Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE3.1.2B Framework and Tools Specifications (Viewers and Players)

Version: 3.2
Date: 16/03/2005
Responsible: DSI

Project Number: IST-2-511299
Project Title: AXMEDIS
Deliverable Type: Private
Visible to User Groups: No
Visible to Affiliated: No
Visible to the Public: No.

Deliverable Number: DE3.1.2 part B
Contractual Date of Delivery: January 2005
Actual Date of Delivery: 17 March 2005
Title of Deliverable: Framework and Tools Specifications (Viewers and Players)
Work-Package contributing to the Deliverable: WP3.1
Task contributing to the Deliverable: WP3, WP2
Nature of the Deliverable: report
Author(s): DSI, EPFL, UNIVLEEDS, IRC< XIM, HP, FUPF, SEJER

Abstract: 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.

Keyword List: AXMEDIS tools and players, integration, viewers, user interface..

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 (DSI, all)

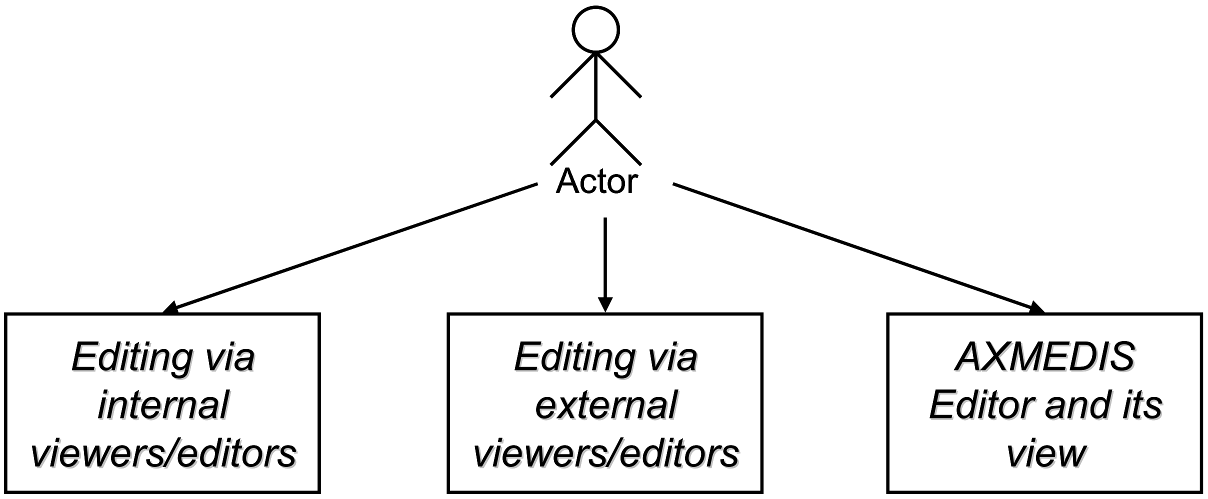
The full AXMEDIS specification document has been decomposed in the following parts:

A. general aspects up to the description of the content model
B. Viewers and players, including plug ins, etc.
C. Content Production tools and algorithms
D. Fingerprint and descriptors algorithms and tools
E. Database area, query support and Content Crawling from CMS
F. AXEPTool area, for B2B distribution and Programme and Publication for B2C distribution
G. Workflow aspects and tools
H. Protection tools and support, Certification and Supervision and Accounting tools
I. Distribution tools and AXMEDIS Portal
J. Definitions, tables, terminology, acronyms, lists, references, links and Appendixes

This document contains Part B only.

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
Other scenarios for the interoperability of the AXMEDIS Editor with other commonly used tools for content production have been consider; in the External Editor/Viewer AXMEDIS plug-ins the capability of activating other tools directly from the AXMEDIS Editor is specified. The specification details how the AXMEDIS editor can master the communication with the external tool and how the DRM and the security requirements are preserved in the data transfer.
The plug-in analysis is carried out also in other scenario about inserting AXMEDIS in the common tool for the end-user: AXMEDIS ActiveX control is specified together with AXMEDIS Plug-in for Mozilla. Other plug-in for the multimedia players (e.g. Windows Media Player) are specified considering general issues.
The AXMEDIS player provided by the AXMEDIS framework is specified in different versions: for PC, for PDA and for mobile devices.

2 AXMEDIS Object Viewers and Editors (DSI, EPFL)


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

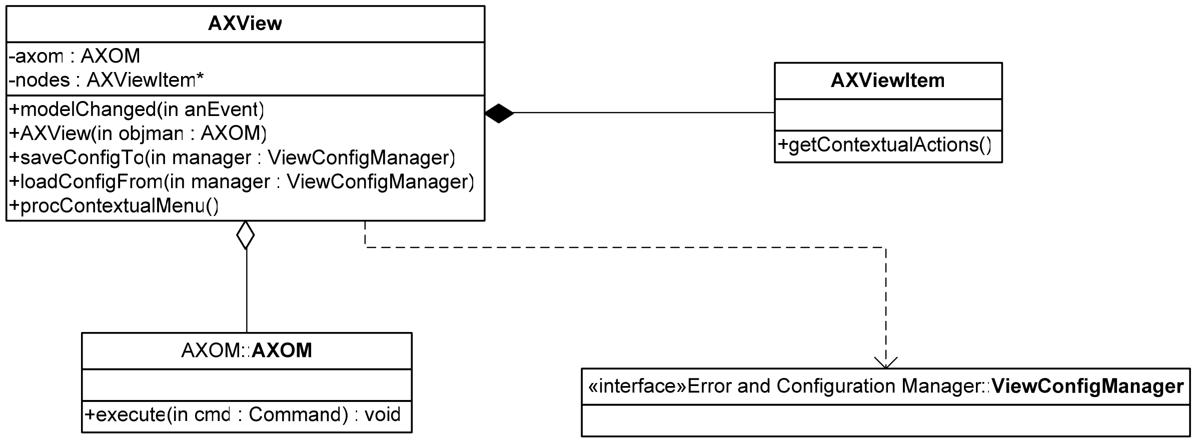
2.1 View *Modules (DRM Edit/View, Hierarchy Edit/View, Metadata Edit/View, etc…)

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;
- Spatial;
- Behavioural and Functional;
- Descriptions and comments;
- Metadata;
- Object editor
- Etc.

Common functionalities have been found in the group of AXMEDIS view:

- Management functionality: view activation
- Save/Load of view item position and state (collapsed/expanded) etc.
- Contextual Menu dynamic creation and processing
- Common relation with other classes (AXOM)

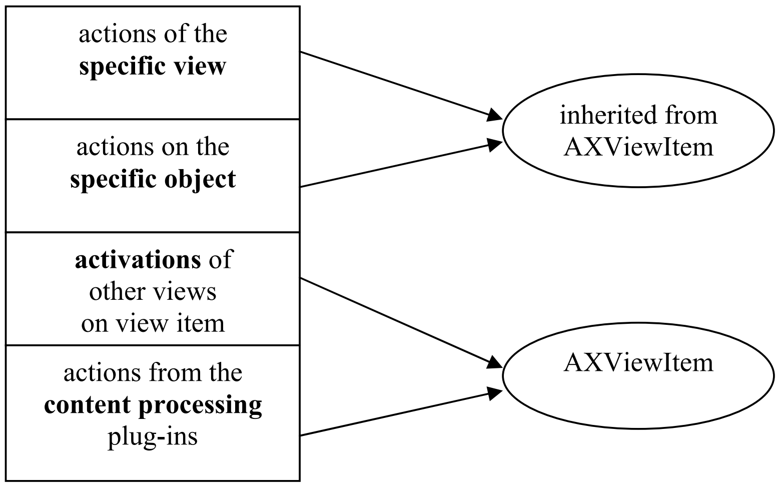

AXView
-axom : AXOM
-nodes : AXViewItem*
+modelChanged(in anEvent)
+AXView(in objman : AXOM)
+saveConfigTo(in manager : ViewConfigManager)
+loadConfigFrom(in manager : ViewConfigManager)
+procContextualMenu()
AXViewItem
+getContextualActions()
AXOM::AXOM
+execute(in cmd : Command) : void
«interface»Error and Configuration Manager::ViewConfigManager

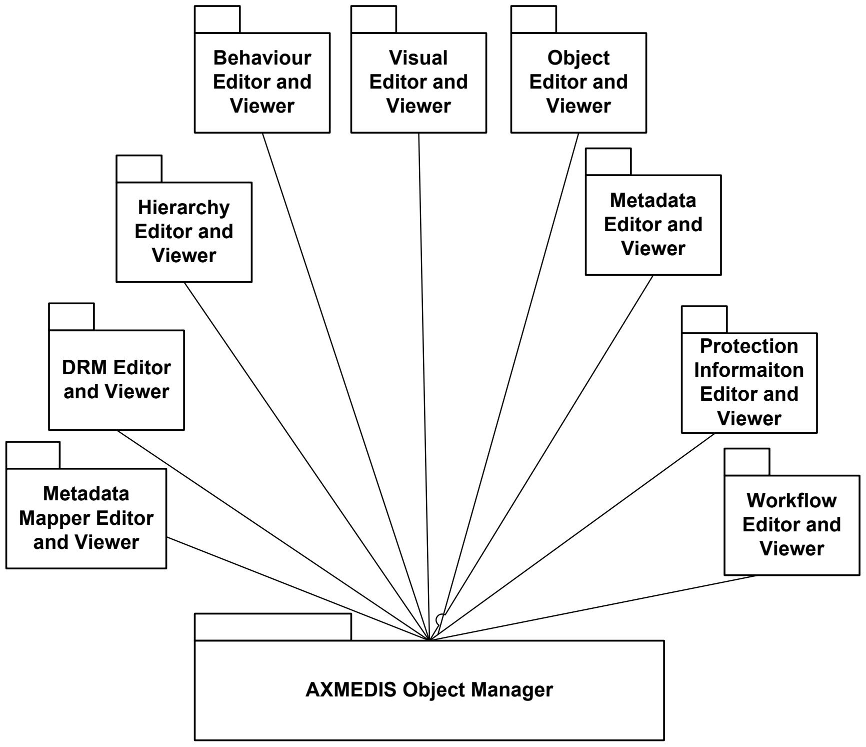
For that two basic classes are considered:

- AXView that generalize all the AXMEDIS views: this base class provides the reference to the AXOM, the node registration and a not structured collection of ViewItems in order to allow navigation of elements in a general manner; the save load function connected to a ViewConfigManager; it provides also an important function that allow a view to process the contextual actions invoked on the view items
- The view item generalize the features of every element present in any view. As first property, the capability to build its own dynamic contextual menu.

Dynamic Context Menu

In any view if the event that shows contextual menu is fired (right click or double) the dynamic contextual menu is composed as depicted in the following figure.

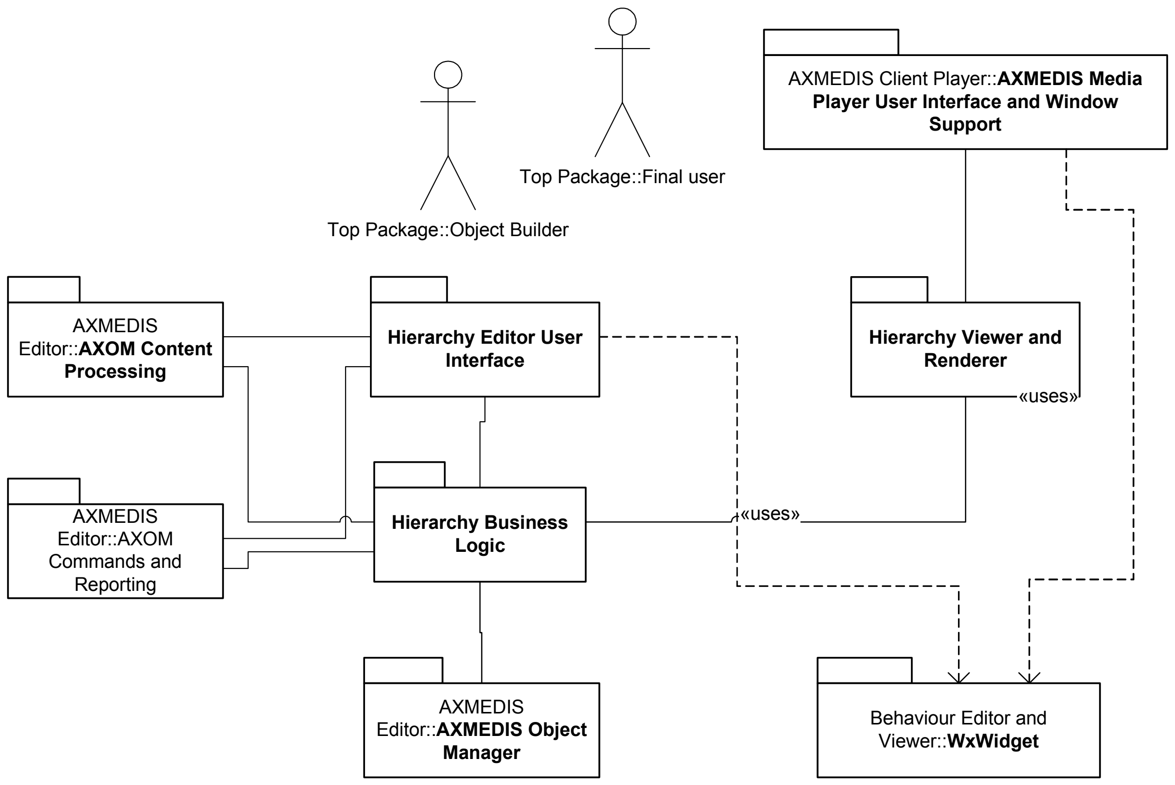

actions of the specific view
actions on the specific object
activations of other views on view item
actions from the content processing plug-ins
inherited from AXViewItem
AXViewItem


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

2.2 Hierarchy Editor and Viewer (DSI)

Module Profile		
Hierarchy Editor and Viewer		
Executable or Library(Support)	Dynamic library	
Single Thread or Multithread	Single Thread	
Language of Development	C++	
Responsible Name	Davide Rogai, Andrea Vallotti	
Responsible Partner	DSI	
Status (proposed/approved)	proposed	
Platforms supported	Microsoft Windows, Linux, MACOS X	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
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

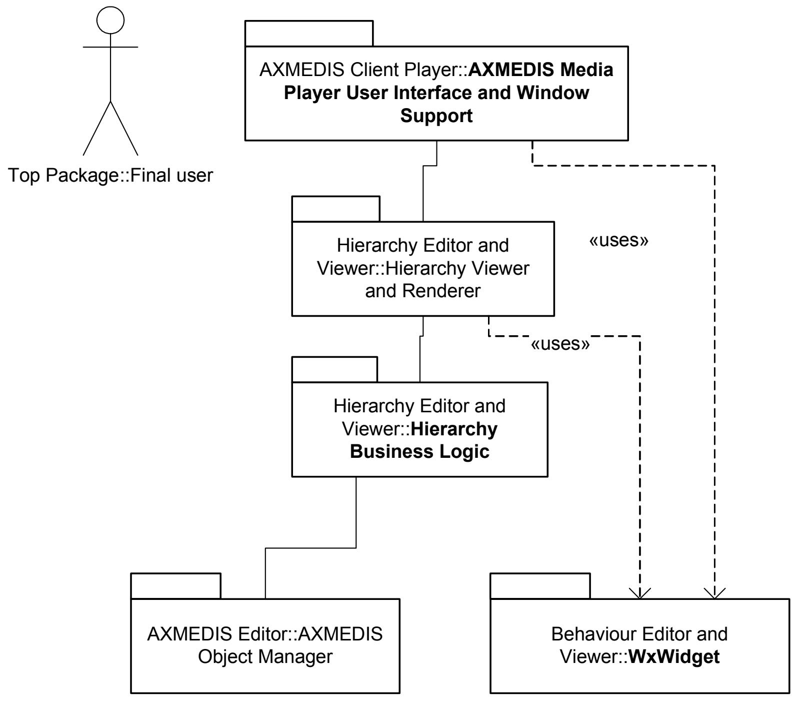
Hierarchy Editor and Viewer

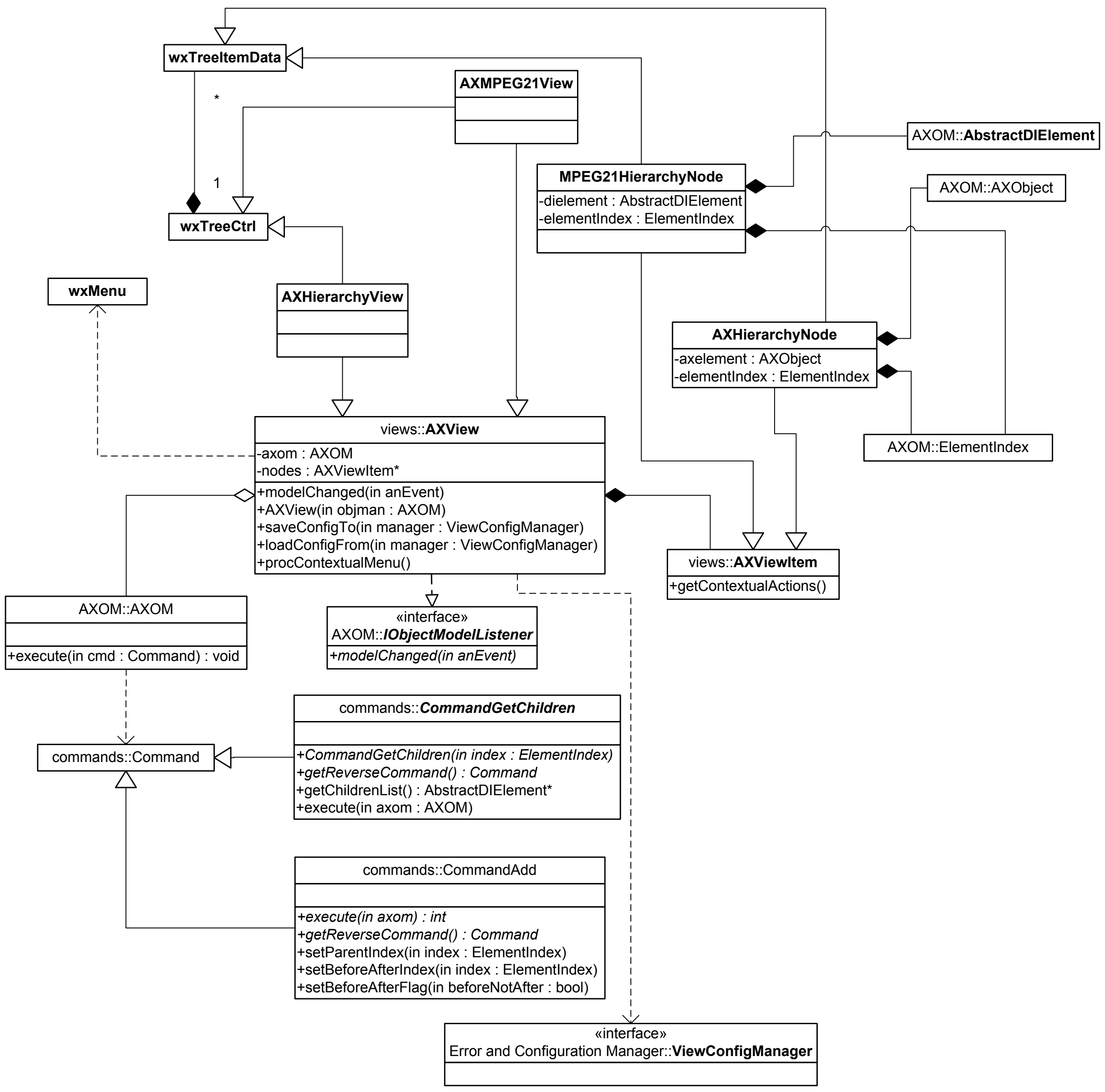

Top Package::Object Builder
Top Package::Final user
AXMEDIS Client Player::AXMEDIS Media Player User Interface and Window Support
AXMEDIS Editor::AXOM Content Processing
Hierarchy Editor User Interface
Hierarchy Viewer and Renderer
«uses»
AXMEDIS Editor::AXOM Commands and Reporting
Hierarchy Business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

Hierarchy Viewer/Player


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


wxTreeItemData
*
1
wxTreeCtrl
AXMPEG21View
MPEG21HierarchyNode
-dielement : AbstractDIElement
-elementIndex : ElementIndex
AXOM::AbstractDIElement
AXOM::AXObject
wxMenu
AXHierarchyView
AXHierarchyNode
-axelement : AXObject
-elementIndex : ElementIndex
AXOM::ElementIndex
views::AXView
-axom : AXOM
-nodes : AXViewItem*
+modelChanged(in anEvent)
+AXView(in objman : AXOM)
+saveConfigTo(in manager : ViewConfigManager)
+loadConfigFrom(in manager : ViewConfigManager)
+procContextualMenu()
views::AXViewItem
+getContextualActions()
AXOM::AXOM
+execute(in cmd : Command) : void
«interface»
AXOM::IObjectModelListener
+modelChanged(in anEvent)
commands::CommandGetChildren
+CommandGetChildren(in index : ElementIndex)
+getReverseCommand() : Command
+getChildrenList() : AbstractDIElement*
+execute(in axom : AXOM)
commands::Command
commands::CommandAdd
+execute(in axom) : int
+getReverseCommand() : Command
+setParentIndex(in index : ElementIndex)
+setBeforeAfterIndex(in index : ElementIndex)
+setBeforeAfterFlag(in beforeNotAfter : bool)
«interface»
Error and Configuration Manager::ViewConfigManager

In the above diagram the basic relation between entities are depicted:
AXOM and the interface *execute* is accessible from any view that access to an object. The AXOM executes the requested command after the needed check for the corresponding rights,. The **AXHierarchyNode** are the connection between data model and the **wxTreeItemData** of the wxWidgets library: information tree nodes of the **wxTreeCtrl**.
The wxTreeCtrl is the entity the user interact with, obviously each overriding of the wxTreeItemData refers to an AXMEDIS element:

1. filled if the *axelement* attribute refers to an **AXObject**, which is a copy of the element stored in the data model without direct references to other information inside the model, but only those needed to show element properties to the user;
2. not filled if the *axelement* is null and the **ElementIndex** refers to the AXelement that must be retrieved by the model, the retreiving operation is controlled by the AXOM.

2.2.1 Hierarchy Business Logic

The business logic under the hierarchy interface will basically cover four action on the hierarchy element

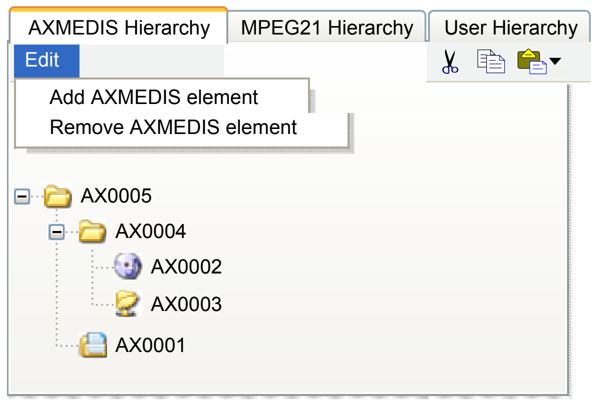
- Add – Insert Before – Insert After
- Remove
- Cut
- Copy
- Paste
- Drag and drop
- Expand
- Collapse
- Activation of other view

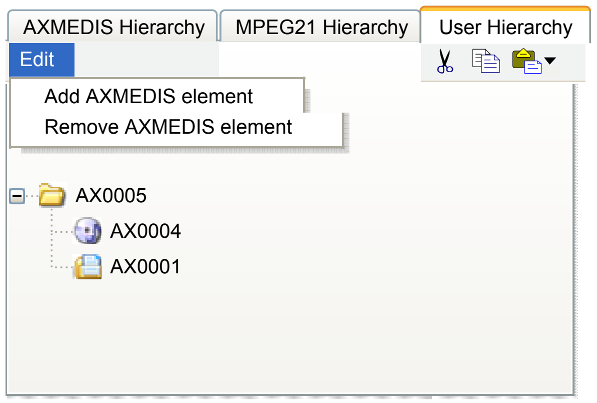
All the commands will be mapped on several command instances:

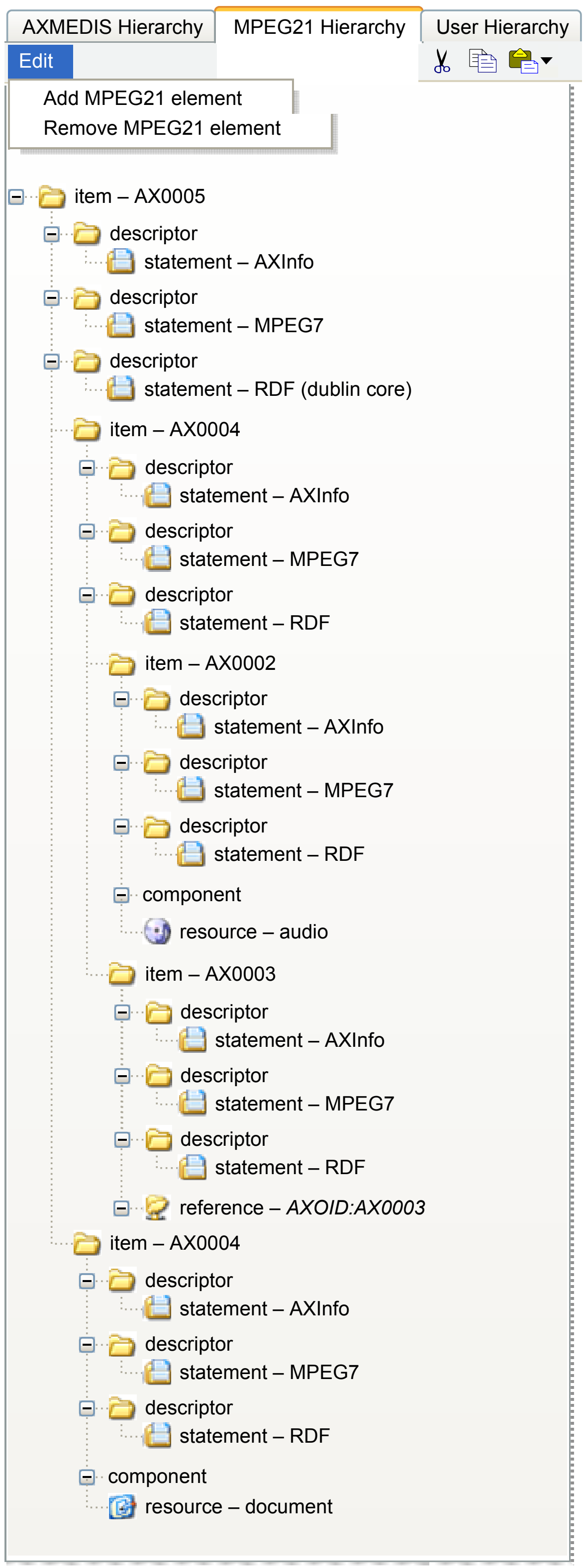
- The Add – Insert Before – InsertAfter generates a new **CommandAdd** class in order to submit such instance to the execute method exposed by AXOM, these actions will set differently the command attribute which determine the addition target in the data model, once the command is processed by the AXOM 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.

