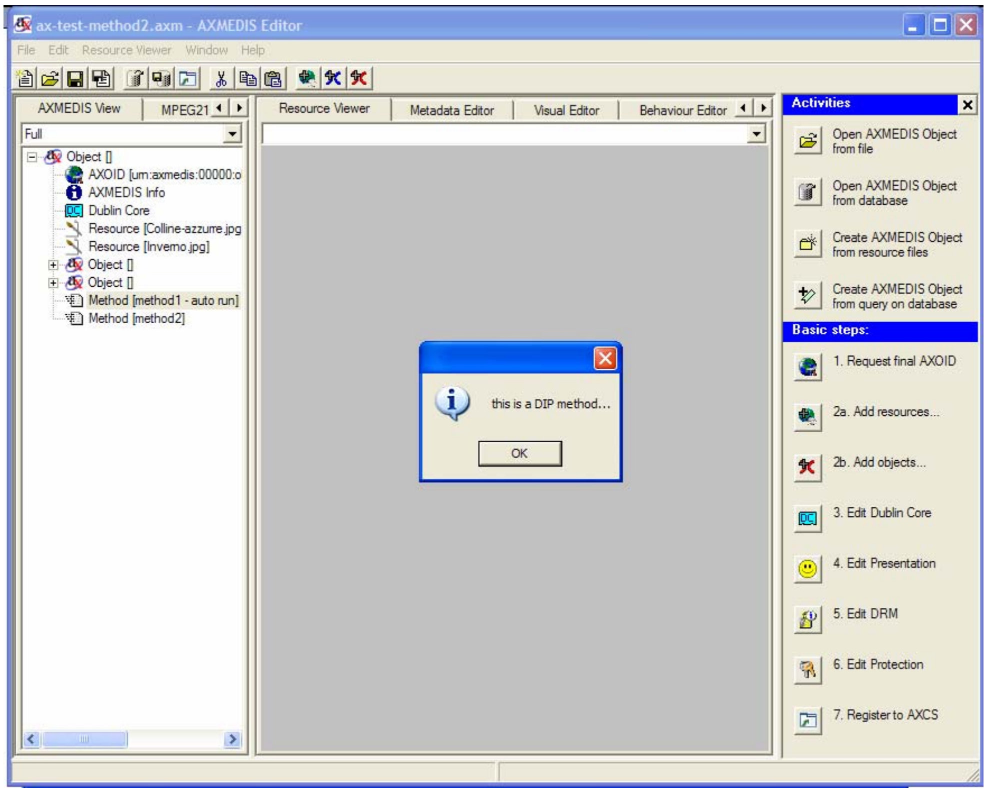
24.2 Module Design in terms of Classes


AxMediaPlayer
#axom : AxObjectManager *
#loadedAsset : DataSource *
#decoder : ResourceDecoder *
+AxMediaPlayer()
+bindTo(in value : AxObjectManager*)
+load(inout index : const AxIndex)
+load(inout stream : istream, in mimetype : wxString)
+load(in url : wxString)
+play()
+getCapabilities() : unsigned int
+getStatus() : wxArrayString
+getCount() : int
+goNext()
+goPrev()
+getMediaTimeControl() : AxMediaTimeControl *
+getMediaVisualControl() : AxMediaVisualControl *
+extractTo(in ostream)
1
1
AxMediaPlayerPanel
«uses»
«uses»
AxMediaVisualControl
#full_screen : bool
#click_autoScale : bool
+getMediaClient() : wxWindow *
+fit(in width : double, in height : double)
+fitToWindow()
+isAutoFit() : bool
+setAutoFit(in value : bool)
+zoomIn(in ratio : double)
+zoomOut(in ratio : double)
+getZoom() : double
+setZoom(in ratio : double)
+fullScreen(in value : bool = true)
+isFullScreen() : bool
+setSelectRegionMode(in mode : bool)
+isSelectRegionMode() : bool
+hideSelection()
+getSelectionRect()
+print()
AxMediaTimeControl
#playing : bool
+stop()
+pause()
+isPlaying() : bool
+jumpToTime(in t : unsigned long)
+getCurrentTime() : unsigned long
+getDuration() : unsigned long
+setStartTime(in timeStart : unsigned long)
+setEndTime(in timeEnd : unsigned long)
+getStartTime() : unsigned long
+getEndTime() : unsigned long
+getCurrentTimeFormatted() : wxString
-players
wxPanel
AxMethodPlayer
1
1
AxDIPManager
0..1
PlayerInterface
+play(in resource : AxIndex, in asynch : bool) : int
+release(in ref : int)
+print(in resource)
+getStatus()
+getValues()
+getObject()
+getValues()
+getExternalData()
+alert()
«uses»
Rule Executor

Class *AxMethodPlayer* implements the functionalities of *AxMediaPlayer*, *AxMediaTimeControl* and *AxMediaVisualControl* for playing a DIP method.
Class *AxMethodPlayer* has been implemented using *AxDIPManager* that is based on the Javascript executor wrapped in AXCP Rule Executor. The *AxMethodPlayer* will implement the *PlayerInterface* interface to support the functionalities needed for the DIP standard.

24.3 User interface description

The following may be the user interface of the Method Player:


this is a DIP method...