2.2.2 Hierarchy Editor User Interface

It follows two snapshots of the same Hierarchy Editor conceived as tabs.







In the above GUI sketches are represented the two roles which Hierarchical Editor plays. On the left side the AXMEDIS hierarchy is depicted: the hierarchy (parent-child relationship) in AXMEDIS shows how a

composite has been structured it also show which part are at disposal locally and which are only references to other AXMEDIS objects (see the italic text on AX0003 and the temporary icon).
The MPEG21 hierarchy is much more complicated since it shows all the relationship between MPEG21 tags of the didmodel: for example AXMEDIS metadata are mapped on several descriptors which are element of the MPEG21 hierarchy. Components and resources are depicted as well, please note that the referenced object is mapped on MPEG21 reference tag.
In addition the hierarchy view has to support drag-and-drop functionality, in order to copy or move a whole sub-hierarchy from an AXMEDIS/MPEG21 object to another, obviously such feature is supported on homogeneous view (MPEG21 to MPEG21).

An additional view is provided

2.2.3 Hierarchy Viewer and Renderer

The viewer and renderer will only allow to expand/collapse item:
All the drag-drop cut-copy-paste are inhibited ass the addition or deletion from the hierarchy

2.3 DRM Editor and Viewer (WP4.5.1: FUPF)

Module Profile		
DRM Editor and Viewer		
Executable or Library(Support)	Library (Support)	
Single Thread or Multithread	Single Thread	
Language of Development	C++	
Responsible Name		
Responsible Partner	FUPF	
Status (proposed/approved)	Proposed	
Platforms supported	PC (Windows / Linux)	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
XML based (MPEG-21 REL)		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
AXDRMView	C++	WxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
Xerces		
Xalan		
XPath / XQuery		

DRM View shall be implemented by the use of DRM User Interface, which shall permit to browse and manage all DRM-related aspects, like the following ones:

- DRM User Interface shall permit to add new rules, remove and change existing rules on the whole object or parts thereof;
- DRM User Interface shall graphically show DRM relationship among components and among actual AXMEDIS object and others object. Relationships should be represented with a graph structure within arrows and links will have given meaning/semantic;
- DRM User Interface shall be able to test the usability of an AXMEDIS object, and components thereof, in a given context. Such context could be a set of fake licenses and a set of constraints (place of consumption, time instant or interval, etc…). DRM View shall answer in two ways: true or false. In latter case, DRM View shall give a set of clauses that explain why not, e.g. a specific license is needed or a specific time interval is partially covered by the set of licenses owned by the user;

2.3.1 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).

The operations permitted over the DRM information describe above will be the following:

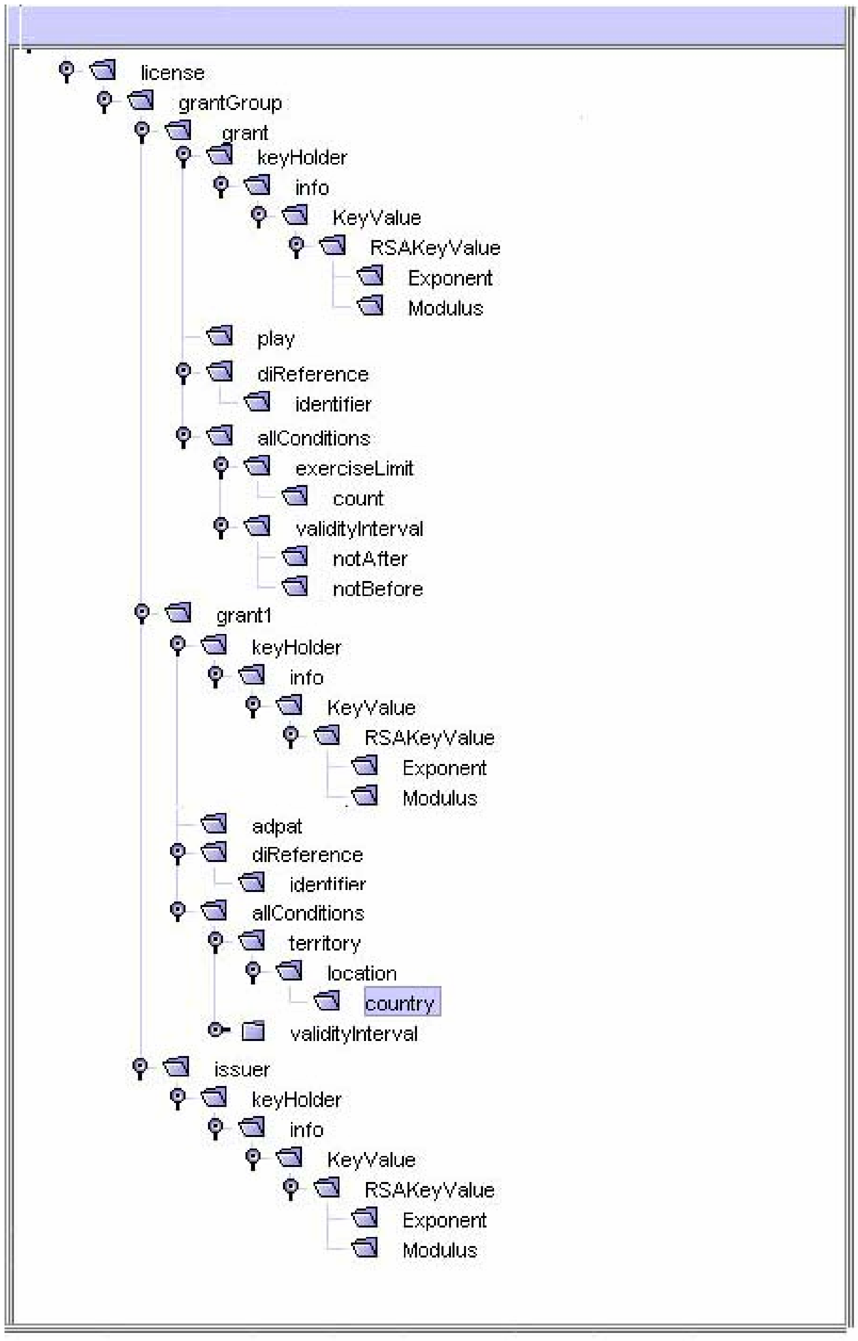
- Add a new element. The elements available will depend on the father element.
- Remove an element. This action will also imply the removal of the child elements, if any.
- Cut
- Copy
- Paste
- Drag and Drop
- Expand
- Collapse

Moreover, and not really related with the creation of DRM information, but with checking the scope of the DRM information, that is, ask if one action is available to the user under some conditions, a form will be also provided to do so.

2.3.2 DRM Editor User Interface

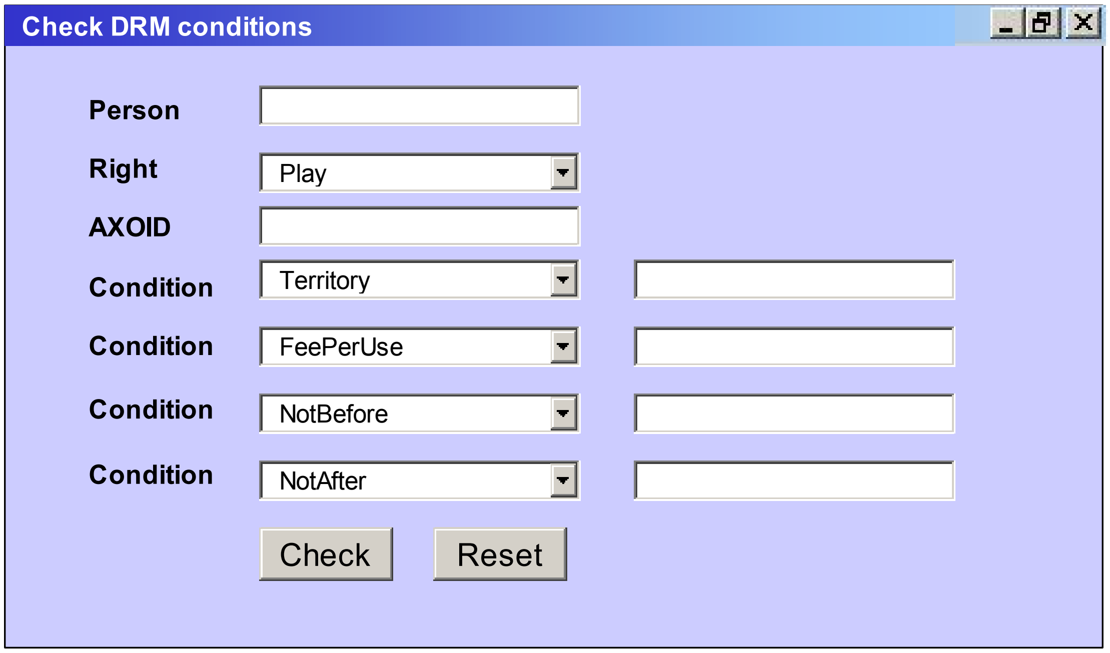
The DRM Editor user interface will have two different views, one for creation of DRM information, and the other for the checking if some action is available with the information currently created.

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

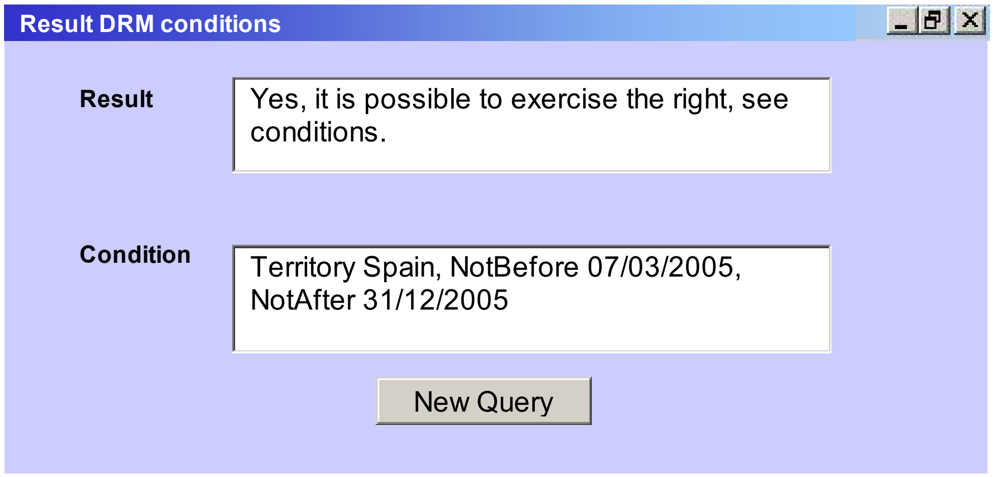


```
license
  grantGroup
    grant
      keyHolder
        info
          KeyValue
            RSAKeyValue
              Exponent
              Modulus
      play
      diReference
        identifier
      allConditions
        exerciseLimit
          count
        validityInterval
          notAfter
          notBefore
    grant1
      keyHolder
        info
          KeyValue
            RSAKeyValue
              Exponent
              Modulus
      adpat
      diReference
        identifier
      allConditions
        territory
          location
            country
        validityInterval
    issuer
      keyHolder
        info
          KeyValue
            RSAKeyValue
              Exponent
              Modulus
```

User interface for DRM editor



User interface for checking permitted actions based on the DRM information



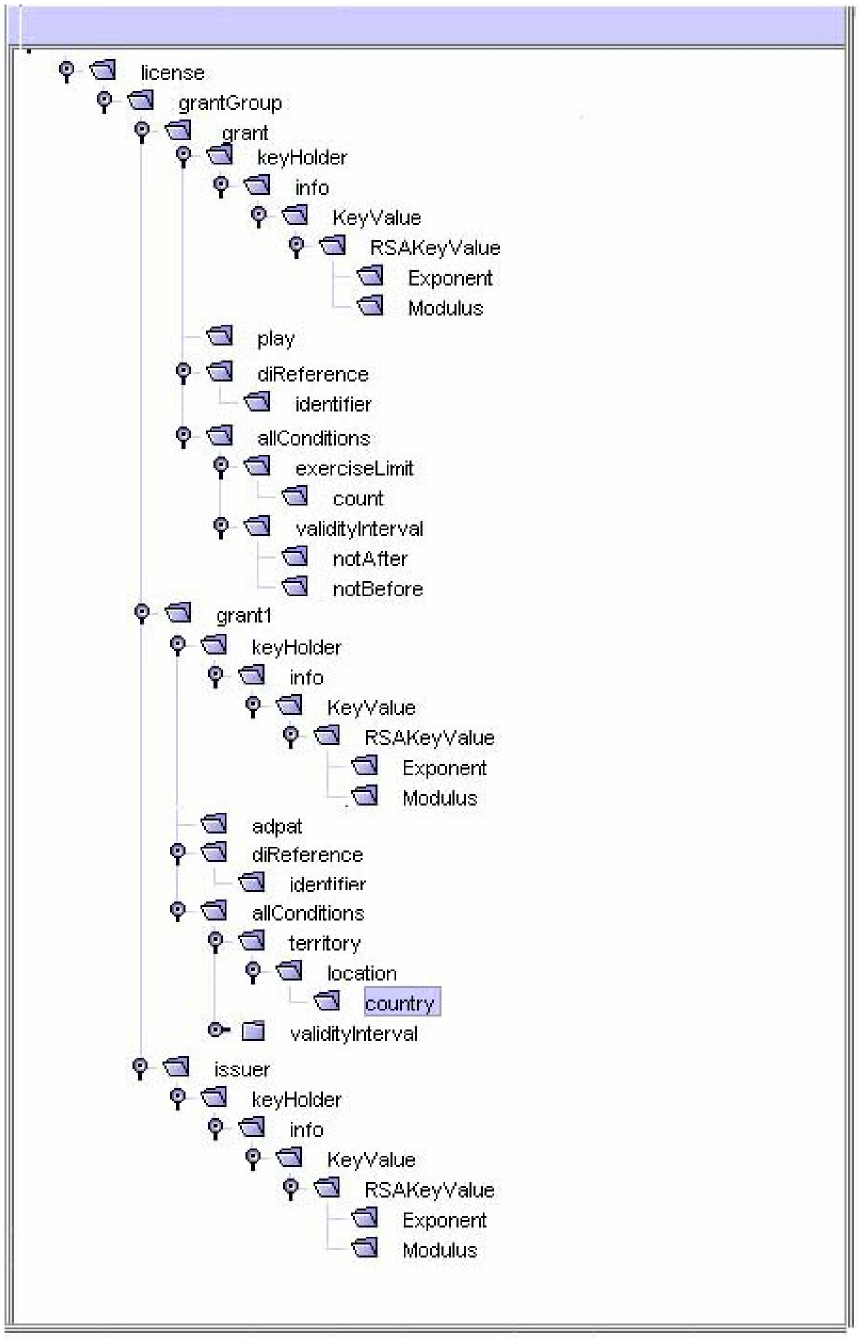
Yes, it is possible to exercise the right, see conditions.

Territory Spain, NotBefore 07/03/2005, NotAfter 31/12/2005

User interface for presenting the results of the query

2.3.3 DRM Viewer and Renderer

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.



```
license
  grantGroup
    grant
      keyHolder
        info
          KeyValue
            RSAKeyValue
              Exponent
              Modulus
      play
      diReference
        identifier
      allConditions
        exerciseLimit
          count
        validityInterval
          notAfter
          notBefore
    grant1
      keyHolder
        info
          KeyValue
            RSAKeyValue
              Exponent
              Modulus
      adpat
      diReference
        identifier
      allConditions
        territory
          location
            country
        validityInterval
    issuer
      keyHolder
        info
          KeyValue
            RSAKeyValue
              Exponent
              Modulus
```

User interface for DRM viewer

2.4 Visual Editor and Viewer (EPFL)

Module Profile		
Visual Editor and Viewer		
Executable or Library(Support)		
Single Thread or Multithread	Multithreaded	
Language of Development	C++	
Responsible Name	Giorgio Zoia	
Responsible Partner	EPFL	
Status (proposed/approved)	Proposed	
Platforms supported	Windows and Linux	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
		wxWidgets
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
	C++	
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets		LGPL

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 labelled 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.

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:

```
<smil>
<head>
  <layout>
    <!-- The Visual Editor will handle this part: 2D layout-->
  </layout>
</head>
<body>
  <!-- The Behaviour Editor will handle this part: time scheduling-->
</body>
</smil>
```

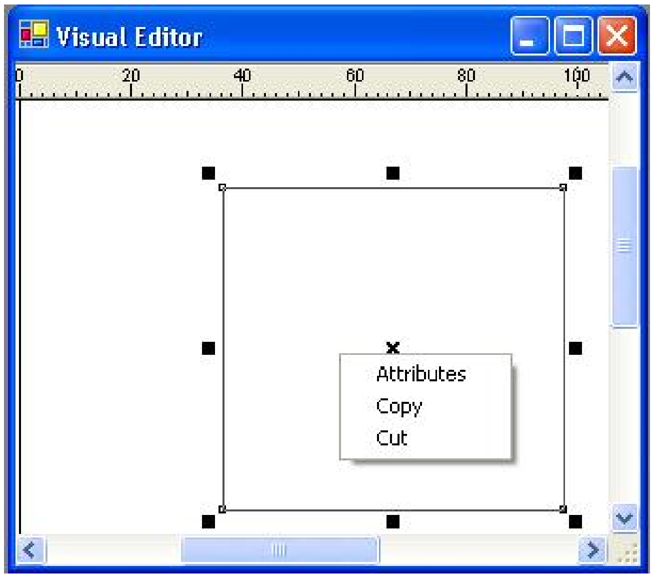
The “smil” element is needed to be SMIL compatible. The “smil” element is the root element of any presentation. It only has an identifier attribute.
The “head” element is a child of the “smil” element and it will be needed to be SMIL compatible. Like the “smile” element, it only has an identifier attribute.
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:

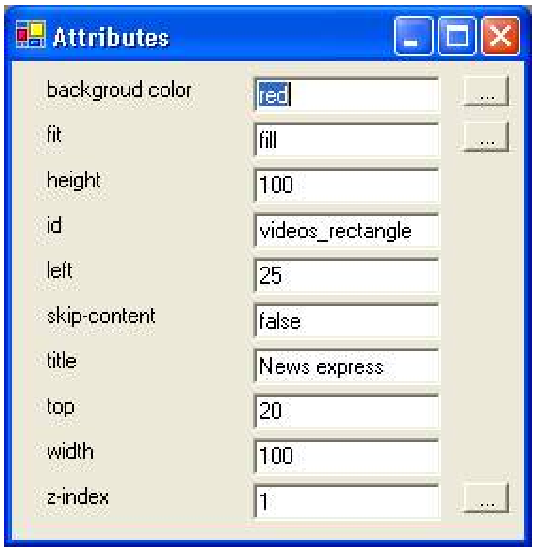
- <root-layout>: to set the width and height for the window in which the presentation will be rendered. E.g.: **<root-layout width="300" height="200" background-color="white"/>**
- <region>: to define a rectangular region of the display area where a media object will be placed. E.g.: **<region id = ”some_id”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

2.4.1 Visual Editor User Interface

The Visual Editor GUI will allow the user to draw rectangles on the screen. To draw a new rectangle it will suffice to click-and-hold a mouse button, drag the mouse, and release the mouse button; this will define the upper left and lower right corners of the rectangle. Every rectangle will represent the region where a media object will be rendered. Every rectangle will be associated with a “region” element of the SMIL document managed by the Visual Business Logic. The user will be able to draw, resize, move, copy, paste, and delete

rectangles. Every of this operations will be notified to the Visual Business Logic that in return will update the SMIL document. In this way, the SMIL document will be a faithful representation of what the rectangles the user will place on the screen. By right clicking on a rectangle the user will access a context menu (see picture below) to set the rectangle attributes: identifier, width, height, background-color etc For instance if the user changes the "width" attribute, the rectangle will be resized to accommodate to the new width; this "width" attribute will also be updated in the SMIL document.
If the user right clicks on an area where there are not rectangles another context menu will pop up. This context menu will contain the "root-layout" attributes such as width, and height . These SMIL attributes control the display area of the presentation.




Attributes

backgroud color	red
fit	fill
height	100
id	videos_rectangle
left	25
skip-content	false
title	News express
top	20
width	100
z-index	1

2.5 Behaviour and Functional Editor and Viewer (EPFL)

Module Profile		
Behaviour Editor and Viewer		
Executable or Library(Support)		
Single Thread or Multithread	Multithreaded	
Language of Development	C++	
Responsible Name	Giorgio Zoia	
Responsible Partner	EPFL	
Status (proposed/approved)	Proposed	
Platforms supported	Windows	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
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		LPGL

The Behaviour Editor together with the Visual Editor will provide the user with the infrastructure to produce multimedia presentations. A multimedia presentation can be composed of many media objects (text, audio, video, vector graphics...). The user will use the Behaviour and Visual Editors to organize the media objects in space and time. As explained in the section 2.4, the user will use the Visual Editor to place the media objects in different positions of the screen. The Behaviour Editor will complement the Visual Editor by adding time boundaries to the media objects. This means that every media object will be visible only for a period defined by the user. The simplest example for this is a slide show: the user specifies a group of slides and each slide is only visible during a slot of time defined by the user.
The Behaviour Editor will use a subset of the SMIL language to describe the multimedia presentation. SMIL (pronounced "smile") is defined as a set of XML modules which are used to describe the temporal, positional, and interactive behavior of a multimedia presentation.