24.4 Technical and Installation information

References to other major components needed	
Problems not solved	•
Configuration and execution context	

24.5 Draft User Manual

Not yet available

24.6 Examples of usage

24.7 Integration and compilation issues

None

24.8 Configuration Parameters

Config parameter	Possible values

24.9 Errors reported and that may occur

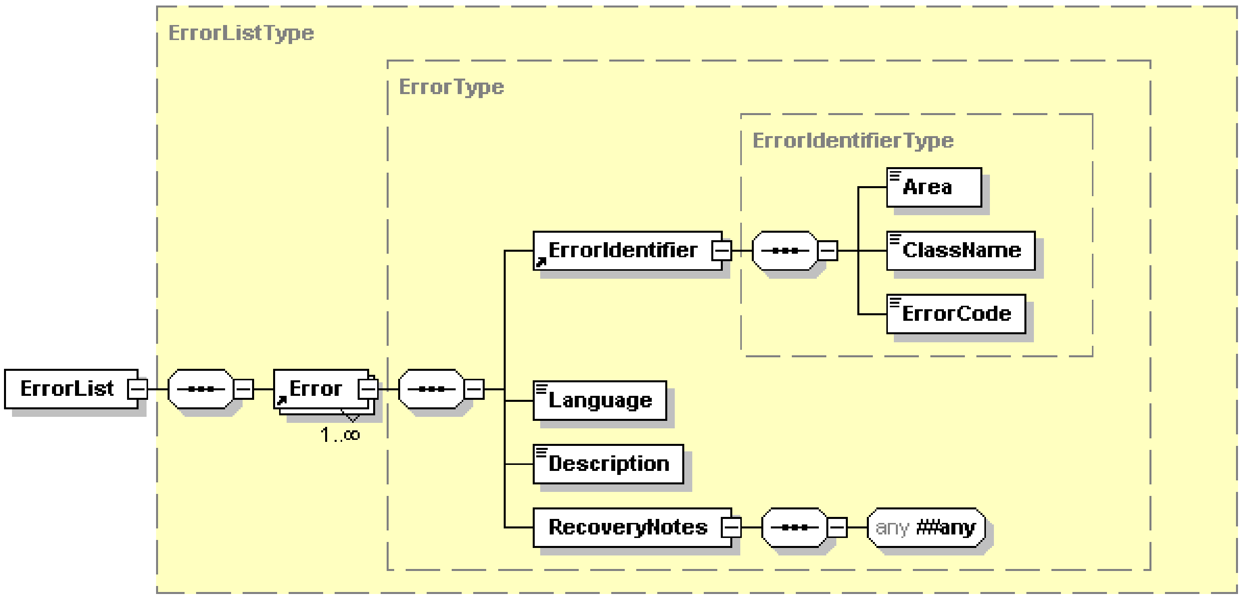
Error code	Description and rationales

25 Formal description of format – Error Coding (DSI)

Each AXMEDIS module can define its own errors. Each errors is defined by the following information:

- **Error Identifier** – area name, full class name, error code
- **Language** – the language used for description and recovery notes
- **Error Description** – the error description in a specific language
- **Recovery notes** – a language dependant description which describe in more details the error and the causes which could have given rise to the error

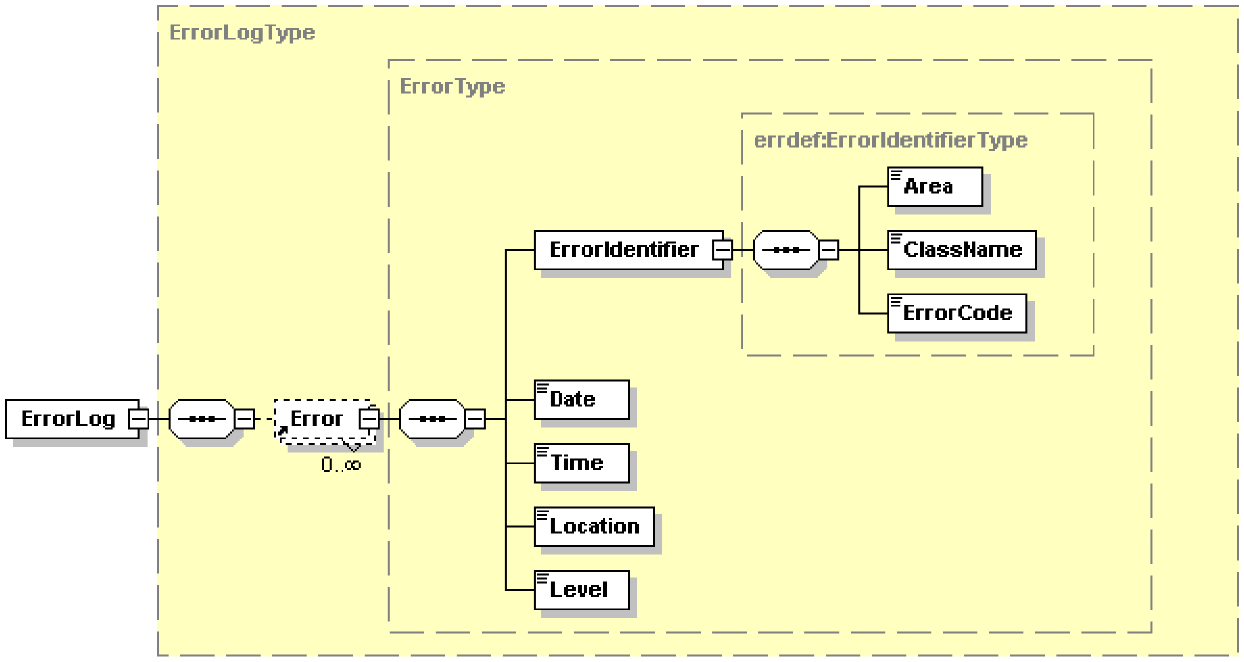
Obviously, the same error (recognized by its identifier) can be defined several time with different languages.
Error definitions are coded in XML file with the following schema:



```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.axmedis.org/error-definition" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.axmedis.org/error-definition" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="ErrorList" type="ErrorListType"/>
    <xs:complexType name="ErrorListType">
        <xs:sequence>
            <xs:element ref="Error" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Error" type="ErrorType"/>
    <xs:complexType name="ErrorType">
        <xs:sequence>
            <xs:element ref="ErrorIdentifier"/>
            <xs:element name="Language" type="xs:language"/>
            <xs:element name="Description" type="xs:string"/>
            <xs:element name="RecoveryNotes">
                <xs:complexType>
                    <xs:sequence>
                        <xs:any namespace="##any"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ErrorIdentifier" type="ErrorIdentifierType"/>
    <xs:complexType name="ErrorIdentifierType">
        <xs:sequence>
            <xs:element name="Area" type="xs:string"/>
            <xs:element name="ClassName" type="xs:string"/>
            <xs:element name="ErrorCode" type="xs:integer"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

26 Formal description of format – Error Log (DSI)

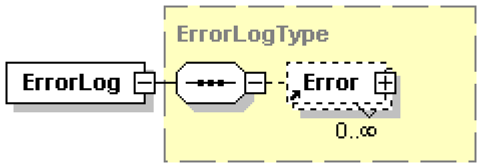
Every time an error is raised through the error manager interface the error is logged somewhere. The location of the log file can be configured using the AXMEDIS Configuration Manager. The log file is in XML format and has the following schema:



```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.axmedis.org/error-log" xmlns="http://www.axmedis.org/error-log"
xmlns:errdef="http://www.axmedis.org/error-definition" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:import namespace="http://www.axmedis.org/error-definition" schemaLocation="error-def.xsd"/>
    <xs:element name="ErrorLog" type="ErrorLogType"/>
    <xs:complexType name="ErrorLogType">
        <xs:sequence>
            <xs:element ref="Error" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Error" type="ErrorType"/>
    <xs:complexType name="ErrorType">
        <xs:sequence>
            <xs:element name="ErrorIdentifier" type="errdef:ErrorIdentifierType"/>
            <xs:element name="Date" type="xs:date"/>
            <xs:element name="Time" type="xs:time"/>
            <xs:element name="Location" type="xs:string"/>
            <xs:element name="Level">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="FATAL"/>
                        <xs:enumeration value="ERROR"/>
                        <xs:enumeration value="WARNING"/>
                        <xs:enumeration value="INFO"/>
                        <xs:enumeration value="DEBUG"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

element **ErrorLog**

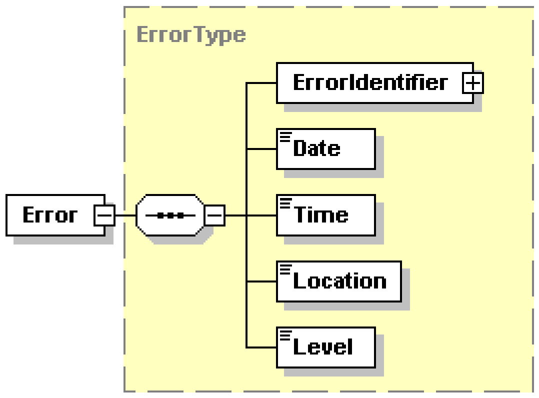
diagram



namespace	http://www.axmedis.org/error-log
type	**ErrorLogType**
children	**Error**
source	`<xs:element name="ErrorLog" type="ErrorLogType"/>`
description	This is the root element of the log file. It can contain none or more **Error**s.

element **Error**

diagram



namespace	http://www.axmedis.org/error-log
type	**ErrorType**
children	**ErrorIdentifier** **Date** **Time** **Location** **Level**
used by	complexType **ErrorLogType**
source	`<xs:element name="Error" type="ErrorType"/>`

description This element represent a logged error. It consist of the following information:

- **ErrorIdentifier** – this is the same element of the error definition schema and it identifies the logged error
- **Date** – the date when the error has been logged
- **Time** – the time when the error has been logged
- **Location** – represent the location where the error has been raised. It consist of an IP address (or something equivalent) and of a process/thread identifier
- **Level** – the level of the error. It can assume the following values FATAL, ERROR, WARNING, INFO or DEBUG on the base of the severity of the error