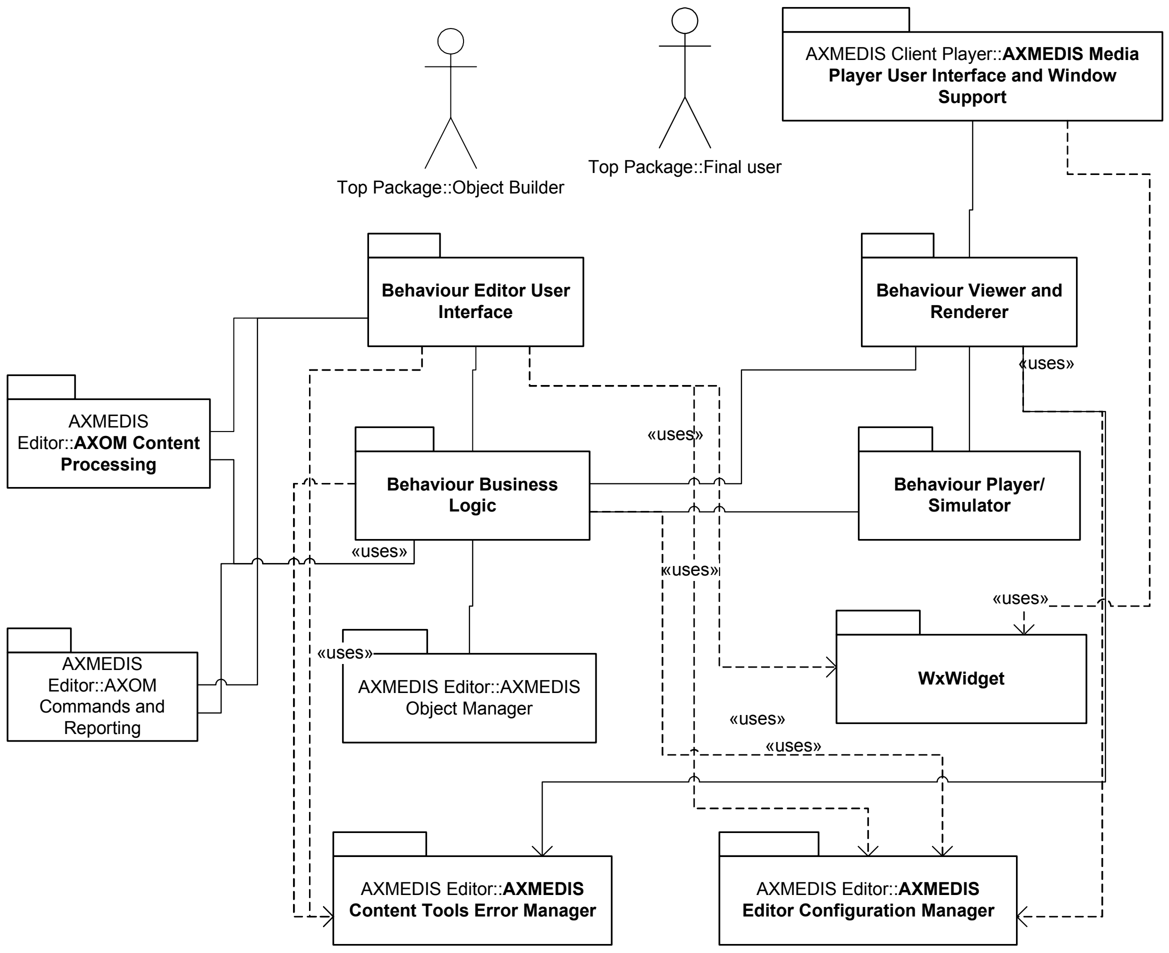
Behaviour and Functional View is intended first as a View where the main modalities and layers of the Editor can be activated. The AXMEDIS Object can be a heterogeneous, multi-layer piece of information for which different modalities of exploration/manipulation are possible and for which different layers may be accessible according to preferences. In this sense the Functional View displays the available possibilities and

allows selecting a text view, other than composited media view (scene layer) or a media-by-media view. The functional view may also allow displaying available modes of operation, like the selection between file / broadcast (save later, transmit immediately) and related configuration.
Behaviour and Functional View is also an interfacing view to external editors affecting synchronization and timing among different media objects. An AXMEDIS object may include multimedia and cross-media scenes like those that can be produced by MPEG-4 BIFS, SMIL, etc. Through suitable plug-ins and interfaces the AXMEDIS Editor may allow inserting and taking out portions or elements of these composited elements. Direct internal editing through internal functions and menus could be limited to a minimum of simple straightforward cases. Overall, the Behaviour and Functional View will:

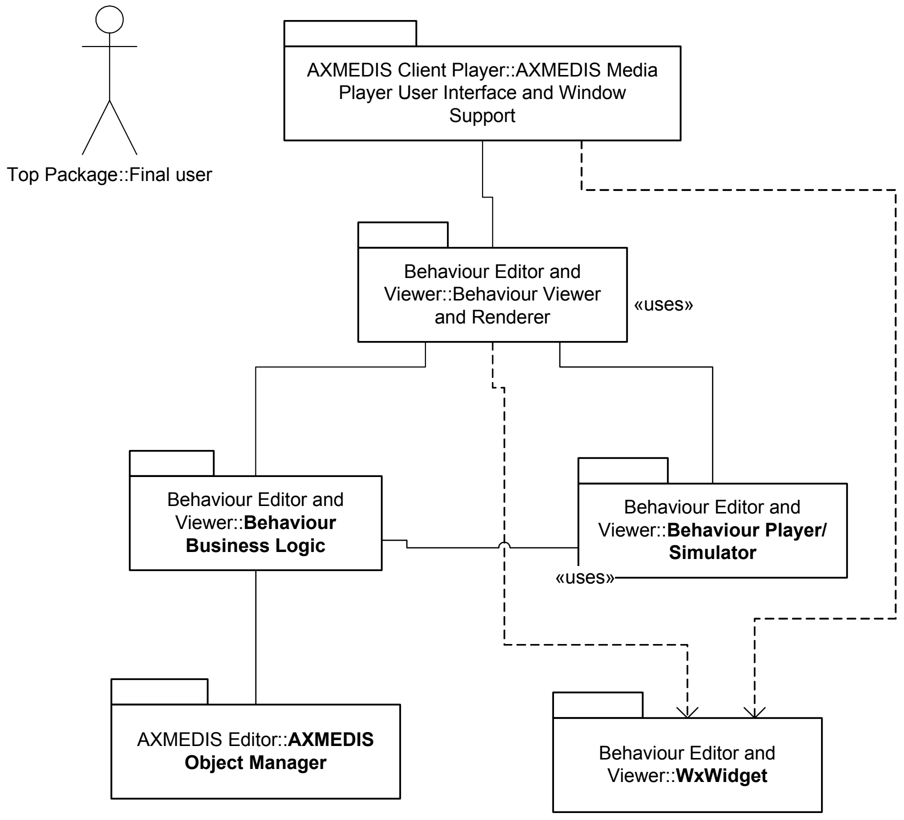
- allow the production and modification of behavioural and functional parts of the AXMEDIS Objects. These parts are the functional parts of the MPEG-21 object. They are used to describe the behaviour of the object when it is open, played, etc. It is possible in this way to describe the execution sequence and the buttons to activate them, etc. See Digital Item Processing, Digital Item Methods.
- allow switching by one touch tabs among different views such as text view (XML text view), composite media scene view, single media view (video, audio, hyperlinks, animations, etc.), media delivery view (embedded elements, streaming elements, etc.).
- for aggregated composite objects, allow activating different windows with different view modalities for different subparts of the object.
- support plug-ins to show and manage specific composite media types that are to be reasonably expected for these elements, given the complex nature that these elements may have. At least a few of the major formats and tools for multimedia synchronization and timing (BIFS, SMIL), including maybe also QT, avi (divx) and the like.
- permit the visualization of simple time diagrams and/or spatial layouts from AXMEDIS objects to permit selection of some parts (on the base of annotations, etc.) to activate related editing tools for the supported functionality and formats

be configurable, i.e. user should be able to select, for each kind of components, which view to activate by default.

Behaviour Editor and Viewer
This has to be taken as an example of any AXMEDIS Editor and Viewer connected to the AXOM for working on its details.


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

Behaviour Viewer/Player
The structure of this component and tool could be taken as an example of any Viewer/Player connected to the AXOM for rendering its details in final players/viewers.


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

2.5.1 Behaviour Business Logic

The Behaviour Business Logic will maintain an XML document. This XML document will describe the schedule of a group of media objects. It will be possible to specify when every media object will be displayed, for how long, and when the media object will abandon the display area. To describe the temporal behaviour of the media objects we will use SMIL. SMIL (see also the Visual Editor section) is an XML based language that enables the description of audiovisual presentations.
The user will interact with the Behaviour GUI to define, in a graphical way, the showtimes of the media objects. The Behaviour Business Logic will receive notifications from the Behaviour GUI and it will modify the SMIL document accordingly. In this way, the SMIL document will become a text description of the temporal planning that the user will design in a graphical manner.
It is important to notice that the SMIL document edited by the Behaviour Business Logic is the same document edited by the Visual Editor. However, there will be no conflict because the Behaviour Editor and the Visual Editor will modify different parts of the same document. The Visual editor is concerned with the spatial plannig while the Behaviour Editor regards the temporal planning. Fortunately, a SMIL document keeps in separate sections the schedule and the layout. The SMIL document will be hold inside the AXMEDIS document and will be treated as any other Component. This means that the SMIL document will be accessed via the Command Manager.
The Behaviour Business Logic will be able to handle a subset of the SMIL language. To incorporate media objects in the presentation we will need the following media tags:

- **img**: a still image (jpg, gif, etc)
- **video**: a video (mpeg, avi, mov, etc) formally we can use any kind of video here. For instance, mpeg-4 provided that we have a SMIL player that is able to instanctiate an mpeg-4 player.
- **text**: plain text.

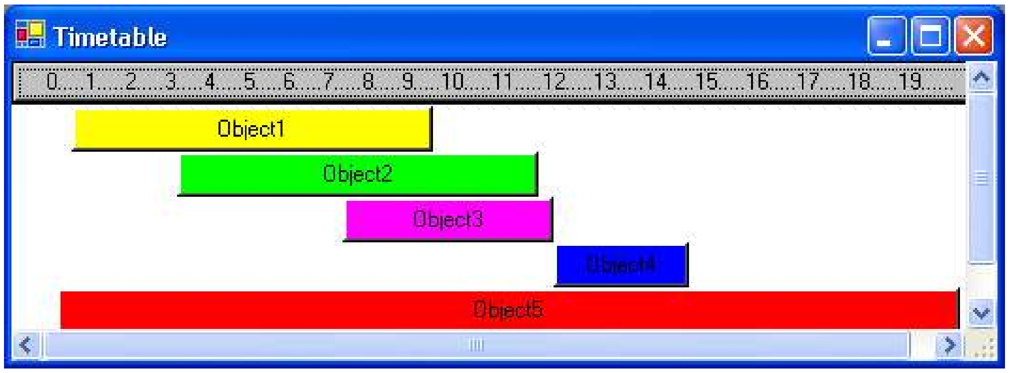
The media objects will need a start time, an end time, and a region on the screen where they will be rendered. This will be achieved by using the following attributes:

- **begin**: defines when the element will be visible (start playing).
- **dur**: defines how long the element will be visible.
- **region**: contains the identifier of a region on the screen defined with the Spatial Editor.

Putting it all together, we could have something like this:
<video src=”weather.mpg” region=”videos_rectangle” begin=”0s” dur=”59s” />
The line above states that the video weather.mpg will be played for 59 seconds in the region on the screen named “videos_rectangle” that must has been previously defined using the Visual Editor.
Finally, we will need to use the **par** (parallel) tag. We will enclose all the media tags inside a **par** tag pair. In SMIL, the **par** tag is used to display media objects simultaneously. However, we can also present elements sequentially by setting the **begin** attribute appropiately.

2.5.2 Behaviour Editor User Interface

The Behaviour GUI will display graphically the temporal boundaries of every media object. This GUI will have a horizontal time line similar to a ruler (see next picture) with marks on it that will indicate the time. The media objects will be represented as horizontal bars of different lengths according to the duration of the media object. The bars will be drawn under the time line in such a way that the limits of a bar will be aligned with the start and end times. This way, every bar will give a graphical impression of when a media object will start playing and when will it stop.
To add, modify or remove bars in this GUI the user will use a contextual menu accessible via right click. To associate a media object with a bar, the user will have to select a media object from the Hyerachical View and drag it onto the bar. To define the region of the screen where the media object will be rendered, the user will have to select a rectangle from the Visual Editor and drag it onto the bar.

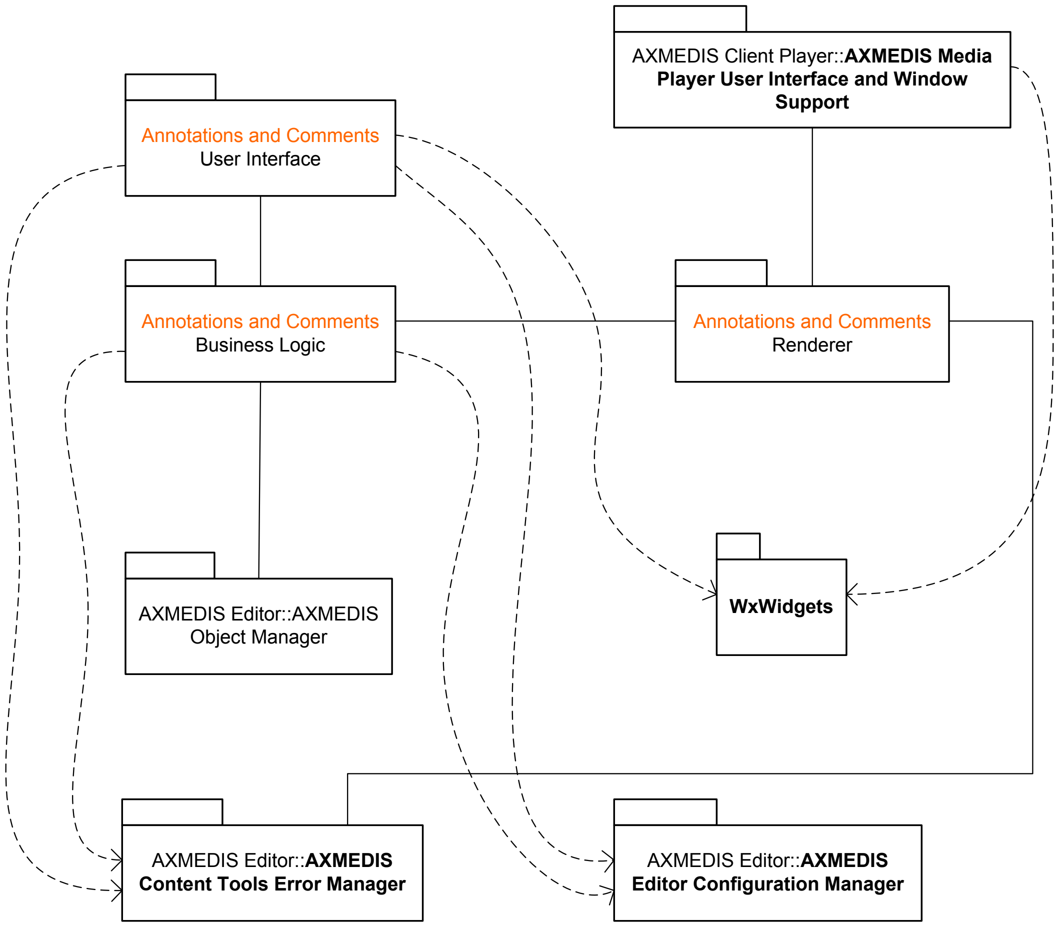


2.6 AXMEDIS Object Editor and Viewer (Descriptions and Comments) (EPFL, DSI)

Module Profile		
AXMEDIS Object Editor and Viewer		
Executable or Library(Support)		
Single Thread or Multithread	Multithreaded	
Language of Development	C++	
Responsible Name	Giorgio Zoia	
Responsible Partner	EPFL	
Status (proposed/approved)	Proposed	
Platforms supported	Windows, Linux	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
		wxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets	wxWidgets	LGPL

Annotations and Comments View will permit to display information including annotations and comments within AXMEDIS objects or parts of them. Annotations and Comments will be logically included as elements in AXMEDIS objects without actually modifying their contents. Such View will be user customizable, i.e. users will be able to select which types of media have to be made accessible for each component. Annotations and Comments View will:

- permit to add/edit/delete annotations and comments by means of graphical actions such as drag-and-drop, contextual menus, etc
- be configurable, i.e. user should be able to select, for each kind of components, which annotations and comments types (textual, audio, alternatives, etc…) have to be made accessible;
- support plug-ins to show and manage specific media types that are to be reasonably expected for these elements, given the different nature that annotations and comments may have. Other than normal text support, at least audio annotations/comments and possibly text-to-speech should be included; still pictures should also be available in at least one common format.
- support for printing and visualising the metadata in a human understandable format.


AXMEDIS Client Player::AXMEDIS Media Player User Interface and Window Support
Annotations and Comments User Interface
Annotations and Comments Business Logic
Annotations and Comments Renderer
AXMEDIS Editor::AXMEDIS Object Manager
WxWidgets
AXMEDIS Editor::AXMEDIS Content Tools Error Manager
AXMEDIS Editor::AXMEDIS Editor Configuration Manager

2.6.1 Business Logic

Instead of "Annotations and Comments", "Comments" will be used from now on. The Comments Business Logic is the 'brain' of the Comments Editor. This is the list of characteristics of the Comments Business Logic:

- This block is able to associate a set of Comments with an AXMEDIS object.
- This block does not need to ask permission to the AXMEDIS Object Manager to perform any action because the Comment operations do not interfere with the DRM rules of the AXMEDIS object.
- This block does not need to report its activities to the AXOM Commands and Reporting because the Comment operations do not affect the integrity of the AXMEDIS object.
- This block can ask to the AXMEDIS Object Manager information about the current AXMEDIS Object: name, playing state, paused state...
- This block is able to read and write the file system where the Comments will be stored.
- This block is able to instantiate a simple text editor to display text Comments and to allow the user to write text Comments.
- This block is able to instantiate a simple audio player/recorder to record or play audio Comments.
- This block is able to instantiate simple picture viewer/editor to allow the user to display graphic Comments.
- This block is able to store the configuration of the Comments Editor.

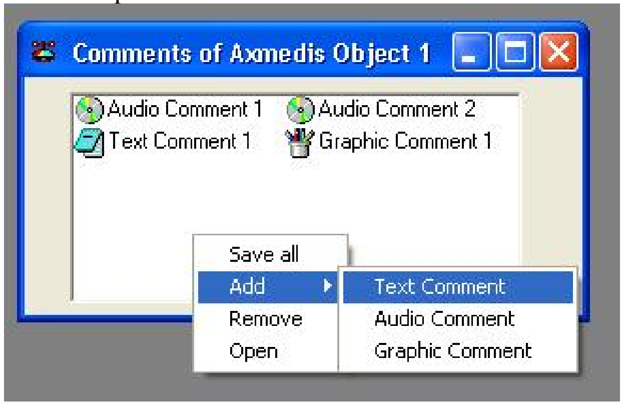
2.6.2 User Interface

The Annotations and Comments User Interface will be composed of several Graphical User Interfaces (GUIs): one for each type of Comment, a main GUI that will group all the Comments belonging to the same AXMEDIS object, and finally a Configuration GUI that will be displayed though the AXMEDIS Editor Configuration Manager.

2.6.2.1 Main GUI

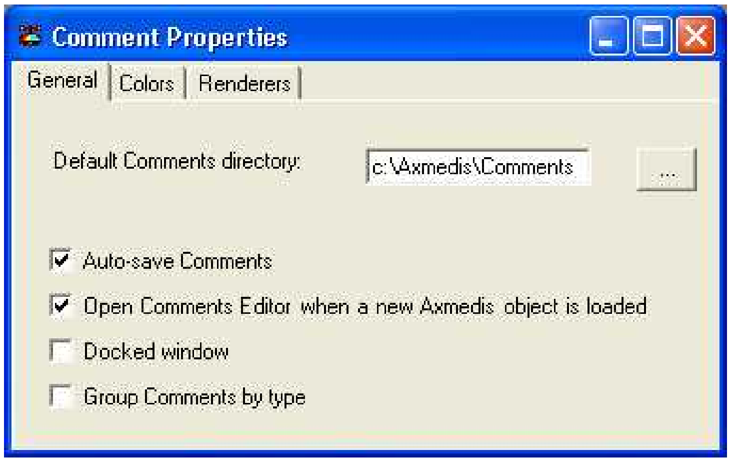
The main GUI will show the list of Comments associated with one AXMEDIS object. This GUI will permit the following operations by means of a contextual menu (see next picture) :

1. Save all
2. Add a Comment
3. Remove a Comment
4. Open a Comment



2.6.2.2 Configuration GUI

The Configuration GUI will allow the customization of the Comments Editor. The user will be able to configure the default type of comments e.g: plain text or the default directory where the comments will be stored. This could be implemented as ActiveX Property pages (in PC platform). By using Property pages this GUI can be driven and be made accessible from the AXMEDIS Editor Configuration Manger which is in charge of centralizing the configuration information of the whole AXMEDIS Editor.



Default Comments directory: c:\Axmedis\Comments

[✓] Auto-save Comments

[✓] Open Comments Editor when a new Axmedis object is loaded

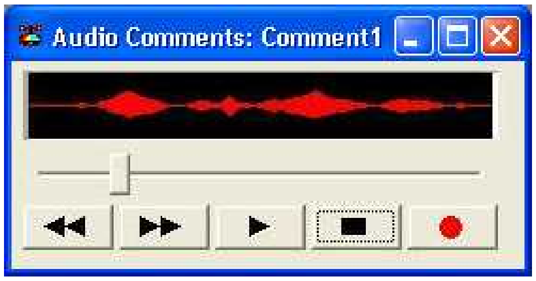
[ ] Docked window

[ ] Group Comments by type

2.6.2.3 Renderer GUIs

Different types of Comments need different GUIs. A simple text editor will be needed to have text comments. For audio Comments a simple audio player/recorder will be needed. It is not necessary that this player is able to play/record compressed formats like mp3; it suffices if it is capable of playing/recording in non-compressed format. Finally a simple image/drawing editor will be provided.

The following picture shows how the Comments Audio Renderer could look like.



2.7 Metadata Editor and Viewer (UNIVLEEDS)

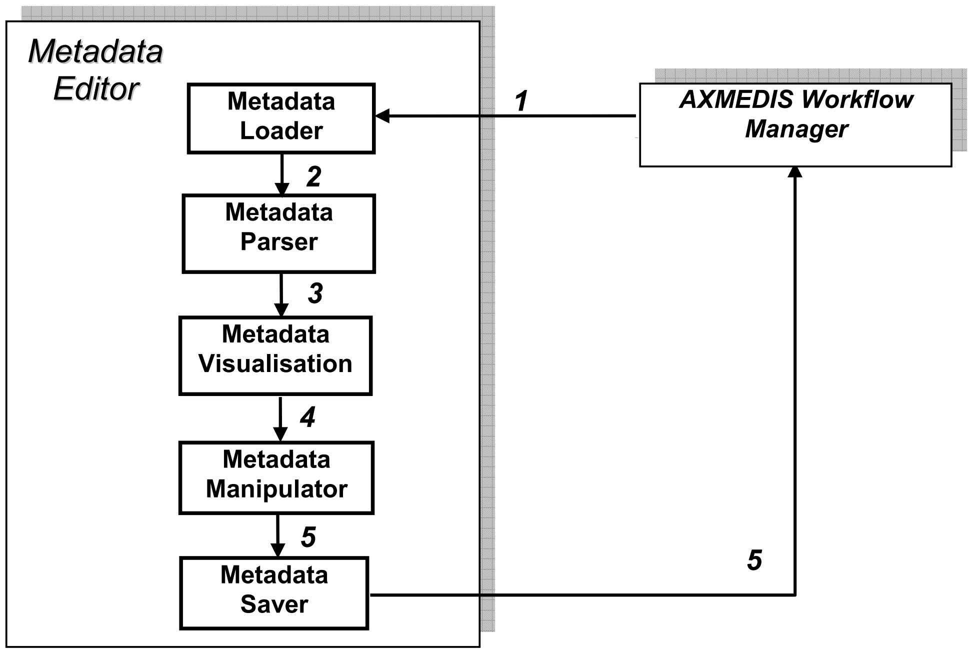
Module Profile		
Metadata Editor and Viewer		
Executable or Library(Support)	Executable	
Single Thread or Multithread	Single Thread	
Language of Development	C++	
Responsible Name	Kia Ng and Royce Neagle	
Responsible Partner	UNIVLEEDS	
Status (proposed/approved)	Proposed	
Platforms supported	Windows XP	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
Query Support		
AXOM Commands and Reporting	Plug in Interface	
File Formats Used	Shared with	File format name or reference to a section
XML		
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
	wxWindows, v-2.5.3 or v-2.4.2	LGPL
	Xerces-c++ v-2.6.0	Apache Software Licence, v2.0
	Xalan-c++ v-1.9	The Apache Software License, v1.1

Metadata Viewer provides functionality to display 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, AXInfo.