27 Formal description of format – Configuration (DSI)

The configurations managed by the previous described classes are stored in a specific format. The best way to store that information is to use XML file. In the following, the choosen schema of the XML file is reported and described.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://www.axmedis.org/configuration" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.axmedis.org/configuration configuration.xsd">
    <Module id="ID000000" category="category1/subcategory1" visible="true">
        <Parameter name="parameterLongLong" type="int32">1977</Parameter>
        <Parameter name="parameterString" type="string">Andrea Vallotti</Parameter>
        <Parameter name="parameterDouble" type="double">108.5</Parameter>
    </Module>
    <Module id="ID000001" category="category2/subcategory1/subsubcat3" visible="false">
        <Parameter name="name" type="string">Davide</Parameter>
        <Parameter name="surname" type="string">Rogai</Parameter>
        <Parameter name="age" type="int32">29</Parameter>
    </Module>
</Configuration>
```

In the previous example the modules and parameters are expressed in the needed format. The related XML schema is reported below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.axmedis.org/configuration" xmlns="http://www.axmedis.org/configuration"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="Configuration" type="ConfigurationType"/>
    <xs:complexType name="ConfigurationType">
        <xs:sequence>
            <xs:element ref="Module" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Module" type="ModuleType"/>
    <xs:complexType name="ModuleType">
        <xs:sequence>
            <xs:element ref="Parameter" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:ID" use="required"/>
        <xs:attribute name="category" type="xs:string" use="optional"/>
        <xs:attribute name="visible" type="xs:boolean" use="optional" default="true"/>
    </xs:complexType>
    <xs:element name="Parameter" type="ParameterType"/>
    <xs:complexType name="ParameterType" mixed="false">
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="name" type="xs:string" use="required"/>
                <xs:attribute name="type" use="optional" default="string">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="string"/>
                            <xs:enumeration value="int32"/>
                            <xs:enumeration value="double"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:schema>
```

28 Formal description of format – Plug-ins description (DSI)

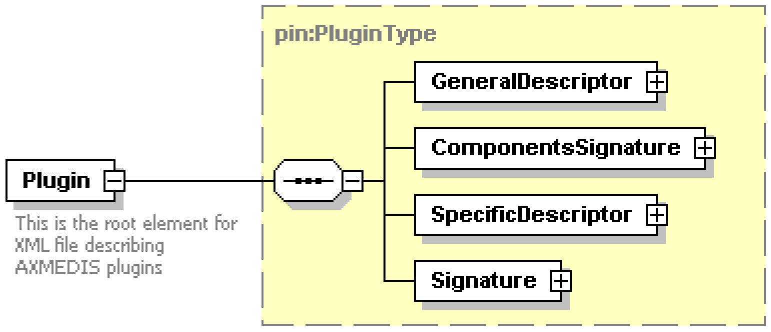
Plug-in profiles have to respect the following XML schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.axmedis.org/plugin-schema" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" xmlns:pin="http://www.axmedis.org/plugin-schema"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="0.1">
    <xs:import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="xmldsig-core-schema.xsd"/>
    <xs:element name="Plugin" type="pin:PluginType">
        <xs:annotation>
            <xs:documentation>This is the root element for XML file describing AXMEDIS plugins</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="PluginType">
        <xs:sequence>
            <xs:element ref="pin:GeneralDescriptor"/>
            <xs:element ref="pin:ComponentsSignature"/>
            <xs:element ref="pin:SpecificDescriptor"/>
            <xs:element ref="pin:Signature"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="GeneralDescriptor" type="pin:GeneralDescriptorType"/>
    <xs:complexType name="GeneralDescriptorType">
        <xs:sequence>
            <xs:element name="Category" type="xs:string"/>
            <xs:element name="Identifier" type="xs:anyURI"/>
            <xs:element name="Library" type="xs:string"/>
            <xs:element name="Version" type="xs:string"/>
            <xs:element name="Vendor" type="xs:string"/>
            <xs:element name="MainLibrary" type="xs:anyURI"/>
            <xs:element name="Description" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ComponentsSignature" type="dsig:SignatureType"/>
    <xs:element name="SpecificDescriptor" type="pin:SpecificDescriptorType" abstract="true"/>
    <xs:complexType name="SpecificDescriptorType" abstract="true"/>
    <xs:element name="Signature" type="dsig:SignatureType"/>
</xs:schema>
```

In the following the semantic and description of each XML element is reported.

element **Plugin**

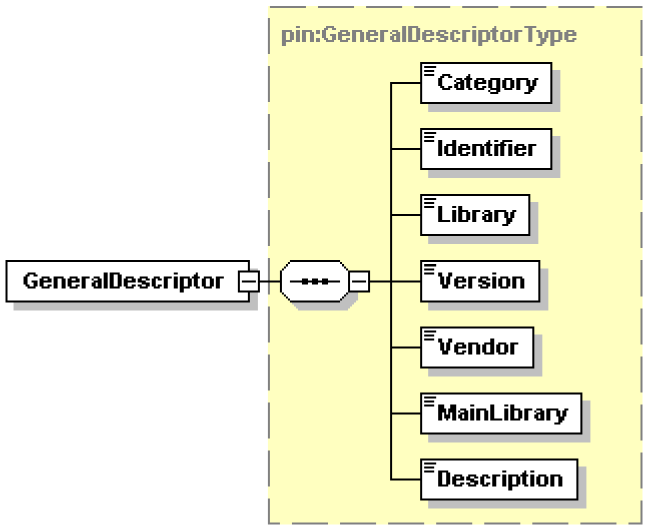
diagram



namespace	http://www.axmedis.org/plugin-schema
type	**pin:PluginType**
children	**GeneralDescriptor ComponentsSignature SpecificDescriptor Signature**
description	This element is the root element of profiles for AXMEDIS plug-ins. It contains necessary information to manage and to use the associated plug-in.

element **GeneralDescriptor**

diagram

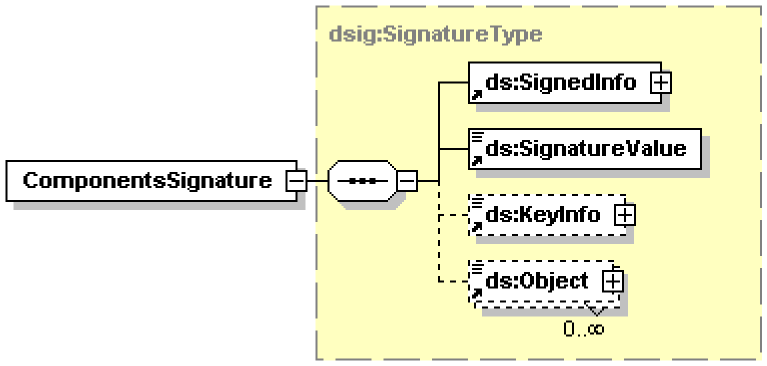


namespace	http://www.axmedis.org/plugin-schema
type	**pin:GeneralDescriptorType**
children	**Category Identifier Library Version Vendor MainLibrary Description**
description	This element contain general information on the plug-in the profile refers to. That information is mandatory and valid for all kind of plug-in. The fields are:

- **Category** represents the type of functionalities the plug-in implements, e.g. content processing, protection tool, etc…
- **Identifier** is the unique identifier of the plug-in in AXMEDIS framework
- **Library** the name of the specific library of the vendor
- **Version** is string representing the version of the software, it could be use for compatibility controls
- **Vendor** is the name/description of the plug-in maker
- **MainLibrary** is a relative URI referencing the dynamic library exposing the interface described in the **SpecificDescriptor** element of the plug-in profile.
- **Descriptor** a human readable description text of the plug-in

element **ComponentsSignature**

diagram



namespace	http://www.axmedis.org/plugin-schema
type	**dsig:SignatureType**
children	**dsig:SignedInfo dsig:SignatureValue dsig:KeyInfo dsig:Object**

attributes	Name	Type	Use	Default	Fixed
	Id	ID	optional		

description This element is a **dsign:SignatureType**. It is the signature (estimated by an AXCS) of the entire plug-in. The signature comprises all relevant resources which compose the plug-in. Those resources are listed as **Reference** elements of **ds:SignedInfo**.

element **SpecificDescriptor**

diagram

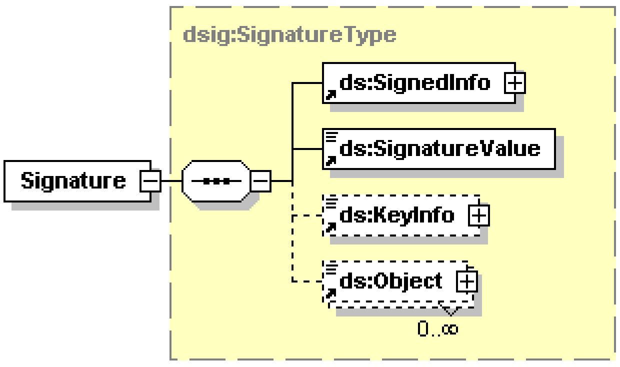
namespace http://www.axmedis.org/plugin-schema

type **pin:SpecificDescriptorType**

description This is an abstract element which can be substituted with any element derived by **SpecificDescriptorType**. In that way, **Category**-dependant XML schema can be used to describe specific features of the plug-in (see **GeneralDescriptor**).

element **Signature**

diagram



namespace http://www.axmedis.org/plugin-schema

type dsig:SignatureType

children dsig:SignedInfo dsig:SignatureValue dsig:KeyInfo dsig:Object

attributes	Name	Type	Use	Default	Fixed
	Id	ID	optional		

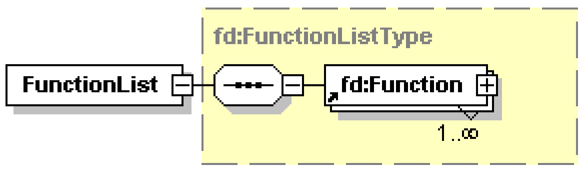
description This is the signature of the whole profile except the Signature element itself. It has been introduced to guarantee the dependability of the data contained in the manifest.

29 Formal description of format – Content Processing Plug-ins specific description (DSI)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.axmedis.org/plugin-function-schema"
xmlns:prm="http://www.axmedis.org/parameter" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:fd="http://www.axmedis.org/plugin-function-schema" xmlns:pin="http://www.axmedis.org/plugin-schema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:import namespace="http://www.axmedis.org/plugin-schema" schemaLocation="plugin-schema.xsd"/>
    <xs:import namespace="http://www.axmedis.org/parameter" schemaLocation="param-schema.xsd"/>
    <xs:element name="FunctionList" type="fd:FunctionListType" substitutionGroup="pin:SpecificDescriptor"/>
    <xs:complexType name="FunctionListType">
        <xs:complexContent>
            <xs:extension base="pin:SpecificDescriptorType">
                <xs:sequence>
                    <xs:element ref="fd:Function" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:element name="Function" type="fd:FunctionType"/>
    <xs:complexType name="FunctionType">
        <xs:sequence>
            <xs:element name="Name" type="xs:ID"/>
            <xs:element name="Version" type="xs:string" minOccurs="0"/>
            <xs:element ref="fd:FunctionDescription"/>
            <xs:element ref="prm:ParameterList" minOccurs="0"/>
            <xs:element ref="fd:Result"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="FunctionDescription" type="xs:string"/>
    <xs:complexType name="DescriptionType">
        <xs:sequence>
            <xs:any namespace="##any"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Result" type="fd:ResultType"/>
    <xs:complexType name="ResultType">
        <xs:sequence>
            <xs:element name="Name" type="xs:ID" minOccurs="0"/>
            <xs:element ref="fd:ResultType"/>
            <xs:element ref="fd:ResultDescription" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ResultDescription" type="xs:string"/>
    <xs:element name="ResultType" type="fd:ResultTypeType"/>
    <xs:simpleType name="ResultTypeType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="UINT16"/>
            <xs:enumeration value="INT16"/>
            <xs:enumeration value="UINT32"/>
            <xs:enumeration value="INT32"/>
            <xs:enumeration value="FLOAT"/>
            <xs:enumeration value="DOUBLE"/>
            <xs:enumeration value="BOOLEAN"/>
            <xs:enumeration value="STRING"/>
            <xs:enumeration value="WSTRING"/>
            <xs:enumeration value="CHAR"/>
            <xs:enumeration value="RESOURCE"/>
            <xs:enumeration value="AXOM"/>
            <xs:enumeration value="AREA"/>
            <xs:enumeration value="VOID"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="Range" type="fd:RangeType" substitutionGroup="prm:Constraint"/>
    <xs:complexType name="RangeType">
        <xs:complexContent>
            <xs:restriction base="prm:ConstraintType">
                <xs:sequence>
```