- Metadata Viewer shall be able to adapt itself (e.g. by analysing the data-related XML schema) automatically to the metadata structure;
- Metadata 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;

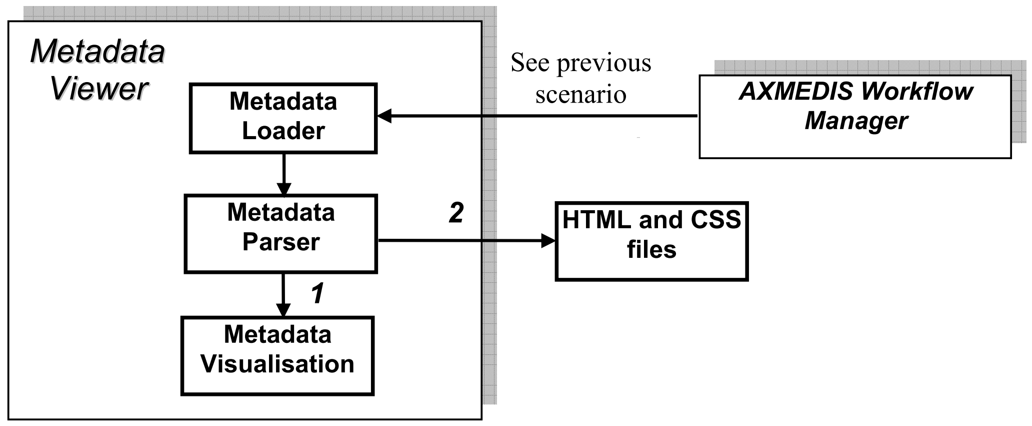
Scenario for editing metadata

1. Receive metadata information in XML from WF
2. Parse XML data 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 send back the revise metadata to the originator (via WF)

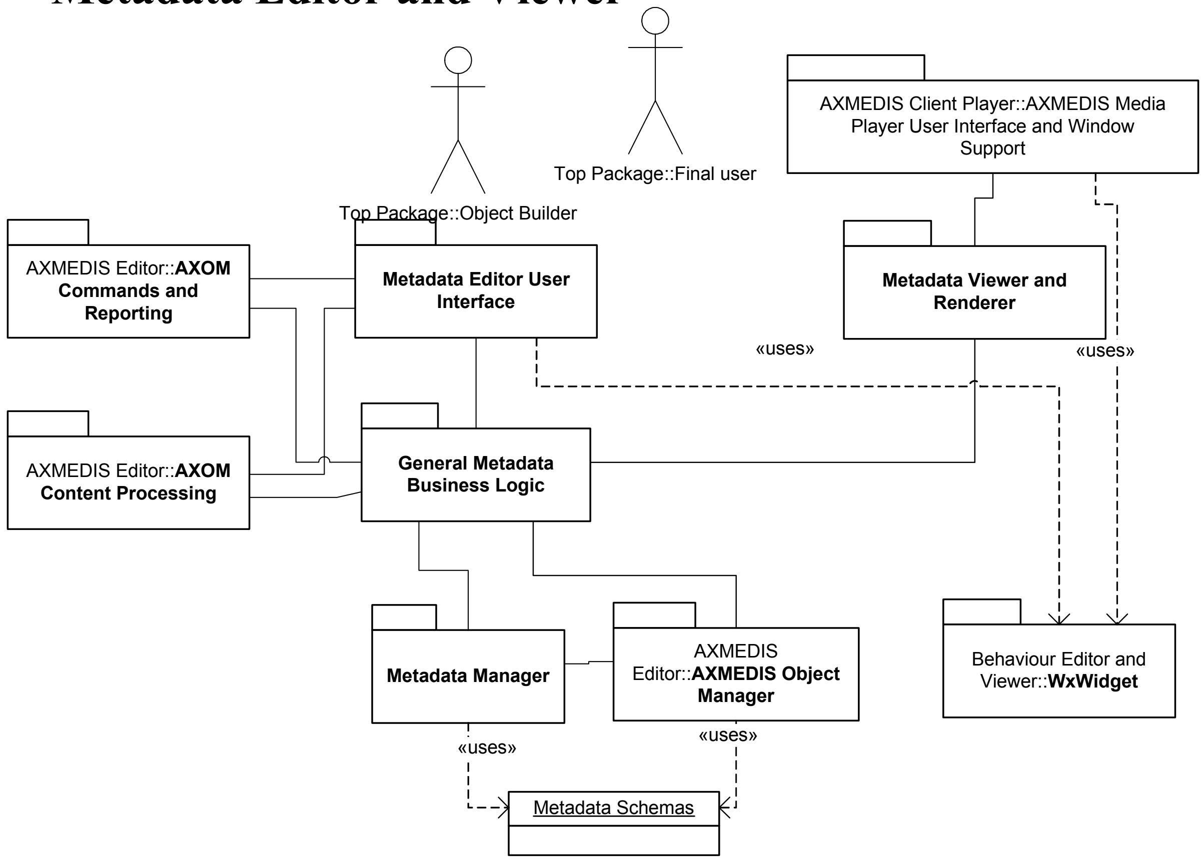


For Metadata visualisation, two possible approaches can be done:

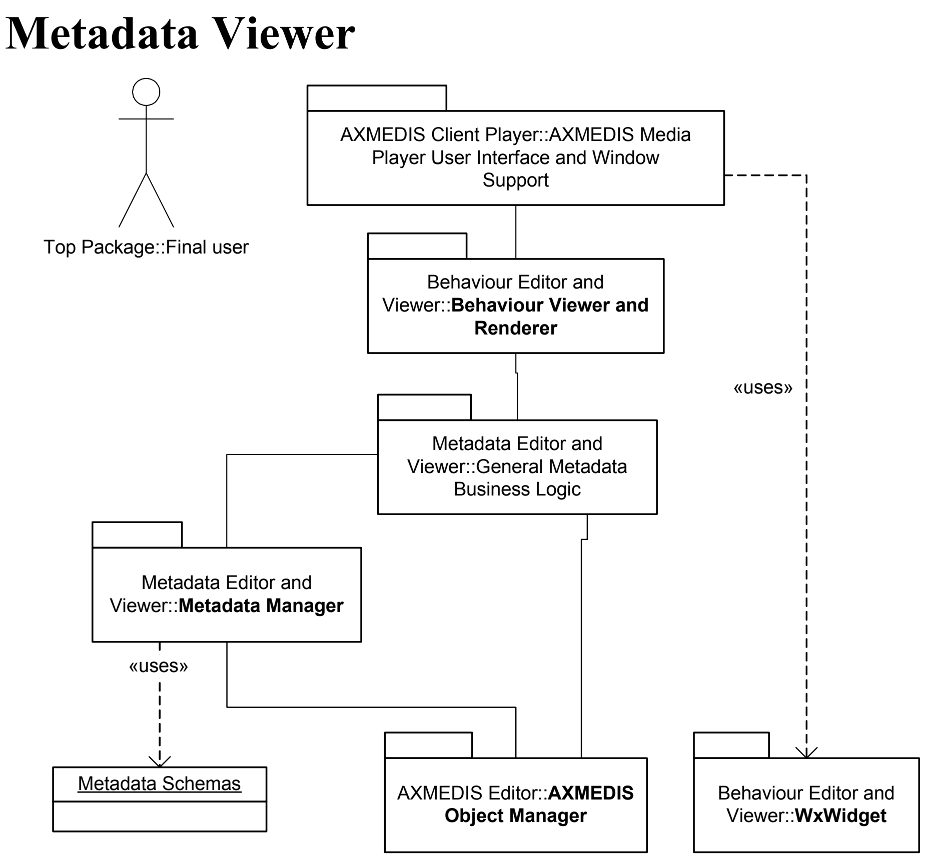
1. Using the above editor without the manipulation and save functionality (4 & 5) activated
2. Generating a HTML file (with CSS) and use a browser to display the file



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

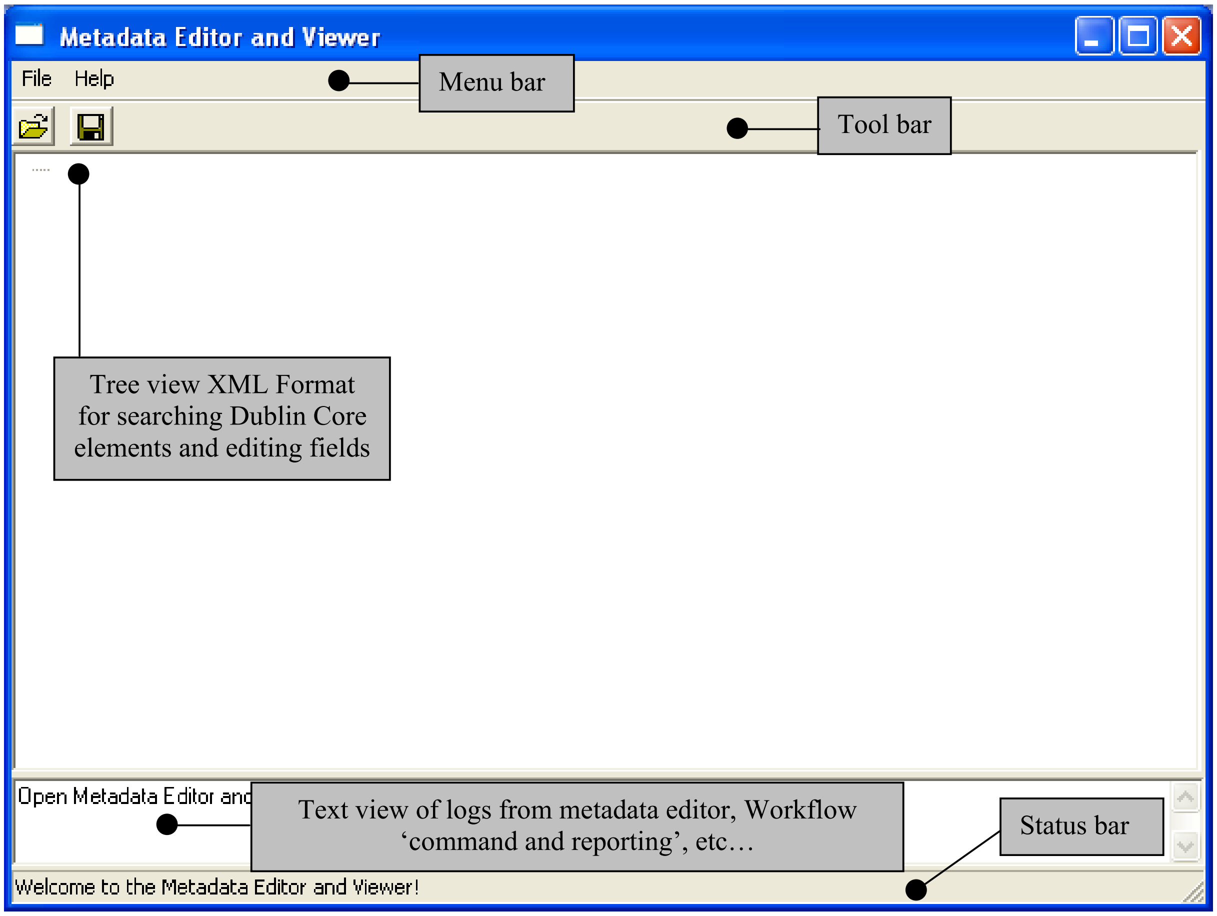
This architecture should allow to cope with different Metadata Sets simply by changing the Metadata Schemas (also taken from the AXMEDIS object, and in particular from the AXInfo): The interested test cases should be UNIMARC, Doublin Core, and all the AXInfo data. The Role of the Metadata Manager is 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

2.7.1 General Metadata Business Logic

The General Metadata Business Logic provide 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 what is the valid child for a particular nodes depending on the context, where the user intended to add an elements. The Business Logic preserves the structure integrity and ensure the correctness of the updated XML.

2.7.2 Metadata Editor User Interface


Metadata Editor and Viewer
File
Help
Menu bar
Tool bar
Tree view XML Format for searching Dublin Core elements and editing fields
Open Metadata Editor and
Text view of logs from metadata editor, Workflow 'command and reporting', etc…
Status bar
Welcome to the Metadata Editor and Viewer!

The Menu Bar
The menu bar will be constituted of the following entries:

File
- **Load** – load XML
- **Save** – save the XML and send via WF
- **Save as** – save metadata as XML on local system
- **Exit** – Quit the editor/viewer

Messages (This may be automatically set to always show messages)
- **Last message –** Displays the last message sent by the AXMEDIS Workflow Manager into the Log text window
- **Messages List -** Displays the list of messages sent by the AXMEDIS Workflow Manager

Help
- **Help** – Call the on line help
- **About** – Information about the authors, version etc.

Tree view area
In this area the structure of the XML is displayed. It will be visualised using a Tree control that will permit to show and browse components according to the generic XML inputted. This view will also permit the editing of fields for the elements for editing

Textual Visualisation Area
This is a text control where log messages, textual description, alert, etc… are displayed.

2.7.3 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 apply a linearization to the structure in order to minimise unnecessary complexity unrelated to metadata editing purposes, removing recursive references.

2.7.4 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.

2.7.5 Metadata Viewer and Renderer

There are two approaches to achieve the metadata visualisation:

1. uses the metadata editor (as described above) with the editing functionalities disabled
2. to automatically generate HTML (with pre-defined or user-defined CSS) and uses standard browser (e.g. IE) for visualisation

2.8 Workflow Editor and Viewer (IRC, XIM, HP)

Module Profile		
Workflow Editor and Viewer		
Executable or Library(Support)	Executable OpenFlow User Interface based upon Zope User Interface	
Single Thread or Multithread	Multithread	
Language of Development	User interface: Zope DTML (a superset of html) Application logic: Python or DTML	
Responsible Name		
Responsible Partner		
Status (proposed/approved)		
Platforms supported	Microsoft Windows, Linux, Mac OS X	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
WorkFlow Engine		Via Zope Web Server
File Formats Used	Shared with	File format name or reference to a section
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Zope Web-based U.I.	DTML, a superset of html	Idem
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
OpenFlow	OpenFlow 1.1	GPL 2.0
Zope by Zope Corporation	Zope 2.7.3	ZPL 2.0 (Zope Public Licence 2.0), open source, GPL compatible
Python by Stichting Mathematisch Centrum, Amsterdam, The Netherlands.	Python 2.3	Free Open Source by Stichting Mathematisch Centrum, Amsterdam, The Netherlands.
Xmlrpclib by Secret Labs AB and by Fredrik Lundh	Xmlrpclib	Free Open source by Secret Labs AB and by Fredrik Lundh
C expat Library by James Clark	C expat	Mozilla Public Licence Version 1.1

The workflow editor and viewer is the gateway interface for creating and changing new project workspaces referred to as NPDs in the terminology adopted for the AXMEDIS Workflow and object life cycle analysis elsewhere in our document.

Naturally the functionality of this editor/viewer at the level of NPD editing will be a subset of the use cases already set-out for the AXMEDIS workflow management system particularly focusing on the global management requirements of the NPD workspace including Actors, Objects, Processes, etc. as follows

A) Search is a generic use case that can search for anything at any time both synchronously (during user interaction online; i.e. whilst the client workspace-instance is online and running, or alternatively whilst it is in-pause/offline (asynchronous search). The search can be blind/exploratory or more specific as in a special case that can be inherited to search for eligible components to be worked on; as follows:

- Search in the AXWF Database for (sub)processes that have to be worked by the user; this is based on specific search criteria
- Retrieve from AXWF DB, the Axmedis Components associated with (sub)processes
- Invoke the Axmedis Object Manager (or Axmedis DB Manager) to search for the components based on search criteria (to be used by the search engine deployed by the Axmedis Object Manager)

B) Create a New Product Development, which typically entails creating new Axmedis Object instance and a new workflow process instance; as follows:

- Create a new process instance in the AXWF DB
- Invoke the Axmedis Object Manager to create new Axmedis Object
- Link the process instance to the Axmedis Object instance

C) Discard an NPD, which implies the termination of the process instance:

- Cancel the process instance from the AXWF DB
- Invoke the Axmedis Object Manager to remove the associated Axmedis Object

D) Add a Component to a specified NPD; this can imply the creation of a new sub-process instance:

- Invoke the Axmedis Object Manager to create a new component in the Axmedis Object instance
- If the process flow requires it, create a sub-process instance in the AXWF DB
- Link the sub-process instance to the Axmedis Object component

E) Remove a Component from a specified NPD; this can imply the termination of a sub-process instance:

- Invoke the Axmedis Object Manager to delete the component
- If the process flow requires it, also cancel the associated sub-process instance

F) Start an Activity in the process flow instance (work_item) selected by the user (for example start editing a component):

- The user selects from the work_items list and the choice of activity to start
- The WF manager automatically performs the selected actions in the process flow:
- Launching compositional/ formatting/ loading tool /publication tool /protection tool/ Program and publications engine; or launching the local PC editor tool (see below for details)

G) Group is responsible for bundling components, people, processes, partners, projects, teams, packets, digital assets products, etc into one entity which may be further referred to.

H) Show the Component in which the user has to work:

- Invokes the local PC viewing tool for showing the component associated with a work-item in the work-item list (see below for details)

I) Track Component: shows the history of what has so far been performed on the component

- Invokes the Axmedis Object Manager (or the Axmedis DB Manager) for retrieving the history of an Axmedis component associated with a work-item in the work-items list

J) Track CPA identifies the Critical Path Activities (CPA) and produces all the information regarding those activities e.g. people involved, components being worked on, processes needing attention, possibly implicitly or explicitly also tracks CPA-slack-critical objects/processes, etc.

K) Time-stamp Generate: it is an internal AXWFM Object Manager function, not visible to the user, typically invoked by the Check-in/Check-out function

L) The AXWFM Object Manager invokes the Axmedis Object Manager (or Axmedis DB Manager) to update the tracking information

M) Generate Version: again a function internal to the AXWFM Object Manager; can be explicitly executed by the user or automatically generated by the AXWF process

- The WF Object Manager invokes the Axmedis Object Manager (or Axmedis DB Manager) for updating the Object/component revision

N) List Work: lists in a hierarchical view the work-items in which the user can or has to perform activities

- The WF Object Manager retrieves from the AXWF DB the list of the work-items in which the user or team is involved

O) Select a workitem is responsible for selecting a workitem from the work-list

P) Complete a Task sends the WorkFlow engine the trigger that causes it to have the respective user activity recorded as completed and then to go to the next activity in the process-instance flow

Q) Distribute Work: AXWF function used for assigning activities to users

- The AXWF Object Manager invokes the AXWF DB for changing the process-instance information related to users

R) Change State/Phase: again a function internal to AXWF Object Manager; can be explicitly executed by the user or automatically generated by the AXWF process

- The AXWF Object Manager invokes the Axmedis Object Manager (or Axmedis DB Manager) to update the Object State/phase; phase change typically occurs as a result of developmental changes as reflected in the workflow-instance/NPD-instance

S) Global Viewer: shows details about the NPD

- Invokes the local PC viewing tool for showing the Axmedis Object associated to a work-item in the work-item list (see below for details)

T) Notification: used to send notifications to other users in the project
   - This may or may not invoke an external Notification engine

U) Check-in: used to lock an Axmedis component, and copy it to a user exclusive access area, ready for download:

   - Invoke the Axmedis Object Manager for locking the Object/component

   - Copy the Object component in a Server area of exclusive access to the user, so that the user can download it to his PC disk

   - Invoke the Axmedis Object Manager for updating tracking information

V) Check-out: Applicable only to previously checked-in Objects. After having uploaded the modified Axmedis component, re-loads it onto the Axmedis Object Manager and updates tracking data

   - Copy to the Axmedis Object Manager the selected Object component from the Server area of exclusive access of the user concerned

   - Invoke the Axmedis Object Manager for unlocking the Object/component

   - Invoke the Axmedis Object Manager for updating tracking information

2.8.1 Workflow Editor Business Logic

It will be possible for the AXMEDIS workflow management system to support inter-factory workflow. An example of this would be collaborating content producers who work jointly on common objects. Content Factory A would create an object, then Factory B would perform some activities to add value to the object, then returning it to Factory A for completion. Conceptually, this process is identical to the normal, intra-factory scenario where activities are carried out in one content factory. In the inter-factory scenario, the collaborating factories will need to establish an agreed workflow in order to manage their division of work productively and efficiently. This workflow agreement can be modelled in the same manner as a conventional intra-factory workflow. We are not proposing centralised single server architecture; rather each partner will have their workflow running with their part of the project workflow definition. The transitions resulting into change of partner will be defined in the workflow to reflect the collaborative workflow logic as agreed between the collaborators. The waiting period for the factories can be defined as “Idle/wait” activities within the workflow which are completed upon receiving the workitem from the external factory. For example when the workitem is handed to Factory B from Factory A, as defined in the workflow, Factory A will then start an “Idle/wait” activity which will end upon receiving the workitem back from Factory B.

It is important that collaborating factories therefore share common WFMS tools in order to manage and track the progress of an NPD across their combined activities. This enables dynamic planning and scheduling of resources across the factories, much in the way that automotive companies operating just-in-time policies use integrated logistics systems to track components through their value chain.

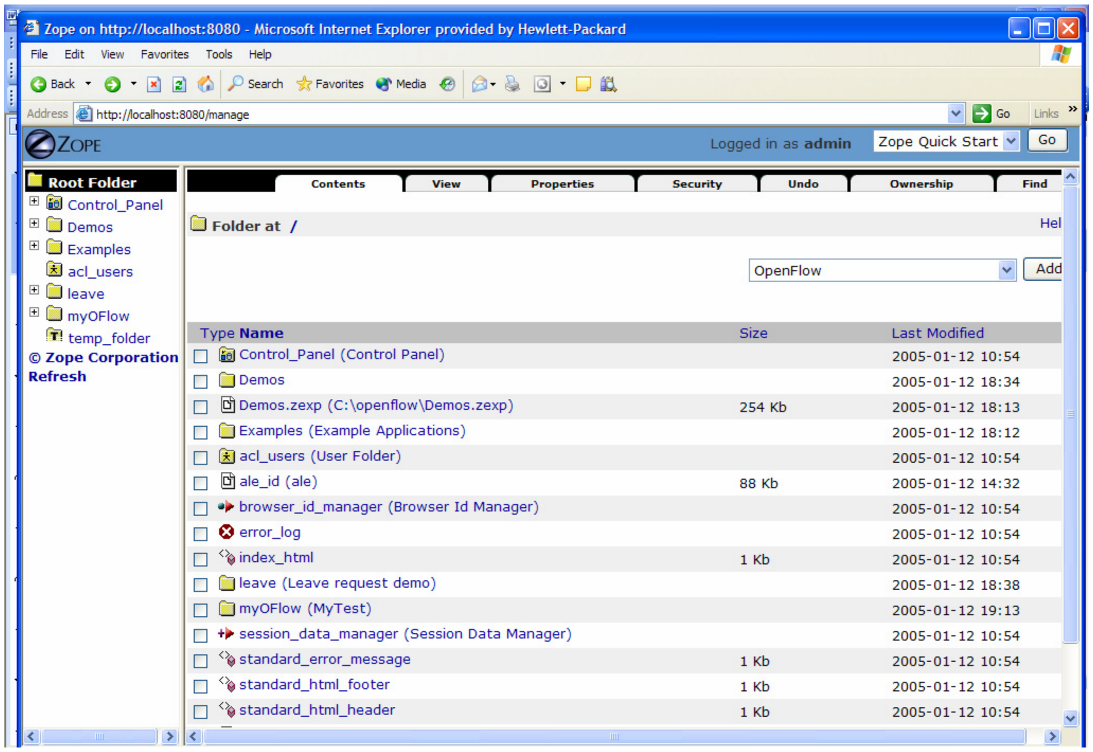
This dynamic visibility would not be possible if separate WFMS tools were employed in each factory and the only communication available were some embedded historic metadata within objects passed between factories.

For this reason, a common web-based editor will be used for the AXMEDIS Workflow user interface, which will be capable of being accessed from multiple collaborating content producers, integrators and distributors sharing a common inter-factory workflow.

2.8.2 Workflow Editor User Interface (Openflow)

Openflow runs on the Zope platform which is managed through the "Zope Management Interface" using industry standard browsers, typically by logging on as the administrator (admin) at URL http://localhost:8080/manage. The screen shot below shows an example of this management interface.

Creating a new process in openflow is a multi-step process which begins with adding an OpenFlow container using the Zope management interface as shown below (delineated by a an ellipse in red).

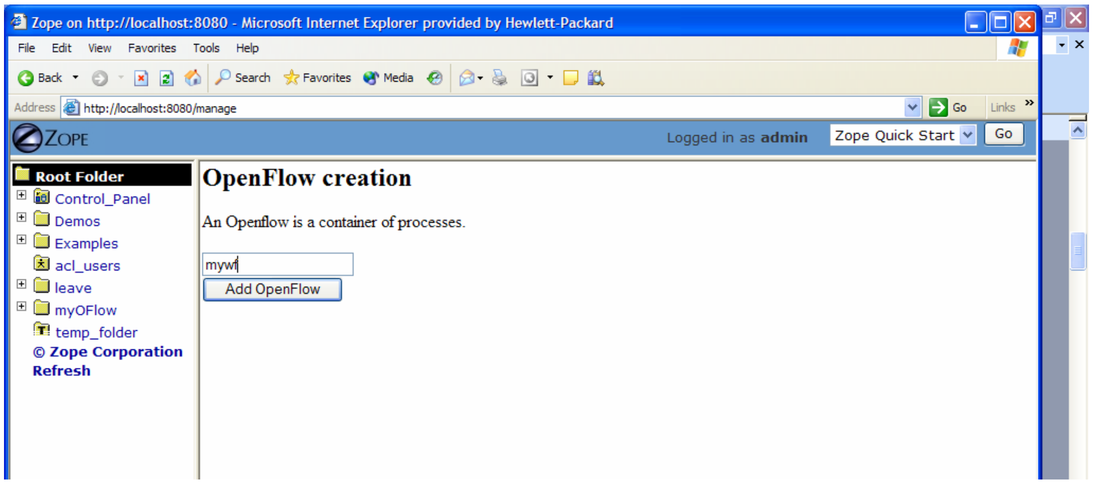

Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packard
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/manage Go Links
ZOPE
Logged in as admin
Zope Quick Start Go
Root Folder
Control_Panel
Demos
Examples
acl_users
leave
myOFlow
temp_folder
© Zope Corporation
Refresh
Contents View Properties Security Undo Ownership Find
Folder at /
Hel
OpenFlow
Add

Type Name	Size	Last Modified
Control_Panel (Control Panel)		2005-01-12 10:54
Demos		2005-01-12 18:34
Demos.zexp (C:\openflow\Demos.zexp)	254 Kb	2005-01-12 18:13
Examples (Example Applications)		2005-01-12 18:12
acl_users (User Folder)		2005-01-12 10:54
ale_id (ale)	88 Kb	2005-01-12 14:32
browser_id_manager (Browser Id Manager)		2005-01-12 10:54
error_log		2005-01-12 10:54
index_html	1 Kb	2005-01-12 10:54
leave (Leave request demo)		2005-01-12 18:38
myOFlow (MyTest)		2005-01-12 19:13
session_data_manager (Session Data Manager)		2005-01-12 10:54
standard_error_message	1 Kb	2005-01-12 10:54
standard_html_footer	1 Kb	2005-01-12 10:54
standard_html_header	1 Kb	2005-01-12 10:54

Figure 1: Adding an OpenFlow container through the Zope Management Interface

During the creation of the OpenFlow container, the name of the container must be specified as shown in the next screen-shot.

.


Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packard
File Edit View Favorites Tools Help
Address http://localhost:8080/manage
Logged in as admin
Zope Quick Start
Go
Root Folder
Control_Panel
Demos
Examples
acl_users
leave
myOFlow
temp_folder
© Zope Corporation
Refresh

OpenFlow creation

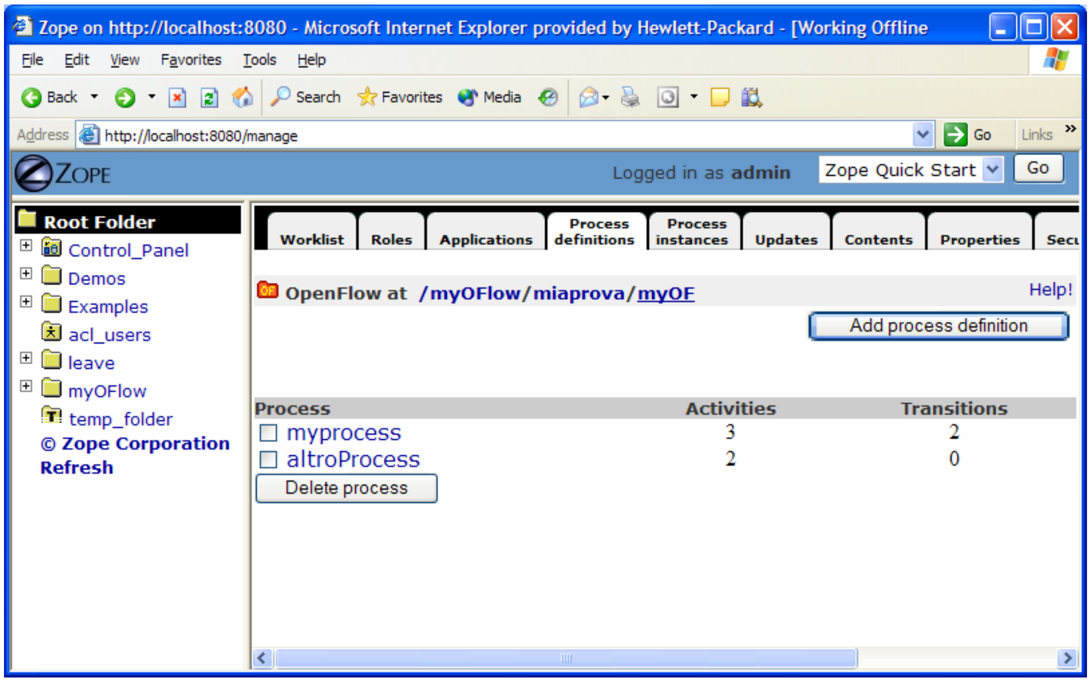
An Openflow is a container of processes.

mywf

Add OpenFlow

Figure 2: Creating the OpenFlow container

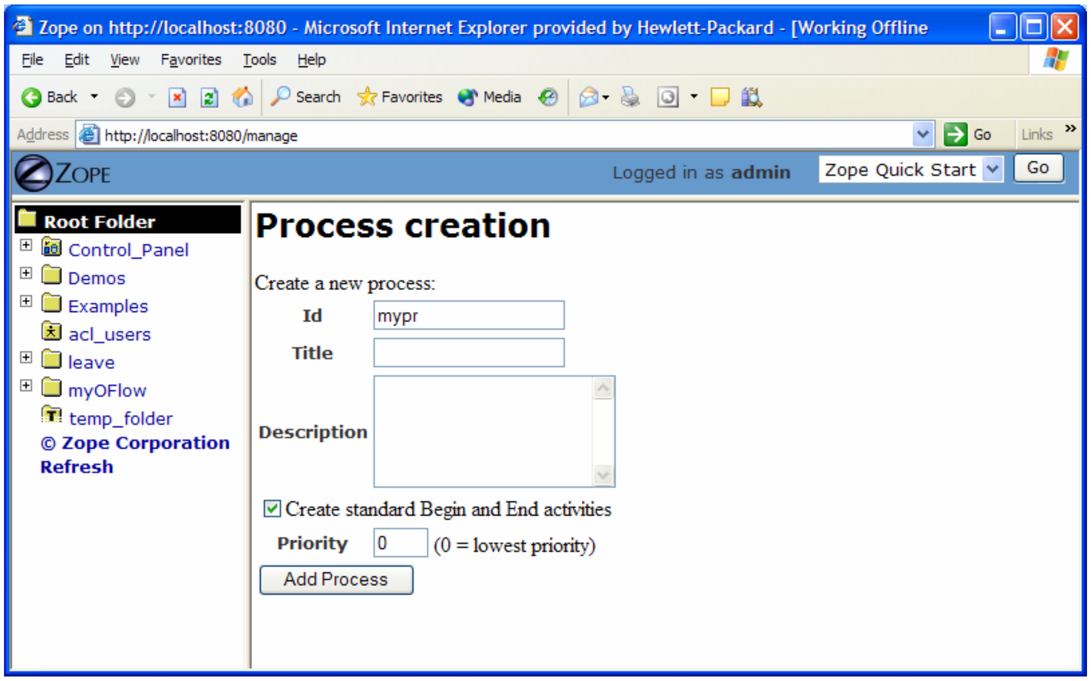
Next it is necessary to define the process and the activities pertaining to the process, together with their transitions (From Activity and To Activity). These operations are performed by accessing the tabs in the Openflow container as shown in the following screen-shots:



OpenFlow at /myOFlow/miaprova/myOF

Process	Activities	Transitions
myprocess	3	2
altroProcess	2	0

Figure 3: The process definition tab


Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packard - [Working Offline
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/manage
Go Links
ZOPE
Logged in as admin
Zope Quick Start
Go
Root Folder
Control_Panel
Demos
Examples
acl_users
leave
myOFlow
temp_folder
© Zope Corporation
Refresh
Add Process

Process creation

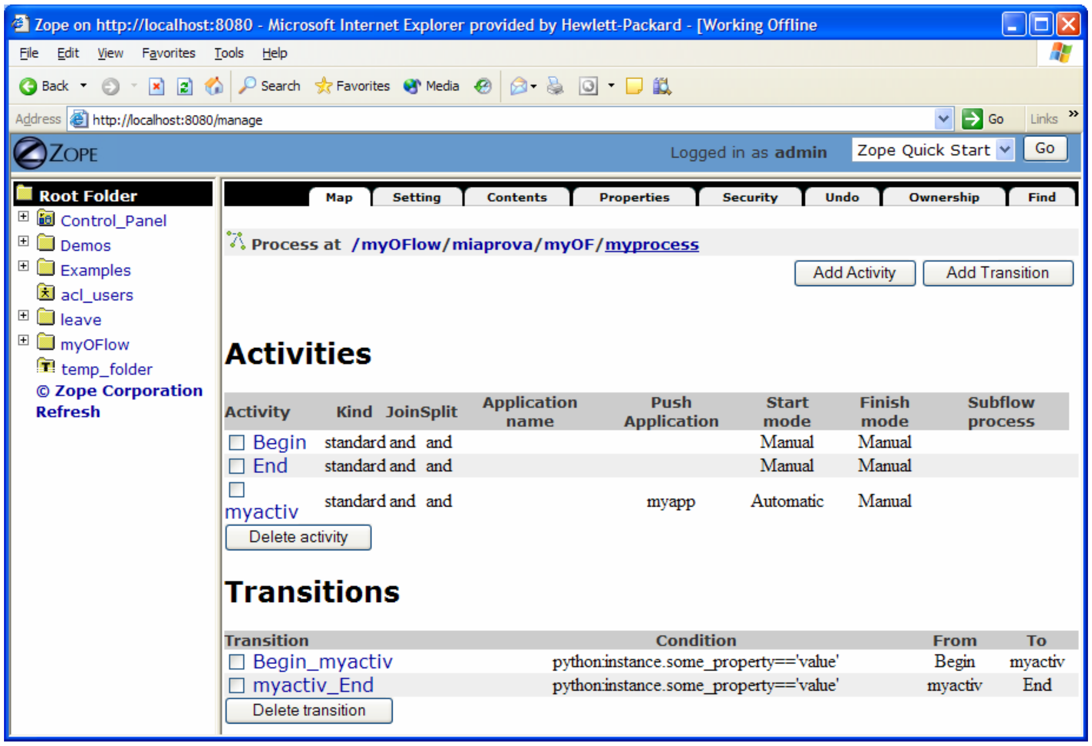
Create a new process:

Id	mypr
Title	
Description	

[✓] Create standard Begin and End activities

Priority 0 (0 = lowest priority)

Figure 4: Creating a new Process definition



Process at /myOFlow/miaprova/myOF/myprocess

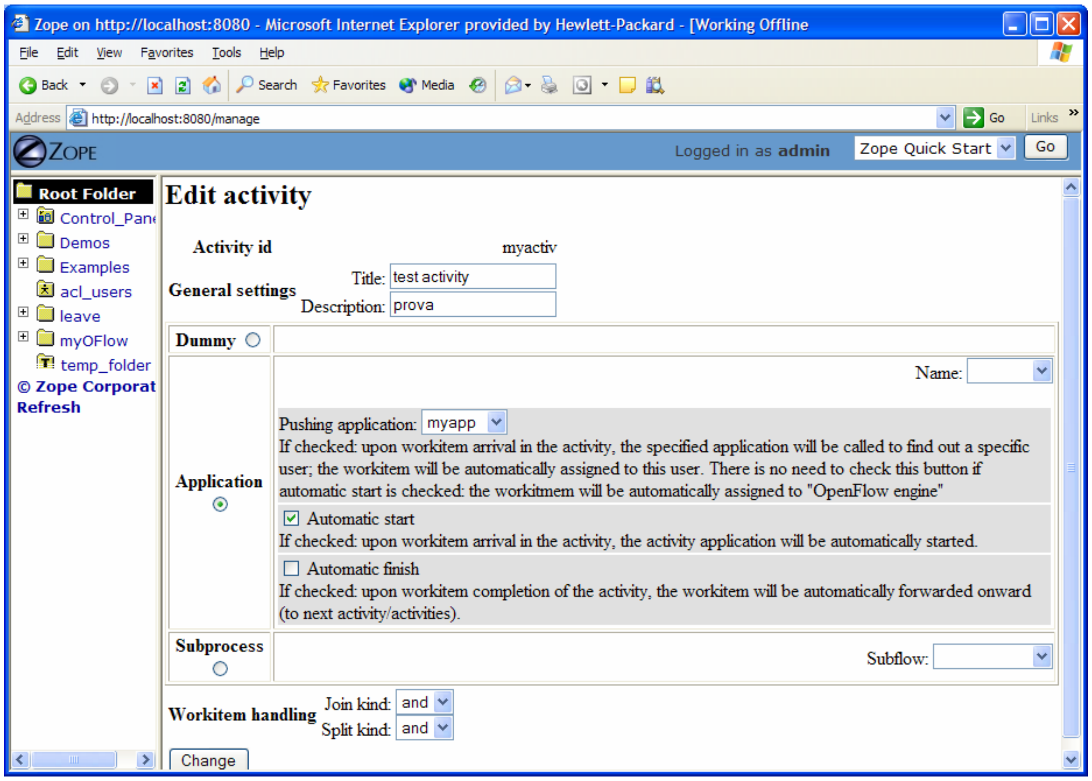
Activities

Activity	Kind	JoinSplit	Application name	Push Application	Start mode	Finish mode	Subflow process
Begin	standard	and and			Manual	Manual	
End	standard	and and			Manual	Manual	
myactiv	standard	and and		myapp	Automatic	Manual	

Transitions

Transition	Condition	From	To
Begin_myactiv	python:instance.some_property=='value'	Begin	myactiv
myactiv_End	python:instance.some_property=='value'	myactiv	End

Figure 5: Management of activity and transitions of a process


Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packard - [Working Offline
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/manage
Go Links
Logged in as admin
Zope Quick Start
Root Folder
Control_Panel
Demos
Examples
acl_users
leave
myOFlow
temp_folder
© Zope Corporat
Refresh

Edit activity

Activity id myactiv

General settings Title: test activity
Description: prova

Dummy ○

Application ⦿ Name:

Pushing application: myapp
If checked: upon workitem arrival in the activity, the specified application will be called to find out a specific user; the workitem will be automatically assigned to this user. There is no need to check this button if automatic start is checked: the workitmem will be automatically assigned to "OpenFlow engine"

[✓] Automatic start
If checked: upon workitem arrival in the activity, the activity application will be automatically started.

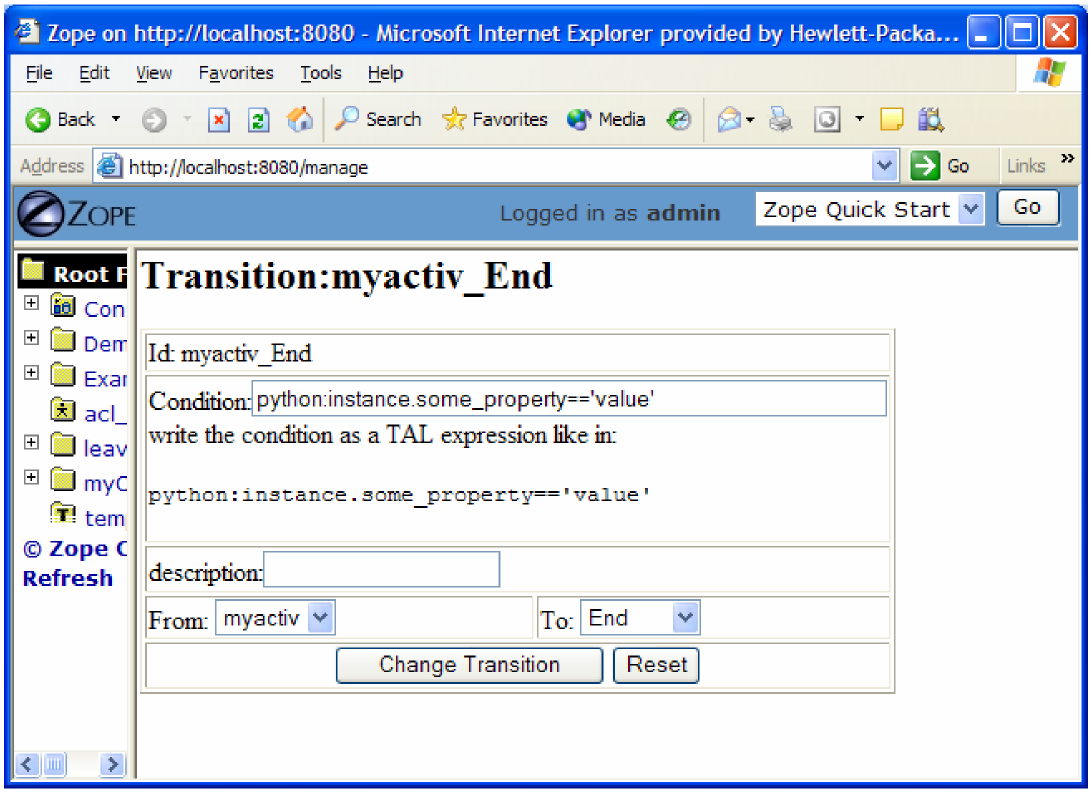
[ ] Automatic finish
If checked: upon workitem completion of the activity, the workitem will be automatically forwarded onward (to next activity/activities).

Subprocess ○ Subflow:

Workitem handling Join kind: and
Split kind: and

Change

Figure 6: Editing a process activity


Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packa...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/manage
Go Links
Logged in as admin
Zope Quick Start
Root F
Con
Dem
Exar
acl_
leav
myC
tem
© Zope C
Refresh

Transition:myactiv_End

Id: myactiv_End

Condition: python:instance.some_property=='value'

write the condition as a TAL expression like in:

```
python:instance.some_property=='value'
```

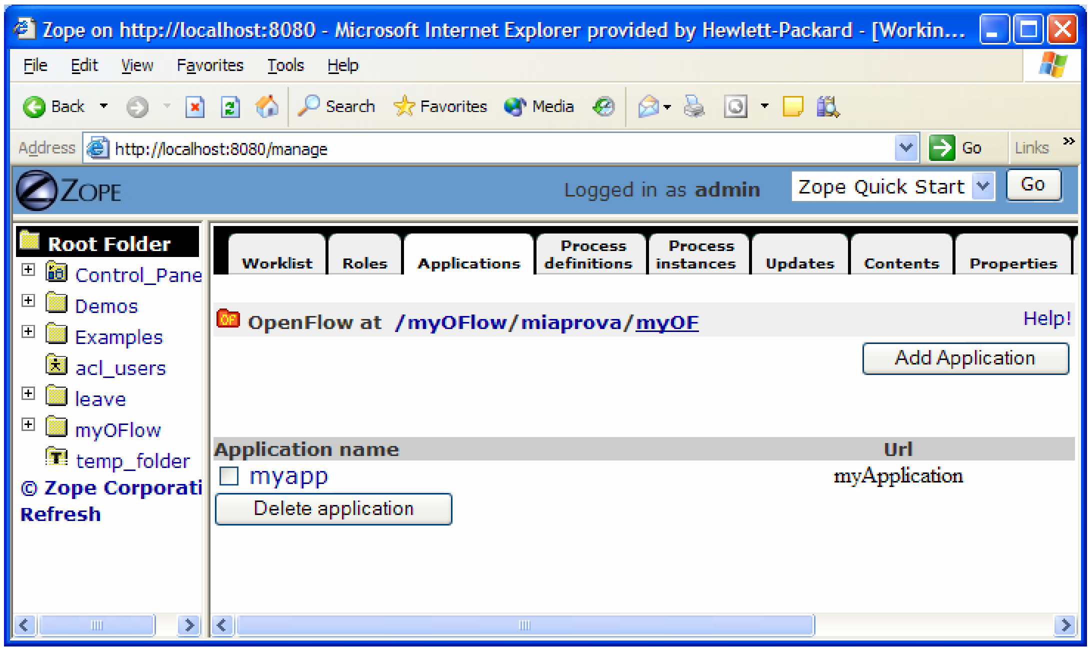
description:

From: myactiv To: End

Change Transition Reset

Figure 7: Defining process transition and related conditions

Applications associated to the activities are then specified selecting the Applications Tab.



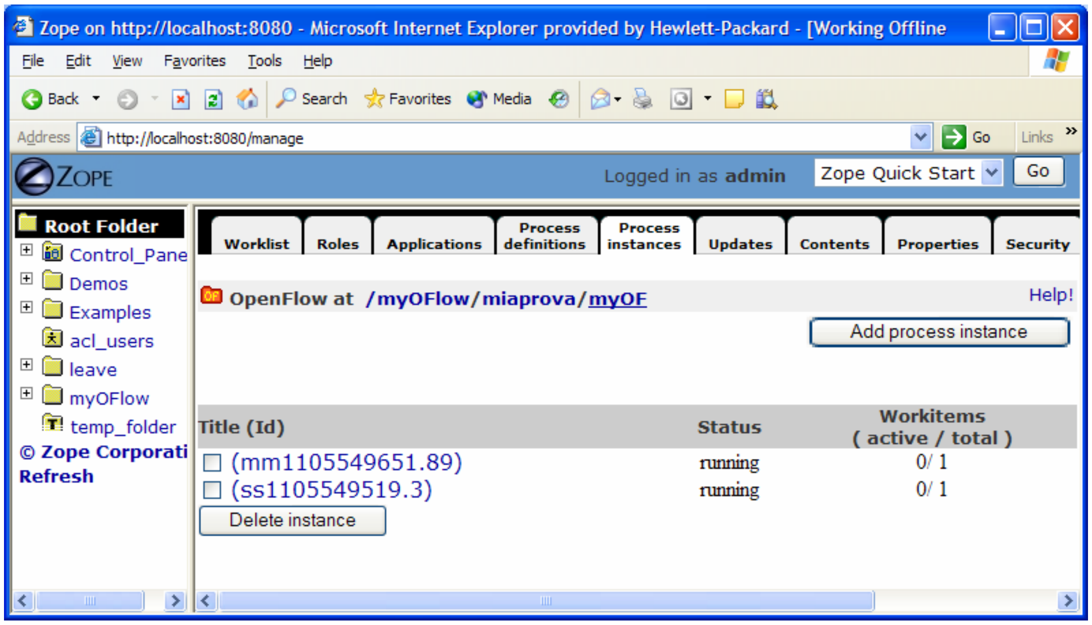
OpenFlow at /myOFlow/miaprova/myOF

Application name	Url
myapp	myApplication

Figure 8: Defining process applications

The users and roles are configured as Zope users and roles as access control list (acl_users).

Once a process has been defined it can be tested. An instance of the process can be created and executed directly in the processflow-instance management tab shown below.

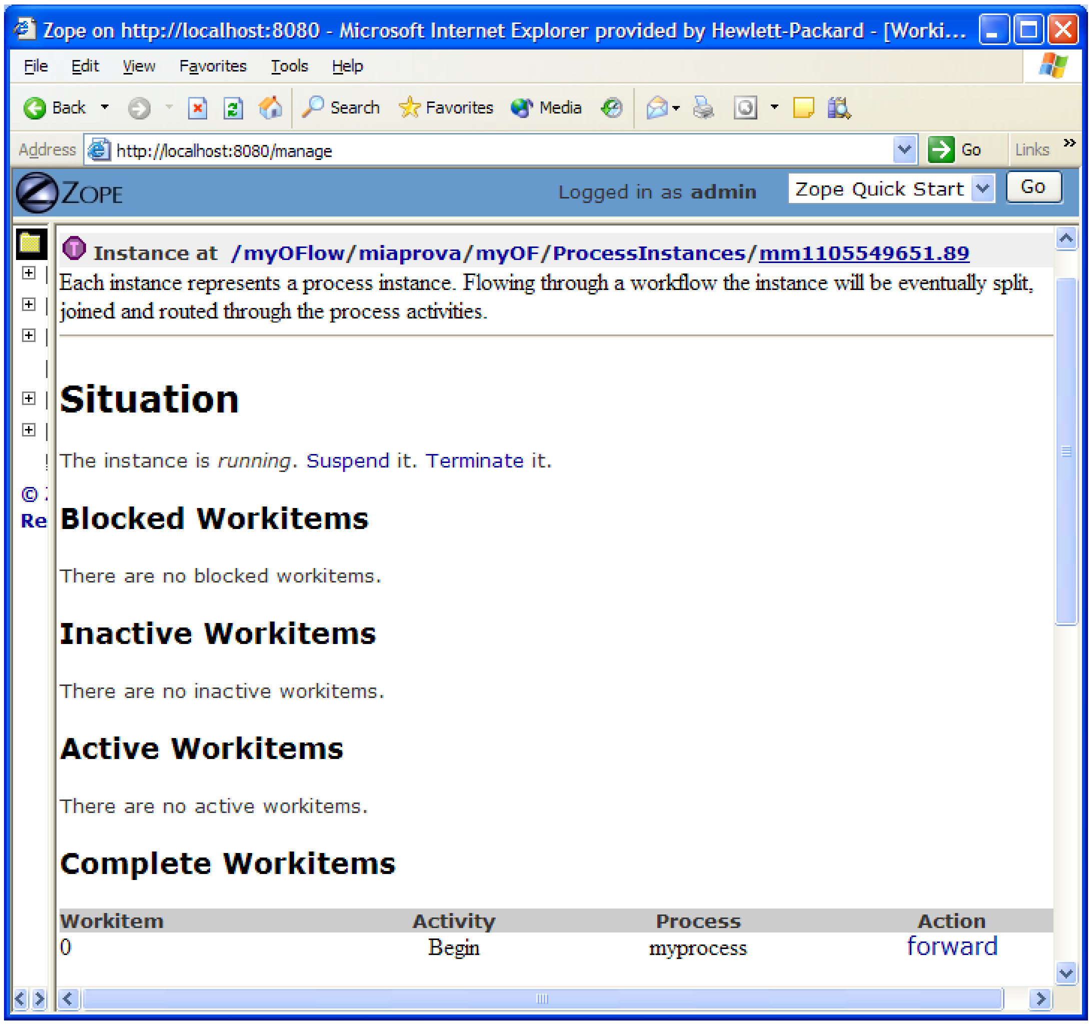


OpenFlow at /myOFlow/miaprova/myOF

Title (Id)	Status	Workitems (active / total)
(mm1105549651.89)	running	0/ 1
(ss1105549519.3)	running	0/ 1

Figure 9: Process instance management tab

The following Figure shows the of the workitems involved in the process instance that has been created.