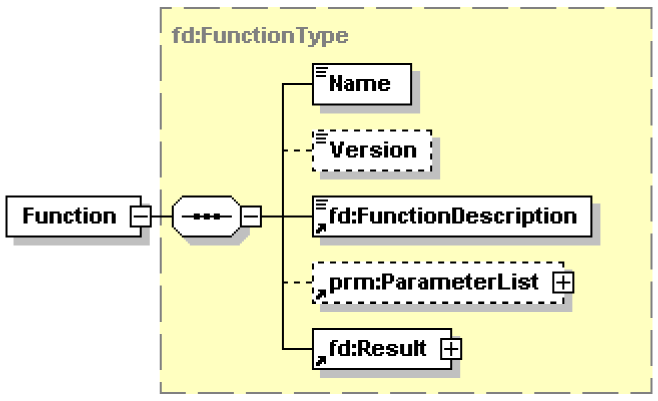
```
                    <xs:element name="From" type="fd:Limit"/>
                    <xs:element name="To" type="fd:Limit"/>
                </xs:sequence>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="Limit">
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="included" type="xs:boolean" use="optional"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:element name="Resource" type="fd:ResourceType" substitutionGroup="prm:Constraint"/>
    <xs:complexType name="ResourceType">
        <xs:complexContent>
            <xs:restriction base="prm:ConstraintType">
                <xs:sequence>
                    <xs:element name="ResourceType" type="xs:string"/>
                    <xs:element name="ResourceFormat" type="xs:string" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

element **FunctionList**

diagram 

namespace	http://www.axmedis.org/plugin-function-schema
type	**fd:FunctionListType**
children	**fd:Function**
source	`<xs:element name="FunctionList" type="fd:FunctionListType" substitutionGroup="pin:SpecificDescriptor"/>`
description	This element is the root element for the description of all function exposed by a content processing plug-in. It is a substitution of the abstract element *SpecificDescriptor* described in the section 28

element **Function**

diagram 

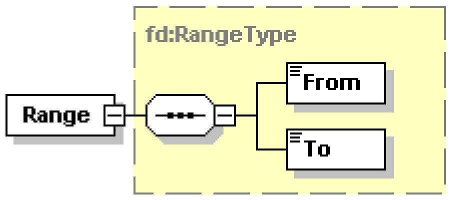
namespace	http://www.axmedis.org/plugin-function-schema
type	**fd:FunctionType**
children	**Name Version fd:FunctionDescription prm:ParameterList fd:Result**
used by	complexType **FunctionListType**
description	This element represent a single function exposed by a plug-in. A function is characterized by a Name, which is used in

the script to call the function, and a version. The *Name* element is of type *ID* while the *Version* element is a *string*

element **FunctionDescription**

diagram	
namespace	http://www.axmedis.org/plugin-function-schema
type	**xs:string**
used by	complexType **FunctionType**
description	This element can contain any string and represents the description of the function (e.g. the help for the function).

element **Range**

diagram	
namespace	http://www.axmedis.org/plugin-function-schema
type	**fd:RangeType**
children	**From** **To**
source	<xs:element name="Range" type="fd:RangeType" substitutionGroup="prm:Constraint"/>

description This element represent an interval constraint of a parameter. It is composed by two values:

- **From** is the start point of the range
- **To** is the end point of the range

both extreme limits can be included or excluded. In fact, those elements have the **included** attribute which can be true or false. This element is a substitution group of the Constrain element which is explained in the section 30.

complexType **Limit**

diagram	
namespace	http://www.axmedis.org/plugin-function-schema
type	extension of **xs:string**
used by	elements **RangeType/From** **RangeType/To**

attributes

Name	Type	Use	Default	Fixed	Annotation
included	xs:boolean	optional			

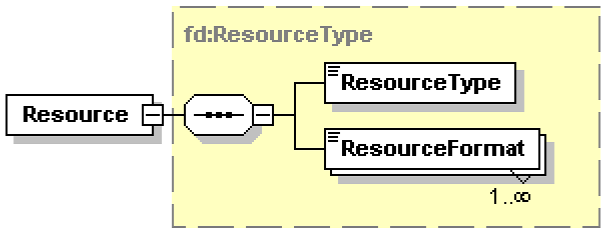
source