Zope on http://localhost:8080 - Microsoft Internet Explorer provided by Hewlett-Packard - [Worki...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/manage Go Links
ZOPE Logged in as admin Zope Quick Start Go

Instance at /myOFlow/miaprova/myOF/ProcessInstances/mm1105549651.89

Each instance represents a process instance. Flowing through a workflow the instance will be eventually split, joined and routed through the process activities.

Situation

The instance is *running*. Suspend it. Terminate it.

Blocked Workitems

There are no blocked workitems.

Inactive Workitems

There are no inactive workitems.

Active Workitems

There are no active workitems.

Complete Workitems

Workitem	Activity	Process	Action
0	Begin	myprocess	forward

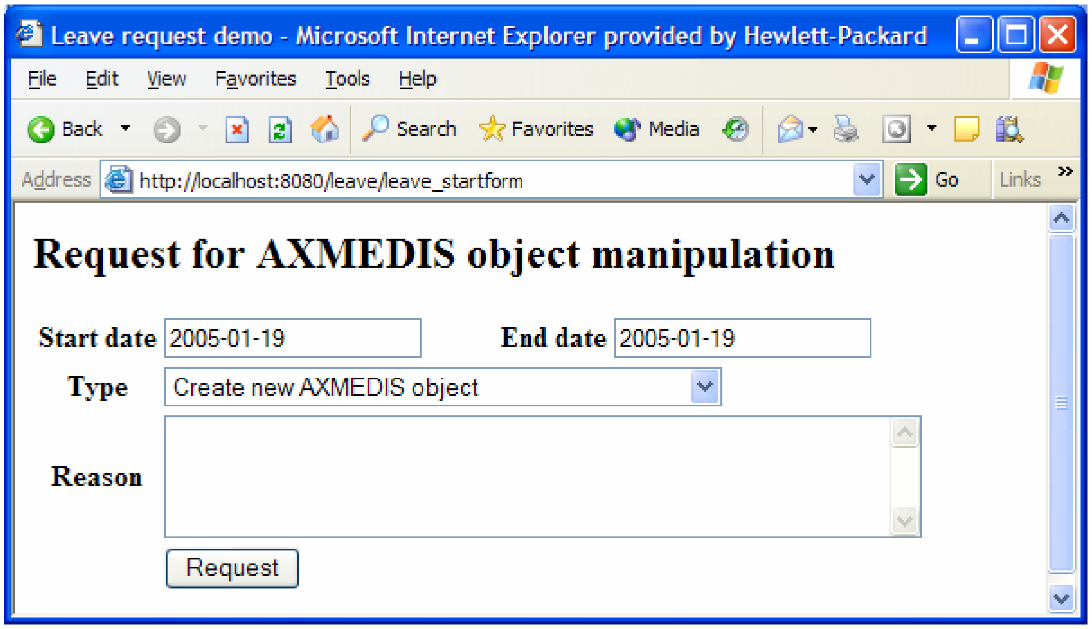
Figure 10: Monitoring and management of a specific process instance

Process Example:

The following simple example illustrates a process to request a AXMEDIS object manipulation (*a mock-up process*). This is an example of explicit forwarding to different actors having different roles. The first actor requests the creation of a new AXMEDIS object by filling out a form. The request goes to the second actor (called Socius) who checks that the request is acceptable. The request is then forwarded to the third actor (called Prefectus) for approval.

The following steps are necessary for the above example process to be enacted:

The first actor (called Tertius) enters an AXMEDIS object manipulation request by filling out the following form as shown in the screen-shot below:

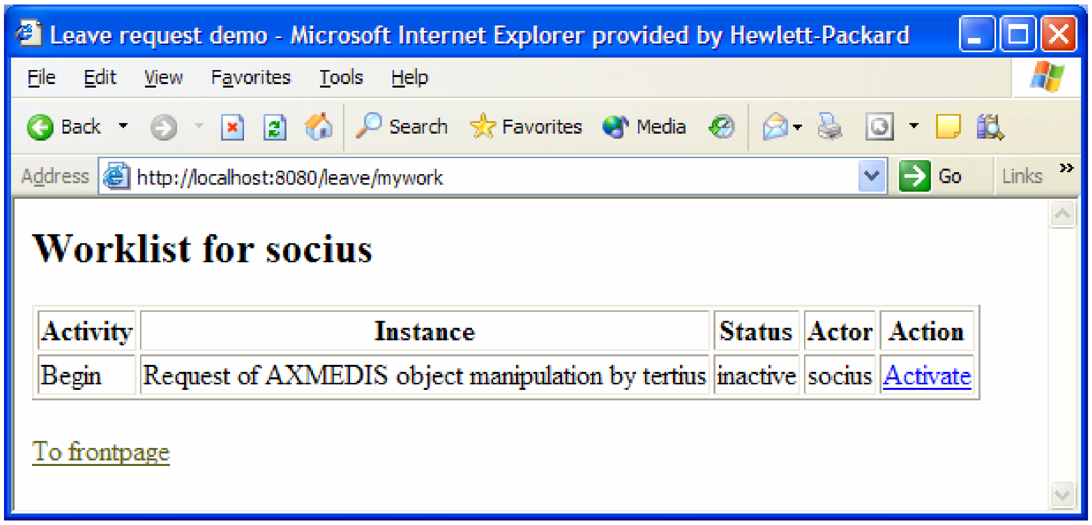

Leave request demo - Microsoft Internet Explorer provided by Hewlett-Packard
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/leave/leave_startform Go Links
Request

Request for AXMEDIS object manipulation

Start date	2005-01-19	End date	2005-01-19
Type	Create new AXMEDIS object		
Reason			

Figure 11: Tertius' AXMEDIS object manipulation form

According to the processflow, the request goes to the next actor (called Socius). When Socius logs in, his work list shows that there is a workitem in his worklist as shown in the screen-shot below:

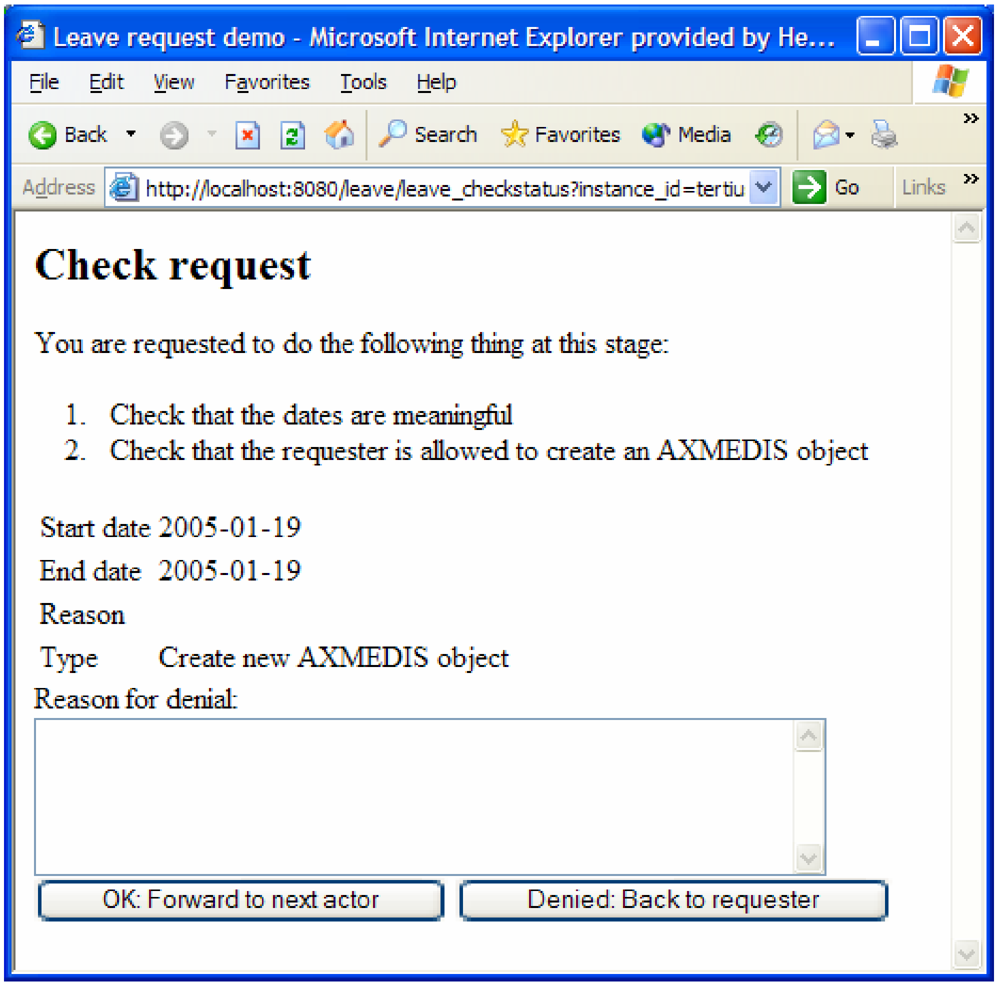

Leave request demo - Microsoft Internet Explorer provided by Hewlett-Packard
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/leave/mywork Go Links
To frontpage

Worklist for socius

Activity	Instance	Status	Actor	Action
Begin	Request of AXMEDIS object manipulation by tertius	inactive	socius	Activate

Figure 12: Socius' worklist and workitem activation

To execute the workitem, the actor (Socius) has to activate the workitem (Begin) and perform the related activities. Next this actor either forwards the workitem to the next actor, which in this case is the supervisor (called Prefectus), or rejects the request; as illustrated by the screen-shot below:


Leave request demo - Microsoft Internet Explorer provided by He...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/leave/leave_checkstatus?instance_id=tertiu
Go Links

Check request

You are requested to do the following thing at this stage:

1. Check that the dates are meaningful
2. Check that the requester is allowed to create an AXMEDIS object

Start date	2005-01-19
End date	2005-01-19
Reason	
Type	Create new AXMEDIS object

Reason for denial:

OK: Forward to next actor
Denied: Back to requester

Figure 13: Socius' workitem execution and forwarding

Then the activity is forwarded to the last actor and the process ends.

3 AXMEDIS Content Tool Error Manager (DSI)

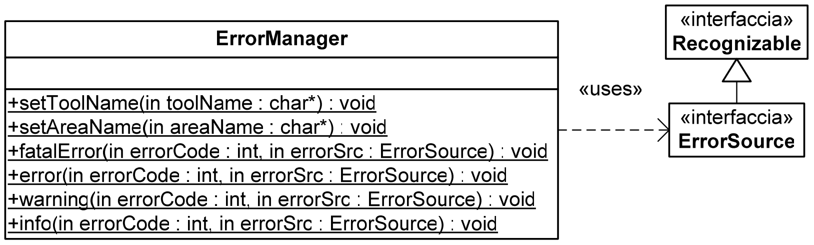
Module Profile		
AXMEDIS Content Tool Error Manager		
Executable or Library(Support)	Library	
Single Thread or Multithread	Single	
Language of Development	C++	
Responsible Name	Bellini	
Responsible Partner	DSI	
Status (proposed/approved)		
Platforms supported	All	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
XML based for error table		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
WXWidget for list of errors		
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
WxWIDGET		
log4cxx		Apache License version 2.0

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).

3.1 Class Hierarchy


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 function>

- **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.

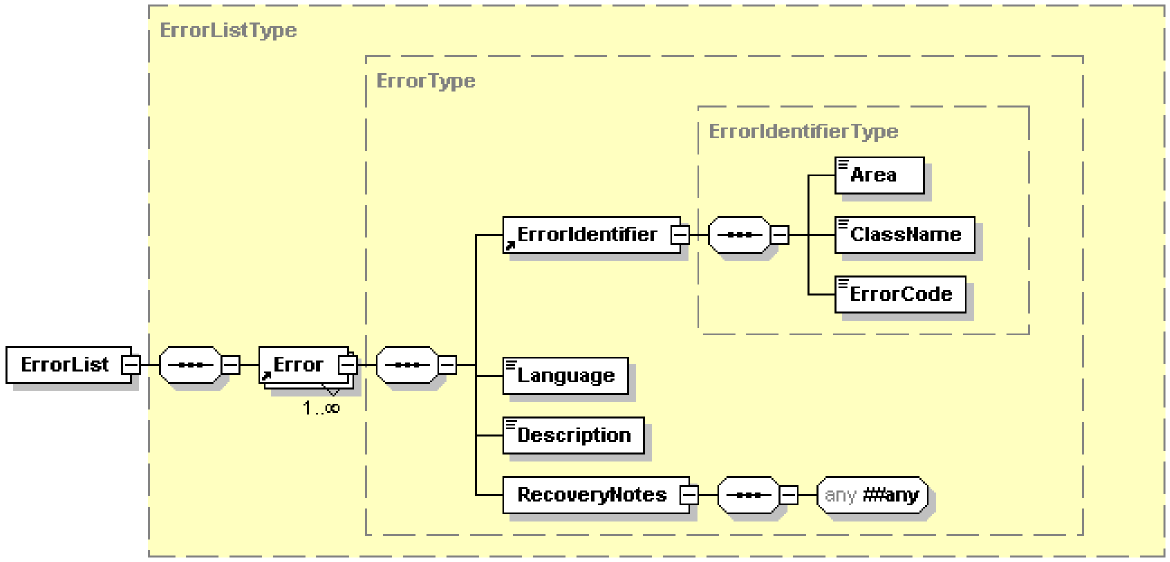
As depicted in the figure above, **ErrorSource** is an interface extending the interface Recognizable thus **ErrorManager** is able to retrieve information on the class without boring the programmer with to much code.

3.2 AXMEDIS Error Coding Format

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

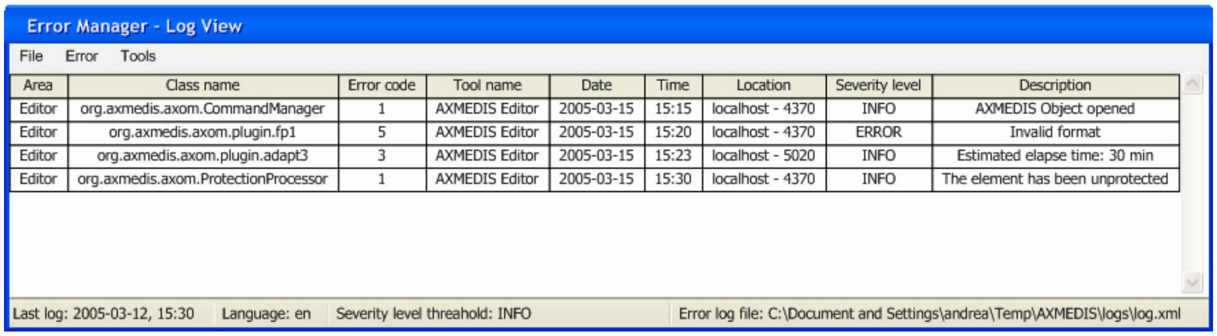
- **Error Identifier** – area name, full class name, error code
- **Language** – the language used for description and recovery notes
- **Error Description** – the error description in a specific language
- **Recovery notes** – a language dependant description which describe in more details the error 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>
```

3.3 AXMEDIS Error Manager User Interface (DSI)



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:

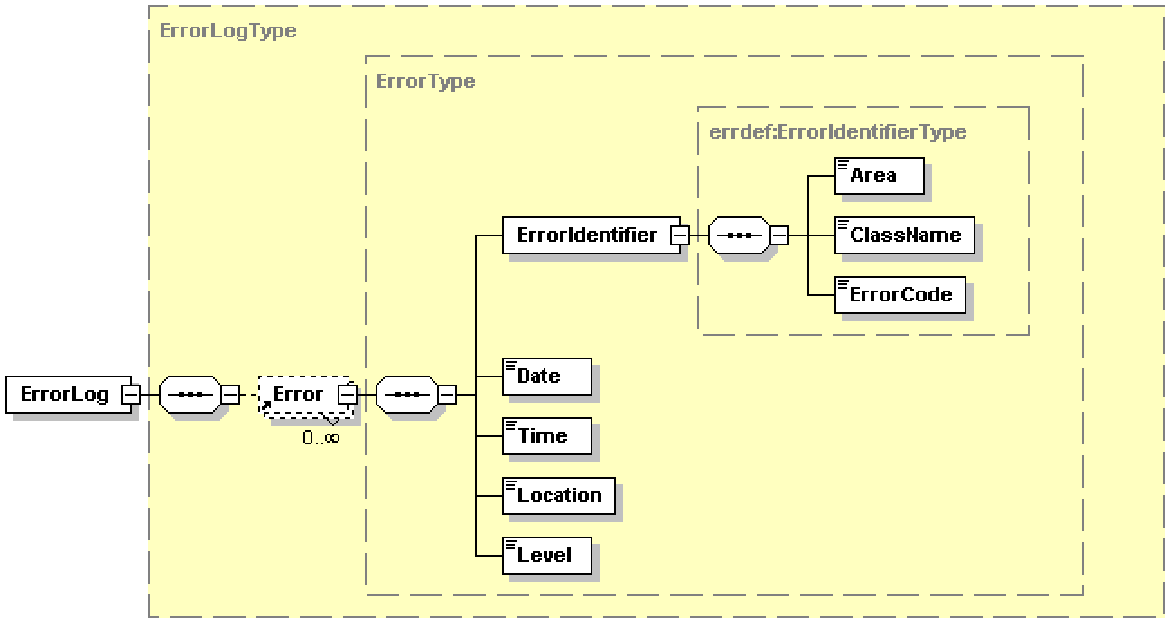
- Menu *File* – it contains file-related actions:
    - *Load* – loads an error log file and visualize it
    - *Save* – saves the visualized logs on a given location
- Menu *Error* – it contains error-related actions:
    - *Flush* – deletes all the logs in the opened error log file
    - *Delete* – 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 redirection
- *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.

3.4 AXMEDIS Error Manager Log Format (DSI)

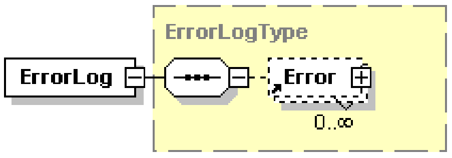
Every time an error is raised through the error manager interface the error is logged somewhere. The location of the log file can be configured using the AXMEDIS Configuration Manager. The log file is in XML format and has the following schema:



```
<?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**

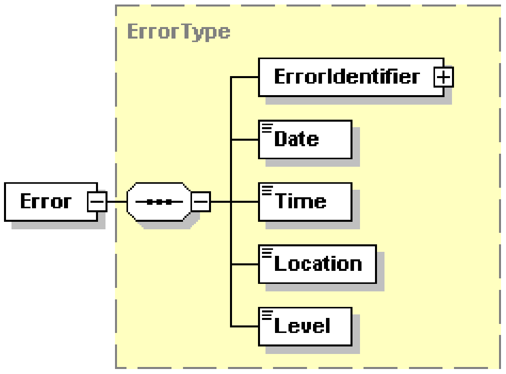
children **Error**

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

4 AXMEDIS Editor Configuration Manager (DSI, EPFL)

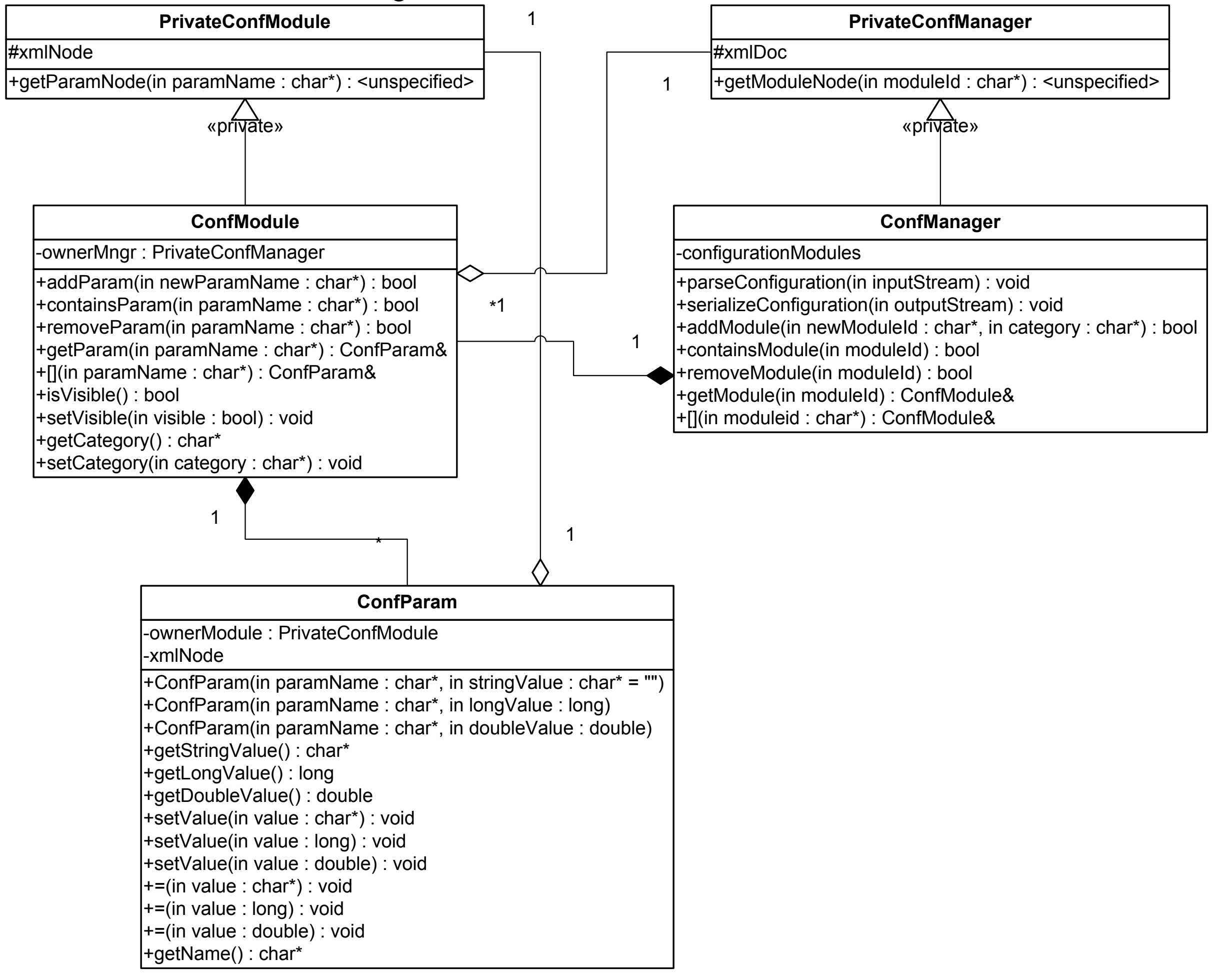
Module Profile		
AXMEDIS Editor Configuration Manageer		
Executable or Library(Support)	Library	
Single Thread or Multithread	Single	
Language of Development	C++	
Responsible Name	Andrea Vallotti	
Responsible Partner	DSI	
Status (proposed/approved)		
Platforms supported		
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File 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	
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		Open

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

4.1 AXMEDIS Editor Configuration Manager 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.


PrivateConfModule
#xmlNode
+getParamNode(in paramName : char*) : <unspecified>
PrivateConfManager
#xmlDoc
+getModuleNode(in moduleId : char*) : <unspecified>
«private»
«private»
ConfModule
-ownerMngr : PrivateConfManager
+addParam(in newParamName : char*) : bool
+containsParam(in paramName : char*) : bool
+removeParam(in paramName : char*) : bool
+getParam(in paramName : char*) : ConfParam&
+[](in paramName : char*) : ConfParam&
+isVisible() : bool
+setVisible(in visible : bool) : void
+getCategory() : char*
+setCategory(in category : char*) : void
ConfManager
-configurationModules
+parseConfiguration(in inputStream) : void
+serializeConfiguration(in outputStream) : void
+addModule(in newModuleId : char*, in category : char*) : bool
+containsModule(in moduleId) : bool
+removeModule(in moduleId) : bool
+getModule(in moduleId) : ConfModule&
+[](in moduleid : char*) : ConfModule&
ConfParam
-ownerModule : PrivateConfModule
-xmlNode
+ConfParam(in paramName : char*, in stringValue : char* = "")
+ConfParam(in paramName : char*, in longValue : long)
+ConfParam(in paramName : char*, in doubleValue : double)
+getStringValue() : char*
+getLongValue() : long
+getDoubleValue() : double
+setValue(in value : char*) : void
+setValue(in value : long) : void
+setValue(in value : double) : void
+=(in value : char*) : void
+=(in value : long) : void
+=(in value : double) : void
+getName() : char*
1
1
*1
1
1
*
1

The Configuration Manager is composed of three main classes:

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

PrivateConfManager and **PrivateConfModule** are two utility classes which are used to expose different interfaces inside and outside this namespace.

ConfManager exposes the following methods to allow management of a set of configurations and parameters:

- *parseConfiguration* – parses and loads a set of configurations from an input stream. The stream have to respect the format described in the following sub-section;
- *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. Moreover, the instance of

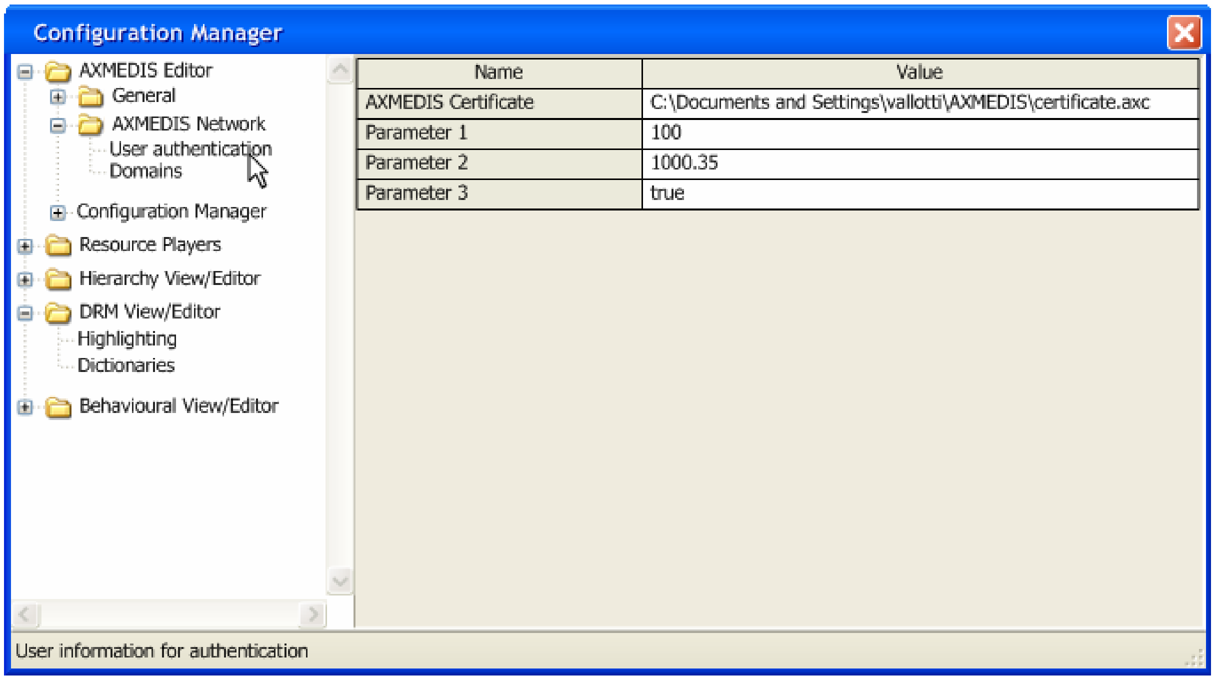
ConfManager sets itself as owner manager (*ownerMngr*) of the newly created instance of **ConfModule**. In this way, the module can interact with the configuration manager;
- *containsModule* – checks if the configuration manager contains a module with the given name (*moduleId*);
- *removeModule* – if exists, removes the module with the given name from the configuration manager;
- *getModule* and *[]* operator – both functions allow to get an instance of **ConfModule** which represent the module having the given name (*moduleId*). Acting on the obtained instance of **ConfModule**, it is possible to manage the parameters contained in the corresponding module of the configuration manager.

Moreover, **ConfManager** privately extends **PrivateConfManager** to provide the *getModuleNode* function only to the instances of **ConfModule** created by it. *getModuleNode* returns the XML node representing the module identified by *moduleId*.
ConfModule exposes the following methods to allow management of a module of parameters contained in the owner configuration manager:
- *addParam* – 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. Moreover, the instance of **ConfModule** sets itself as owner module (*ownerModule*) of the newly created instance of **ConfParam**. In this way, the parameter can interact with the owner module;
- *containsParam* – checks if the module contains a parameter with the given name (*paramName*);
- *removeParam* – if exists, removes the parameter with the given name from the module;
- *getParam* and *[]* operator – both functions allow to get an instance of **ConfParam** which represent the parameter having the given name (*paramName*). Acting on the obtained instance of **ConfParam**, it is possible to manage the parameter contained in the owner module.

ConfModule implements a mechanism similar to those used by **ConfManager**. It privately extends **PrivateConfModule** to provide the *getParamNode* function only to the instances of **ConfParam** created by it. The method returns the XML node representing the parameter having the given name (*paramName*).
ConfParam 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 of the parameter itself.

4.2 AXMEDIS Editor Configuration Manager User Interface

- 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;



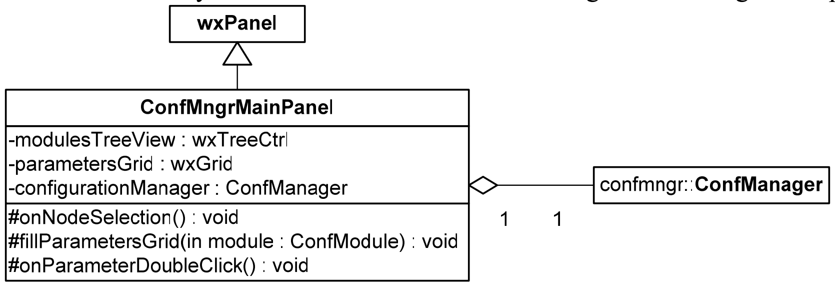
Name	Value
AXMEDIS Certificate	C:\Documents and Settings\vallotti\AXMEDIS\certificate.axc
Parameter 1	100
Parameter 2	1000.35
Parameter 3	true

User information for authentication

In the above figure, the user interface of AXMEDIS Configuration Manager is depicted. It is composed of a tree view, which organizes the modules in user friendly manner, and a generic panel where the parameters of the selected module are displayed. The showed parameters should have an adequate representation to their type, e.g. in the figure above the selected files in the “AXMEDIS Certificate” field.

4.2.1 User interface class hierarchy

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


wxPanel
ConfMngrMainPanel
-modulesTreeView : wxTreeCtrl
-parametersGrid : wxGrid
-configurationManager : ConfManager
#onNodeSelection() : void
#fillParametersGrid(in module : ConfModule) : void
#onParameterDoubleClick() : void
confmngr::ConfManager
1
1

4.3 AXMEDIS Configuration Format (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.

```
<config>
   <module name="User authentication" category="AXMEDIS Editor/AXMEDIS Network">
      <parameter name="AXMEDIS Certificate" type="String">file:/// C:/Documents and
Settings/vallotti/AXMEDIS/certificate.axc</parameter>
      <parameter name="Parameter 1" type="Int32">100</parameter>
      <parameter name="Parameter 2" type="Float">1000.35</parameter>
      <parameter name="Parameter 3" type="Boolean">true</parameter>
   </module>
   <module ...>
      <parameter ...> ... </parameter>
      <parameter ...> ... </parameter>
      <parameter ...> ... </parameter>
   </module>
   .
   .
   .
   .
</config>
```

In the previous example the module and parameters showed in the figure above (the one related to the user interface) are expressed in the needed format. The related schema is reported below.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
<xs:element name="config">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="module">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="parameter">
                     <complexType>
                        <sequence>
                           <any namespace="##any" processContents="lax" minOccurs="0"/>
                        </sequence>
                        <attribute name=”name” type=”string”/>
                        <attribute name=”type” type=”string”/>
                     </complexType>
                  </xs:element>
               </xs:sequence>
               <attribute name=”name” type=”string”/>
               <attribute name=”category” type=”string”/>
            </xs:complexType>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
```

5 AXMEDIS Editor Plug-in Manager (DSI, EPFL)

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 manage the following kind of plug-ins:
  - Data-manipulation plug-ins shall be able to modify AXMEDIS object structure, i.e. plug-ins which shall be able to delete or move existing components, insert new components, etc…
  - Metadata show/manage plug-ins shall be used by Metadata View to adequately display and modify user-defined sets of metadata;
  - Metadata production shall be able, through AXMEDIS object (and parts thereof) analysis, to produce metadata to be included into the object;
  - Configuration plug-ins shall be used by AXMEDIS Editor Configuration Manager to manage and display specific configuration information;
  - Workflow plug-ins which shall permit interaction of AXMEDIS Editor with AXMEDIS Workflow subsystem;
  - Protection plug-ins, which contain protection algorithm enriching the set of those available for the Protection Processor;
- Plug-in Manger shall provide standard interface definition for the above mentioned plug-ins family;
- 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:
  - Kinds of component or actions associated;
  - Mime type association;
  - Etc…

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

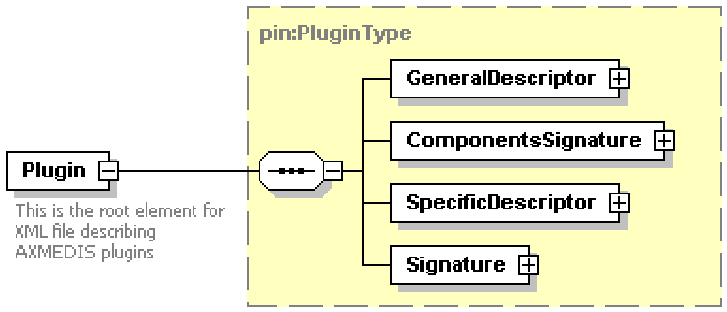
See the XML schema below for more details.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by Paolo Nesi (University of Florence) -->
<xs:schema targetNamespace="http://www.axmedis.org/plugin-schema" xmlns:pin="http://www.axmedis.org/plugin-schema"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" xmlns:xs="http://www.w3.org/2001/XMLSchema"
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 name="GeneralDescriptor" type="pin:GeneralDescriptorType"/>
            <xs:element name="ComponentsSignature" type="dsig:SignatureType"/>
            <xs:element name="SpecificDescriptor" type="pin:SpecificDescriptorType"/>
            <xs:element name="Signature" type="dsig:SignatureType"/>
        </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"/>
        <xs:complexType name="SpecificDescriptorType" abstract="true">
            <xs:sequence>
                <xs:any namespace="##any"/>
            </xs:sequence>
        </xs:complexType>
        <xs:element name="Signature" type="dsig:SignatureType"/>
</xs:schema>
```

element **Plugin**

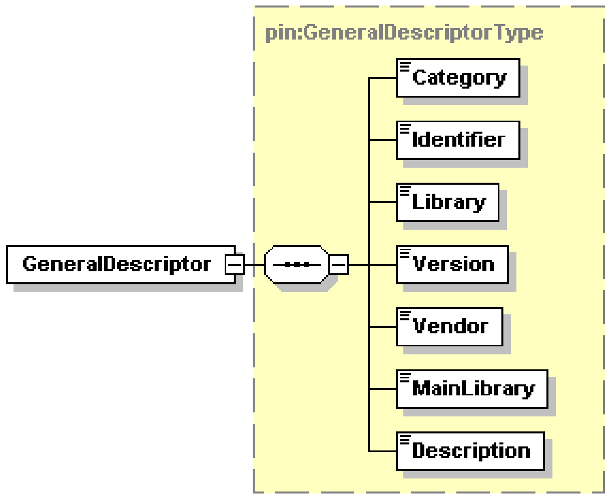
diagram



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



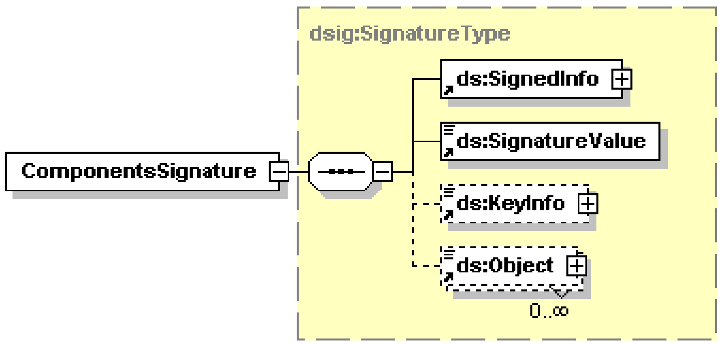
namespace	http://www.axmedis.org/plugin-schema

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**

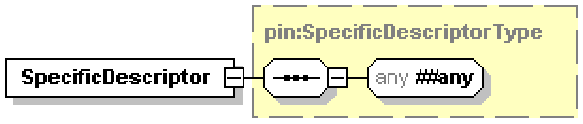
attributes

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



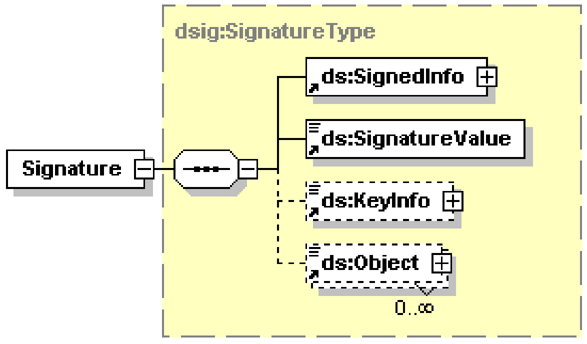
namespace http://www.axmedis.org/plugin-schema

type **pin:SpecificDescriptorType**

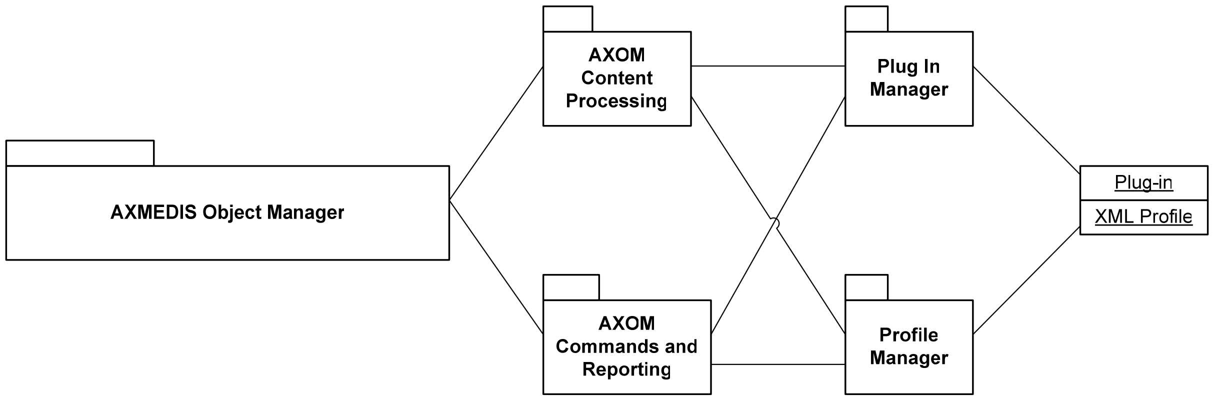
description This is an abstract element which can be substituted with any element derived by **SpecificDescriptorType**. In that way, **Category**-dependant XML schema can be used to describe specific features of the plug-in (see **GeneralDescriptor**).

[DSI] Use of **SpecificDescriptor** as described above is still under discussion. Another proposed solution is to create a unique XML schema which can be capable to acceptably describe all kinds of plug-in. That new solution will allow to use a unique formalism to describe plug-ins and a standard way to use them.

element **Signature**

diagram	
namespace	http://www.axmedis.org/plugin-schema
type	dsig:SignatureType
children	dsig:SignedInfo dsig:SignatureValue dsig:KeyInfo dsig:Object
attributes	Name: Id; Type: ID; Use: optional; Default: ; Fixed:
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.

Notice that all references contained in a profile have to be evaluated as relative to the profile location on the local file-system. This should not be a hard constraint for plug-in developers because AXMEDIS plug-ins will be installed in a given location (which depends on the platform).
Profile authenticity and plug-in signature are not directly verified by the Plug-in Manager, they are verified by the Protection Processor calling *verifySoftware* and passing the location of the profile and the profile itself.



5.1 Plug-in function description

The functions contained in a plug-in, which should be made available to the AXOM, have to be someway described in the plug-in profile. In that way, AXOM knows the signature of that functions and can call them. The functions described in the plug-in profile are accessible in different ways/for different purposes including:

- a user can use a function through a user interface like the AXMEDIS Editor; e.g. a user wants to use a specific function on a given resource then the editor can display an interface dynamically generated
- automatically by an engine like Compositional/Formatting Engine; e.g. in a compositional script a fingerprint extract function can be called

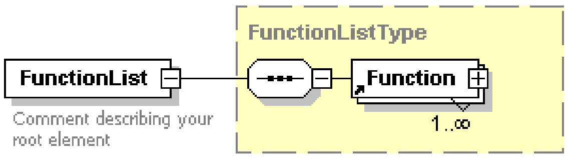
The function description has to be placed within the **SpecificDescriptor** element of the plug-in description schema.
See the XML below for more details

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.axmedis.org/plugin-function-schema"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.axmedis.org/plugin-function-schema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="FunctionList" type="FunctionListType">
        <xs:annotation>
            <xs:documentation>Comment describing your root element</xs:documentation>
        </xs:annotation>
    </xs:element>
    <xs:complexType name="FunctionListType">
        <xs:sequence>
            <xs:element ref="Function" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Function" type="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="FunctionDescription"/>
            <xs:element ref="ParameterList"/>
            <xs:element ref="Result"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="FunctionDescription" type="DescriptionType"/>
    <xs:complexType name="DescriptionType">
        <xs:sequence>
            <xs:any namespace="##any"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ParameterList" type="ParameterListType"/>
    <xs:complexType name="ParameterListType">
        <xs:sequence>
            <xs:element ref="Param" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="Result" type="ParamType"/>
    <xs:element name="Param" type="ParamType"/>
    <xs:complexType name="ParamType">
        <xs:sequence>
            <xs:element name="Name" type="xs:ID"/>
            <xs:element ref="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="ParamDescription"/>
            <xs:element ref="Constraints" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ParamDescription" type="DescriptionType"/>
    <xs:element name="Constraints" type="ConstraintsType"/>
    <xs:complexType name="ConstraintsType">
        <xs:sequence>
            <xs:element ref="Ranges" minOccurs="0"/>
            <xs:element ref="Resource" minOccurs="0"/>
            <xs:any namespace="##any"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ParamType" type="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="CHAR"/>
                <xs:enumeration value="RESOURCE"/>
                <xs:enumeration value="AXOM"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:element name="Ranges" type="RangesType"/>
        <xs:complexType name="RangesType">
            <xs:sequence>
                <xs:element ref="Range" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
        <xs:element name="Range" type="RangeType"/>
        <xs:complexType name="RangeType">
            <xs:sequence>
                <xs:element name="From" type="Limit"/>
                <xs:element name="To" type="Limit"/>
            </xs:sequence>
        </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="ResourceType"/>
        <xs:complexType name="ResourceType">
            <xs:sequence>
                <xs:element name="Type" type="xs:string"/>
                <xs:element name="Format" type="xs:string" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:schema>
```

element **FunctionList**

diagram



namespace	http://www.axmedis.org/plugin-function-schema
type	**FunctionListType**
children	**Function**

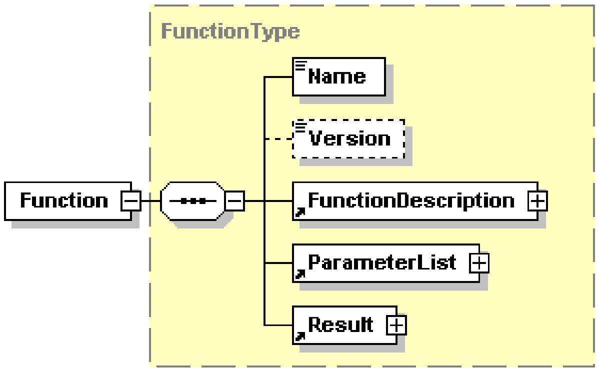
source