```
<xs:complexType name="Limit">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="included" type="xs:boolean" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

description This is the type of the *From* and *To* element contained in the *Range* element. It represents a bound of an interval of values. It can be included or not in the interval itself.

element **Resource**

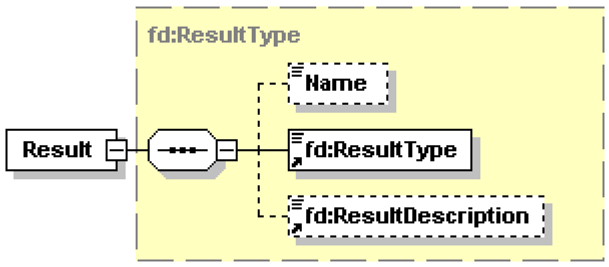
diagram



namespace	http://www.axmedis.org/plugin-function-schema
type	**fd:ResourceType**
children	**ResourceType** **ResourceFormat**
source	<xs:element name="Resource" type="fd:ResourceType" substitutionGroup="prm:Constraint"/>
description	This element represent a constraint on the type of resource which can be passed as parameter to a function. It has to be used in conjunction to a parameter of type **RESOURCE** to determine the acceptable MIME Type for it. The **Type** element corresponds to the MIME type while **Format** element corresponds to the MIME subtype. Information about the MIME content-types can be found in the RFC 2045 [RCF2045], 2046 [RCF2046] and 2077 [RCF2077]. *ResourceType* and *ResourceFormat* elements are of type *string*. This element is a substitution group of the Constrain element which is explained in the section 30.

element **Result**

diagram



namespace	http://www.axmedis.org/plugin-function-schema
type	**fd:ResultType**
children	**Name** **fd:ResultType** **fd:ResultDescription**
used by	complexType **FunctionType**
description	This element represents the return value of the function. It is mainly described by its type but it can also own a short name and a description. The *Name* element is of type *ID*.

element **ResultDescription**

diagram

namespace	http://www.axmedis.org/plugin-function-schema
type	**xs:string**
used by	complexType **ResultType**
description	This element can contain any string and represents the description of the return value of the function

element **ResultType**

diagram

namespace	http://www.axmedis.org/plugin-function-schema
type	**fd:ResultTypeType**
used by	complexType **ResultType**

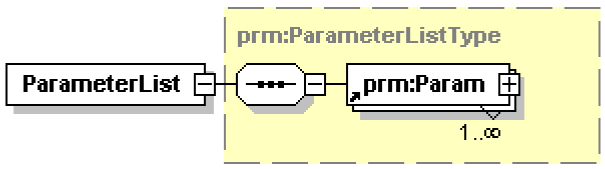
facets	enumeration	UINT16
	enumeration	INT16
	enumeration	UINT32
	enumeration	INT32
	enumeration	FLOAT
	enumeration	DOUBLE
	enumeration	BOOLEAN
	enumeration	STRING
	enumeration	WSTRING
	enumeration	CHAR
	enumeration	RESOURCE
	enumeration	AXOM
	enumeration	AREA
	enumeration	VOID
description	This element represents the type of the return value. It is almost the same of type of parameter (see section 30) but it can be also *VOID*.	

30 Formal description of format – Parameter description

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.axmedis.org/parameter" xmlns:prm="http://www.axmedis.org/parameter"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="ParameterList" type="prm:ParameterListType"/>
    <xs:complexType name="ParameterListType">
        <xs:sequence>
            <xs:element ref="prm:Param" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Param" type="prm:ParamType"/>
    <xs:complexType name="ParamType">
        <xs:sequence>
            <xs:element name="Name" type="xs:ID"/>
            <xs:element ref="prm:ParamType"/>
            <xs:choice minOccurs="0">
                <xs:element name="In"/>
                <xs:element name="Out"/>
                <xs:element name="InOut"/>
            </xs:choice>
            <xs:choice minOccurs="0">
                <xs:element name="Mandatory"/>
                <xs:element name="DefaultValue" type="xs:anySimpleType"/>
            </xs:choice>
            <xs:element ref="prm:ParamDescription" minOccurs="0"/>
            <xs:element ref="prm:Constraints" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ParamDescription" type="xs:string"/>
    <xs:element name="Constraint" type="prm:ConstraintType" abstract="true"/>
    <xs:complexType name="ConstraintType" abstract="true">
        <xs:sequence>
            <xs:any namespace="##any"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Constraints" type="prm:ConstraintsType"/>
    <xs:complexType name="ConstraintsType">
        <xs:sequence>
            <xs:element ref="prm:Constraint" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ParamType" type="prm:ParamTypeType"/>
    <xs:simpleType name="ParamTypeType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="UINT16"/>
            <xs:enumeration value="INT16"/>
            <xs:enumeration value="UINT32"/>
            <xs:enumeration value="INT32"/>
            <xs:enumeration value="FLOAT"/>
            <xs:enumeration value="DOUBLE"/>
            <xs:enumeration value="BOOLEAN"/>
            <xs:enumeration value="STRING"/>
            <xs:enumeration value="WSTRING"/>
            <xs:enumeration value="CHAR"/>
            <xs:enumeration value="RESOURCE"/>
            <xs:enumeration value="AXOM"/>
            <xs:enumeration value="AREA"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

element **ParameterList**

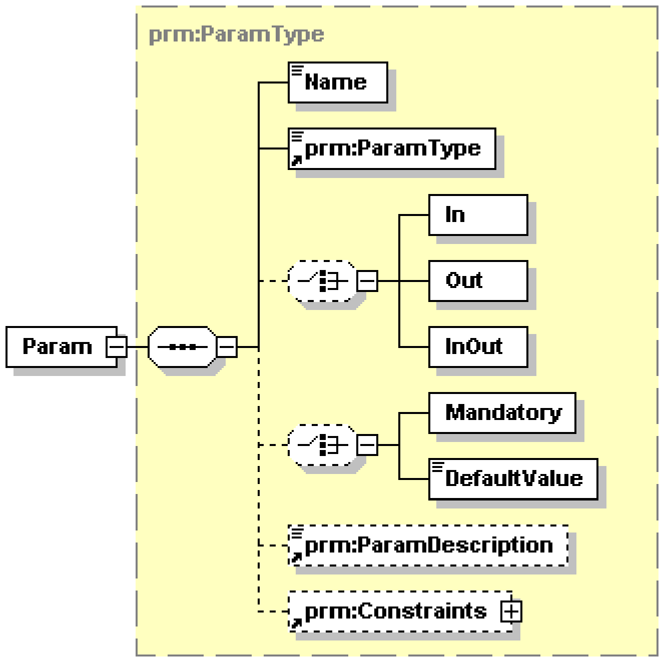
diagram



namespace	http://www.axmedis.org/parameter
type	**prm:ParameterListType**
children	**prm:Param**
description	This element represents a list of parameters.

element **Param**

diagram



namespace	http://www.axmedis.org/parameter
type	**prm:ParamType**
children	**Name** **prm:ParamType** **In** **Out** **InOut** **Mandatory** **DefaultValue** **prm:ParamDescription** **prm:Constraints**
used by	complexType **ParameterListType**

description This element represents a parameter of a plug-in function. Each parameter is characterized by the following field:

- **Name** – the name of the parameter. It is of type *ID*.
- **ParamType** – see below
- **In, Out, InOut** – only one of these field can be used for each parameter. They represent the direction of the parameter:
  - **In** – represents a parameter which will not be changed by the function
  - **Out** – represents a parameter whose value is not used by the function and which will be modified by it
  - **InOut** – represents a parameter whose value is used by the function and which will be modified by it

  by default the parameter will be considered of type **InOut**.
- **Mandatory, DefaultValue** – only one of these field can be used for each parameter. They represent the mandatoryness of the parameter:
  - **Mandatory** – it means that a value has to be given for this parameter
  - **DefaultValue** – it is used to provide a default value for the parameter if it is not explicitly given. It is of type *anySympleType*, i.e. it can contain any value which can be represented as a simple type
- **ParamDescription** – see below

- **Constraints** – see below

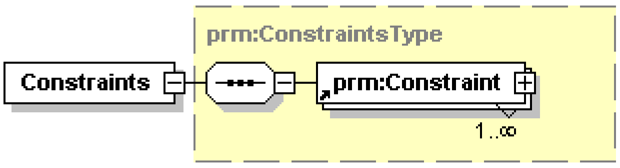
element **ParamType**

diagram	
namespace	http://www.axmedis.org/parameter
type	**prm:ParamTypeType**
used by	complexType **ParamType**
facets	enumeration UINT16
	enumeration INT16
	enumeration UINT32
	enumeration INT32
	enumeration FLOAT
	enumeration DOUBLE
	enumeration BOOLEAN
	enumeration STRING
	enumeration WSTRING
	enumeration CHAR
	enumeration RESOURCE
	enumeration AXOM
	enumeration AREA
description	This element represents the type of a parameter. The definition of this element fixes the set of parameter type which can be exchanged among AXOM and plug-ins. A non-exhaustive list is reported above. Each type reported in the list corresponds to a specific type in the programming language.

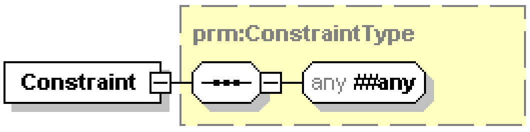
element **ParamDescription**

diagram	
namespace	http://www.axmedis.org/parameter
type	**xs:string**
used by	complexType **ParamType**
description	This element contains a human-readable description of a parameter, e.g. an help for the user.

element **Constraints**

diagram	 prm:ConstraintsType Constraints prm:Constraint 1..∞
namespace	http://www.axmedis.org/parameter
type	**prm:ConstraintsType**
children	**prm:Constraint**
used by	complexType **ParamType**
description	This element contains the constraints which the parameter is liable to. **Constraints** can contain several kind of constraints, see below.

element **Constraint**

diagram	
namespace	http://www.axmedis.org/parameter
type	**prm:ConstraintType**

used by	complexType **ConstraintsType**
source	`<xs:element name="Constraint" type="prm:ConstraintType" abstract="true"/>`
description	This is an abstract element which is the base for all those elements which represent a constrain for a given parameter (see section 29 for some example of constraint).