```
<xs:element name="FunctionList" type="FunctionListType">
  <xs:annotation>
    <xs:documentation>Comment describing your root element</xs:documentation>
  </xs:annotation>
</xs:element>
```

description This element is the root element for the description of all function exposed by a plug-in.

element **Function**

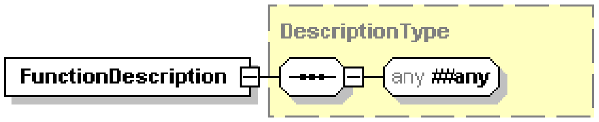
diagram



namespace	http://www.axmedis.org/plugin-function-schema
type	**FunctionType**
children	**Name** **Version** **FunctionDescription** **ParameterList** **Result**
used by	complexType **FunctionListType**
source	<xs:element name="Function" type="FunctionType"/>
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.

Element **FunctionDescription**

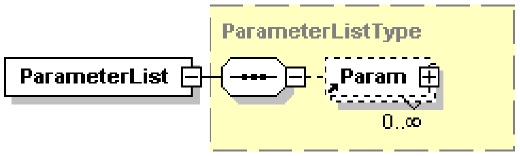
diagram



namespace	http://www.axmedis.org/plugin-function-schema
type	**DescriptionType**
used by	complexType **FunctionType**
source	<xs:element name="FunctionDescription" type="DescriptionType"/>
description	This element can contain any kind of data and represents the description of the function (e.g. the help for the function).

Element **ParameterList**

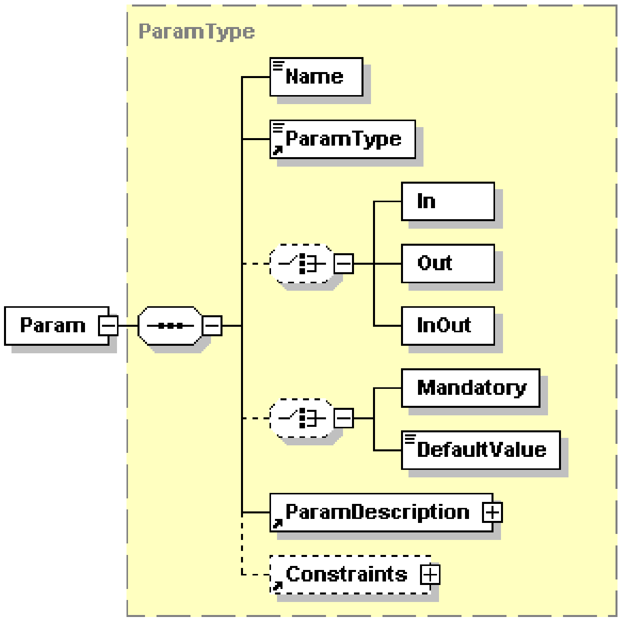
diagram



namespace	http://www.axmedis.org/plugin-function-schema
type	**ParameterListType**
children	**Param**
used by	complexType **FunctionType**
source	<xs:element name="ParameterList" type="ParameterListType"/>
description	This element represents the list of parameters to be passed to the function.

Element **Param**

diagram



namespace	http://www.axmedis.org/plugin-function-schema
type	**ParamType**
children	**Name ParamType In Out InOut Mandatory DefaultValue ParamDescription Constraints**
used by	complexType **ParameterListType**
source	<xs:element name=”Param” type=”ParamType”/>

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
- **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 obligatoriness 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
- **ParamDescription** – see below
- **Constraints** – see below

element **ParamType**

diagram

namespace	http://www.axmedis.org/plugin-function-schema	
type	**ParamTypeType**	
used by	complexType	**ParamType**
facets	enumeration	UINT16
	enumeration	INT16
	enumeration	UINT32

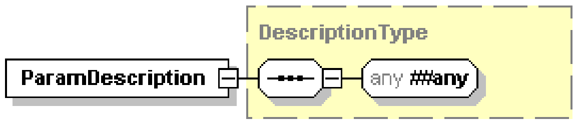
enumeration	INT32
enumeration	FLOAT
enumeration	DOUBLE
enumeration	BOOLEAN
enumeration	STRING
enumeration	CHAR
enumeration	RESOURCE
enumeration	AXOM

source <xs:element name="ParamType" type="ParamTypeType"/>

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. The association is reported below.

element **ParamDescription**

diagram



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

type **DescriptionType**

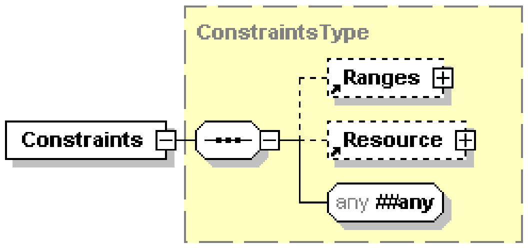
used by complexType **ParamType**

source <xs:element name="ParamDescription" type="DescriptionType"/>

description This element contains a human-readable description of a parameter, e.g. an help for the user.

Element **Constraints**

diagram



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

type **ConstraintsType**

children **Ranges** **Resource**

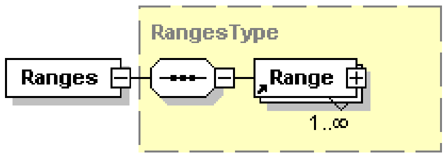
used by complexType **ParamType**

source <xs:element name="Constraints" type="ConstraintsType"/>

description This element contains the constraints which the parameter is liable to. **Constraints** can contain several kind of constraints, in the picture are reported some examples which are explained below.

element **Ranges**

diagram



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

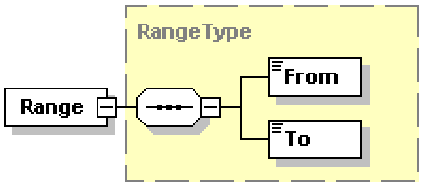
type **RangesType**

children **Range**

used by	complexType **ConstraintsType**
source	<xs:element name="Ranges" type="RangesType"/>
description	This element contains a set of ranges which are used to constrain a parameter.

element **Range**

diagram



namespace	http://www.axmedis.org/plugin-function-schema
type	**RangeType**
children	**From** **To**
used by	complexType **RangesType**
source	<xs:element name="Range" type="RangeType"/>

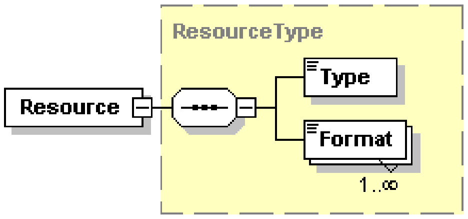
description This element represent an interval constraint of a parameter. It is composed by two values:

- **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.

element **Resource**

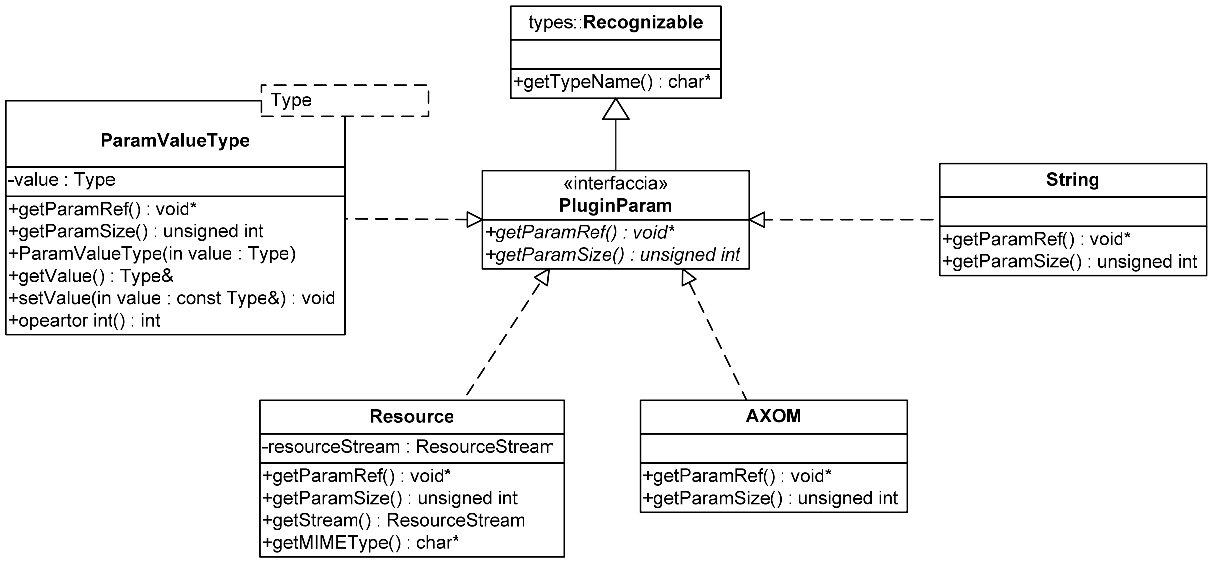
diagram



namespace	http://www.axmedis.org/plugin-function-schema
type	**ResourceType**
children	**Type** **Format**
used by	complexType **ConstraintsType**
source	<xs:element name="Resource" type="ResourceType"/>

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].

5.2 Plug-in function parameters class hierarchy

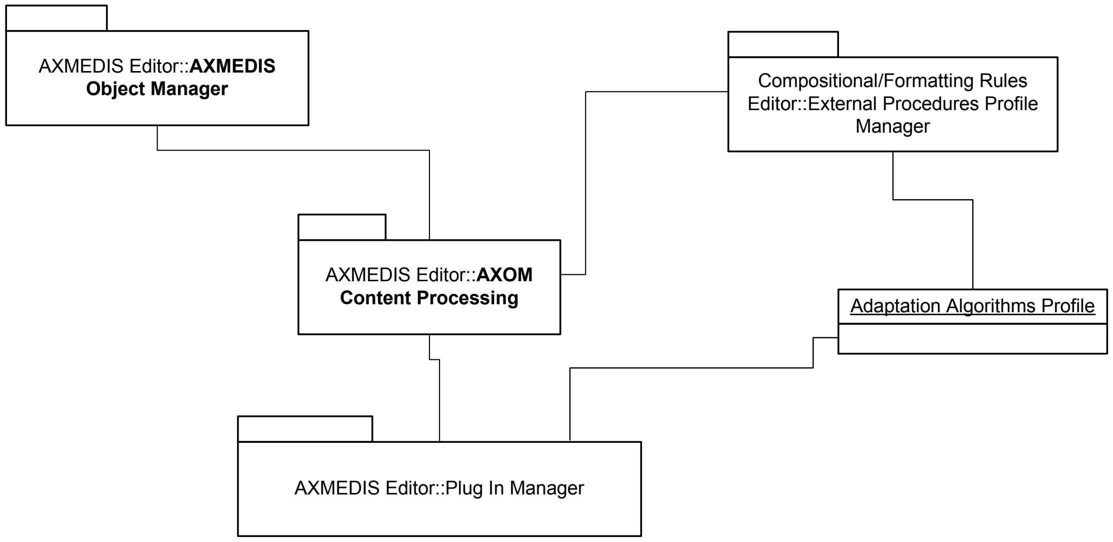


As stated in the description schema, for each parameter type defined there a specific class has to be used in the programming language. Those classes derive from a common base class (**PluginParam**) which provides a basic interface for all possible parameter types. Moreover PluginParam derives from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

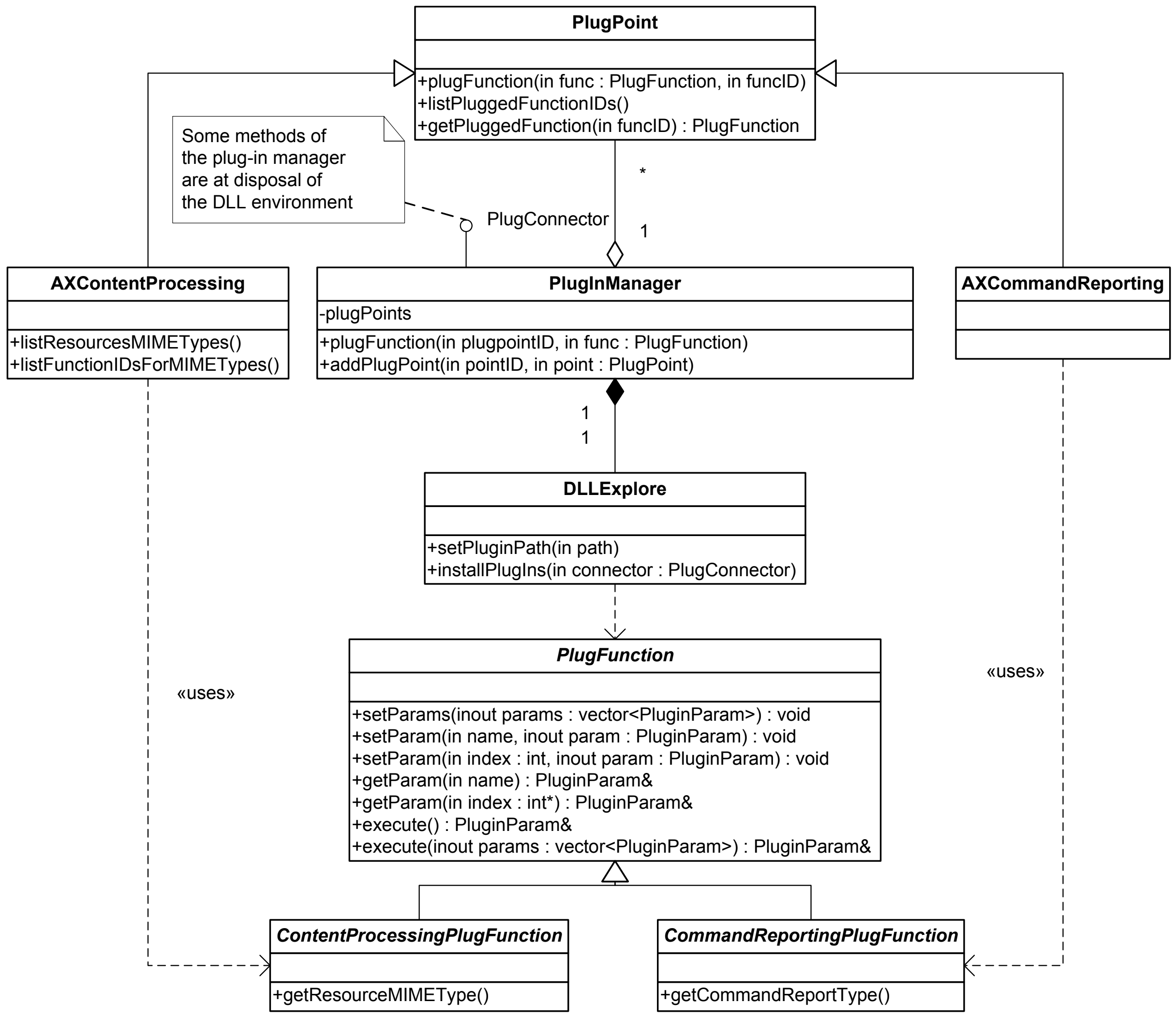
- **ParamValueType** – it is a template class which will be used, through a set of *typedef* definition, to wrap the basic data type, that is: UNIT16, INT16, UINT32, INT32, CHAR, BOOLEAN, FLOAT, DOUBLE.
  This class exposes a constructor and a cast operator which allow to freely use its instances in the same way of base type.
- **String** – it represents a parameter of type STRING
- **Resource** – it represents a RESOURCE. It exposes two meaningful functions: **getMIMEType**, **getStream**. The former allow to know the MIME type of the resource contained in it, the latter returns a stream which allow to get the resource wherever it is physically located.
- **AXOM** – it represents a parameter of type AXOM.

The base class exposes two generic function **getParamRef** and **getParamSize** which respectively return a reference to the real parameter (as void pointer) and the size in byte of the parameter itself.

5.3 Plug In Manager (general tool) (DSI, EPFL)


AXMEDIS Editor::**AXMEDIS Object Manager**
Compositional/Formatting Rules Editor::External Procedures Profile Manager
AXMEDIS Editor::**AXOM Content Processing**
Adaptation Algorithms Profile
AXMEDIS Editor::Plug In Manager

Module Profile		
Plug in Manager		
Executable or Library(Support)	Library	
Single Thread or Multithread	Single	
Language of Development	C++	
Responsible Name	Andrea Vallotti	
Responsible Partner	DSI	
Status (proposed/approved)		
Platforms supported		
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
API	AXOM, Editors and Engines	
File Formats Used	Shared with	File format name or reference to a section
none	None	
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Plug in monitor, list of active plug ins	C++	
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
WxWIDGET		


PlugPoint
+plugFunction(in func : PlugFunction, in funcID)
+listPluggedFunctionIDs()
+getPluggedFunction(in funcID) : PlugFunction
Some methods of the plug-in manager are at disposal of the DLL environment
PlugConnector
*
1
AXContentProcessing
+listResourcesMIMETypes()
+listFunctionIDsForMIMETypes()
PlugInManager
-plugPoints
+plugFunction(in plugpointID, in func : PlugFunction)
+addPlugPoint(in pointID, in point : PlugPoint)
AXCommandReporting
1
1
DLLExplore
+setPluginPath(in path)
+installPlugIns(in connector : PlugConnector)
«uses»
«uses»
PlugFunction
+setParams(inout params : vector<PluginParam>) : void
+setParam(in name, inout param : PluginParam) : void
+setParam(in index : int, inout param : PluginParam) : void
+getParam(in name) : PluginParam&
+getParam(in index : int*) : PluginParam&
+execute() : PluginParam&
+execute(inout params : vector<PluginParam>) : PluginParam&
ContentProcessingPlugFunction
+getResourceMIMEType()
CommandReportingPlugFunction
+getCommandReportType()

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:

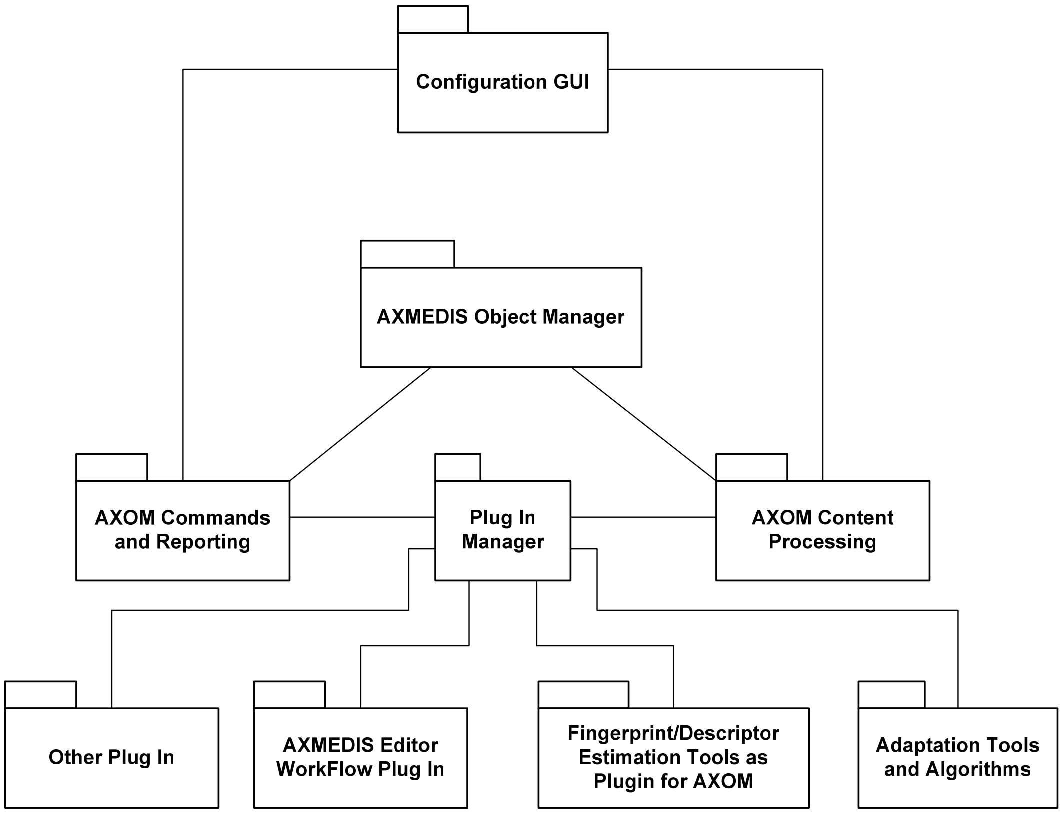
- **PluginManager** – it is the main class of the module which links the plug-in (and their functionalities) to those component which will use them. It contains a list of **PlugPoint**, each of them is registered to a specific category of plug-in and function. Every time **PluginManager** loads a plug-in and its functions, it alerts the related **PlugPoint**.
- **DLLExplorer** – it is an utility class which is used by the **PluginManager** to discover all the plug-in installed on a device. Every time it found a dynamic library with its associated profile it invokes the **PluginManager**.
- **PlugFunction** – it is a base class for all those classes which have to be used as plug-in function. The **execute** method is pure virtual and it have to be defined in the derived classes, it is the real functional part of those classes.
- **PlugPoint** – it is a base class for all those classes which want to use plug-in functions. As depicted in the figure, PlugPoint is the base class the content processing module derives from (as well as the command and reporting module)

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

Fundamentally an AXMEDIS Plug-in is a dynamic library acting as factory of plug-in functions. A plug-in have to expose the following interface:

```
extern "C"
PluginFunction* createPluginFuncton(const std::string& id);
void releasePluginFuncton (PluginFunction*);
```

5.4 AXOM Content processing (DSI, EPFL)


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

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 a subclass of **PlugPoint** and it is registered in the plug-in manager as plug point for all the above-mentioned function categories. 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 (described in the **Constraints** element of the function description).
AXOM uses AXOM Content Processing function to access function s on the base of its needs and of the requests it receives from the user.
AXOM Content Processing configuration is also based on the information provided by the user thorough the Configuration GUI. This module allows the user to visualize the association among resource types and functions and to modify those links so as achieves specific goals.

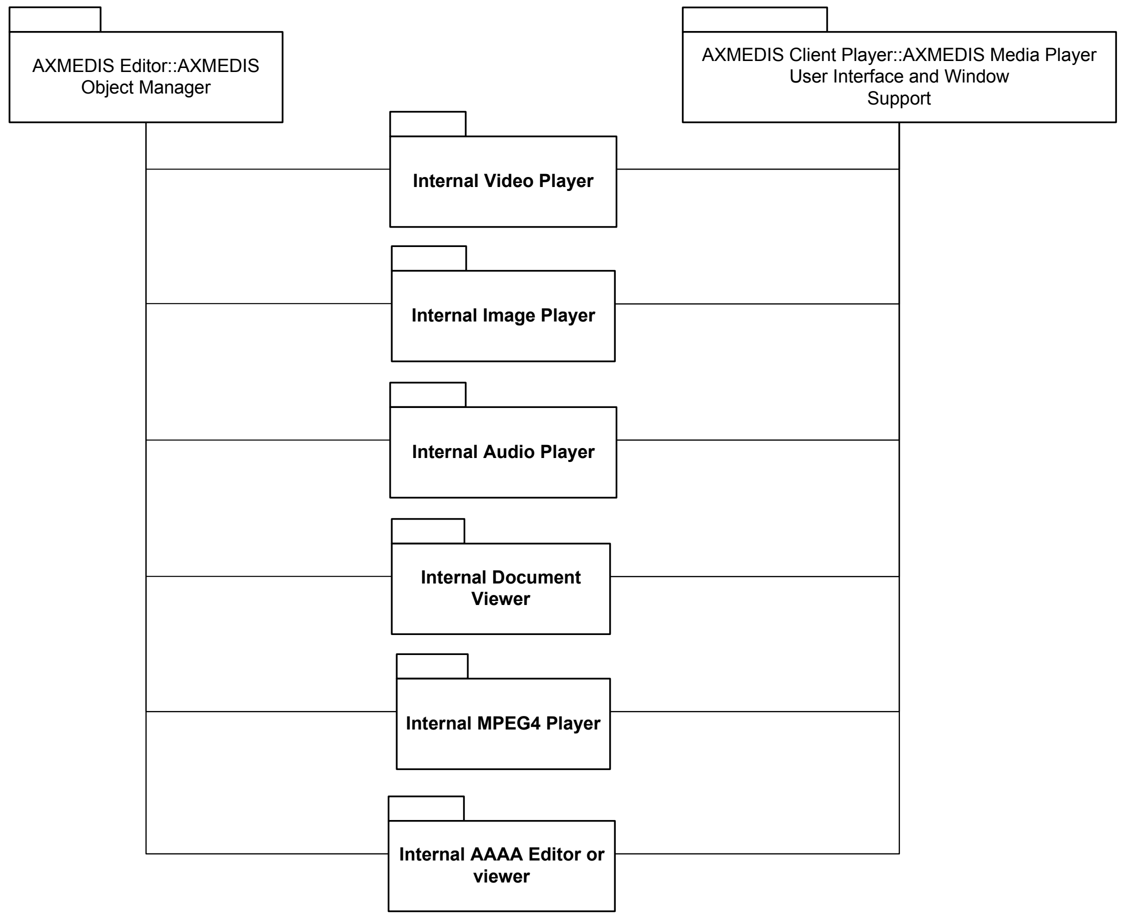
5.5 AXOM Commands and Reporting (DSI, EPFL)

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.

6 Internal AXMEDIS Resource Editors/Viewers (DSI)

a set of view/editor built-in AXMEDIS Editor which guarantees a range of basic behaviors, for example, audio player, video player, doc viewer, etc.;

Internal AXMEDIS Resource Editor/Viewer


AXMEDIS Editor::AXMEDIS Object Manager
AXMEDIS Client Player::AXMEDIS Media Player User Interface and Window Support
Internal Video Player
Internal Image Player
Internal Audio Player
Internal Document Viewer
Internal MPEG4 Player
Internal AAAA Editor or viewer

Internal Editors and viewers of digital resources have to be realised in a compatible manner to be interfaced with the AXOM and to be hosted into the AXMEDIS Media Player User Interface and Windows Support.

They have to:

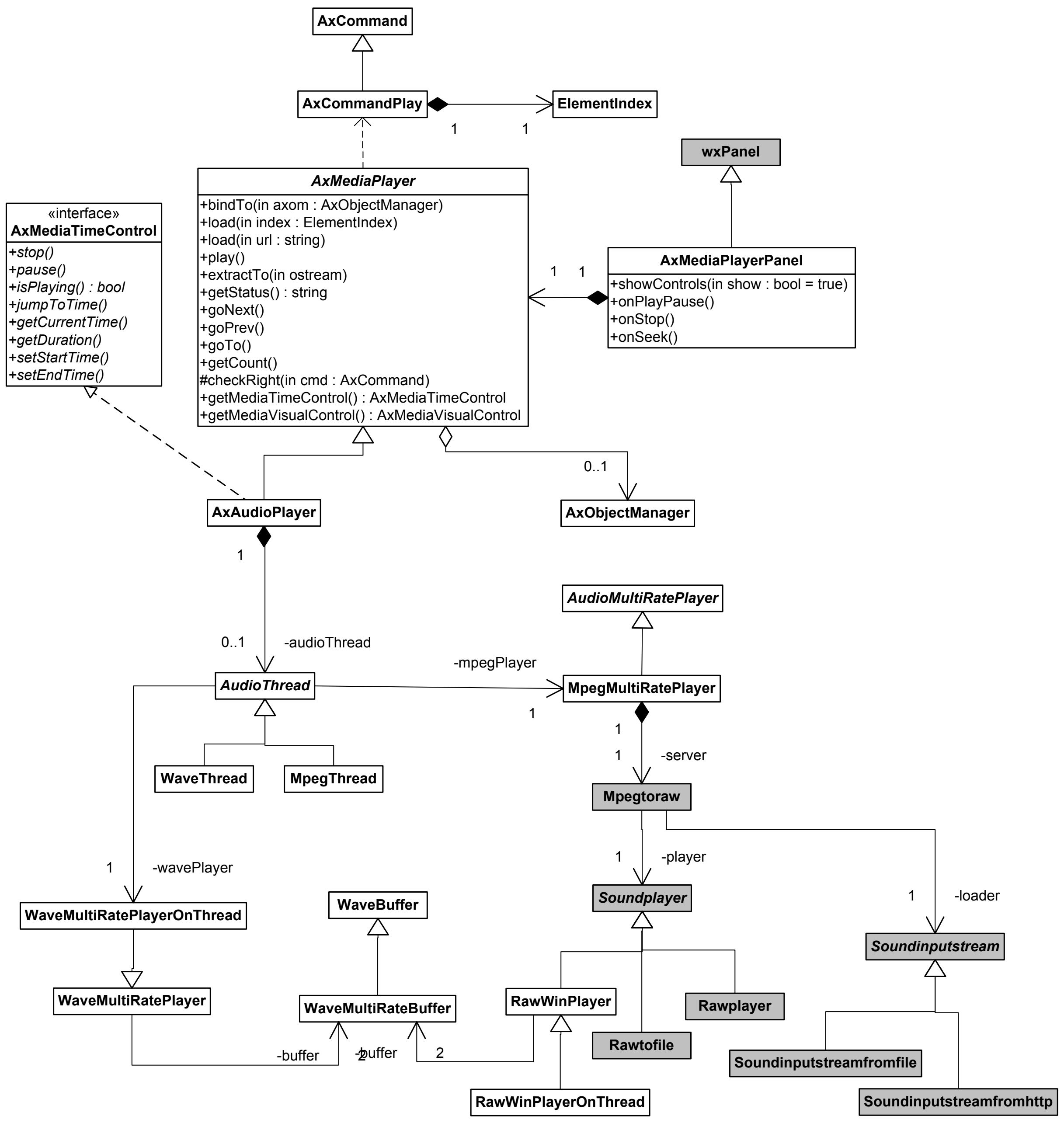
- be provided in Source Code and Lib/Obj to be linked to the rest of the application to guarantee the safeness of the Digital Resources.
- Provide a formal interface to export content processing functionalities if they are EDITORS such as: editing, merge, cut and past, save, load, etc.
- Provide a formal interface to export content Playing functionalities if they are PLAYER or VIEWERS: play, start, stop, pause, save, load, etc.
- Provide a formal interface to INIBIHIT and/or control all the functionalities as above describe
- Provide a formal interface to make a report of the actions performed by the user or requested by means of the interface.

6.1 Internal Audio Player (DSI, EPFL, …………)

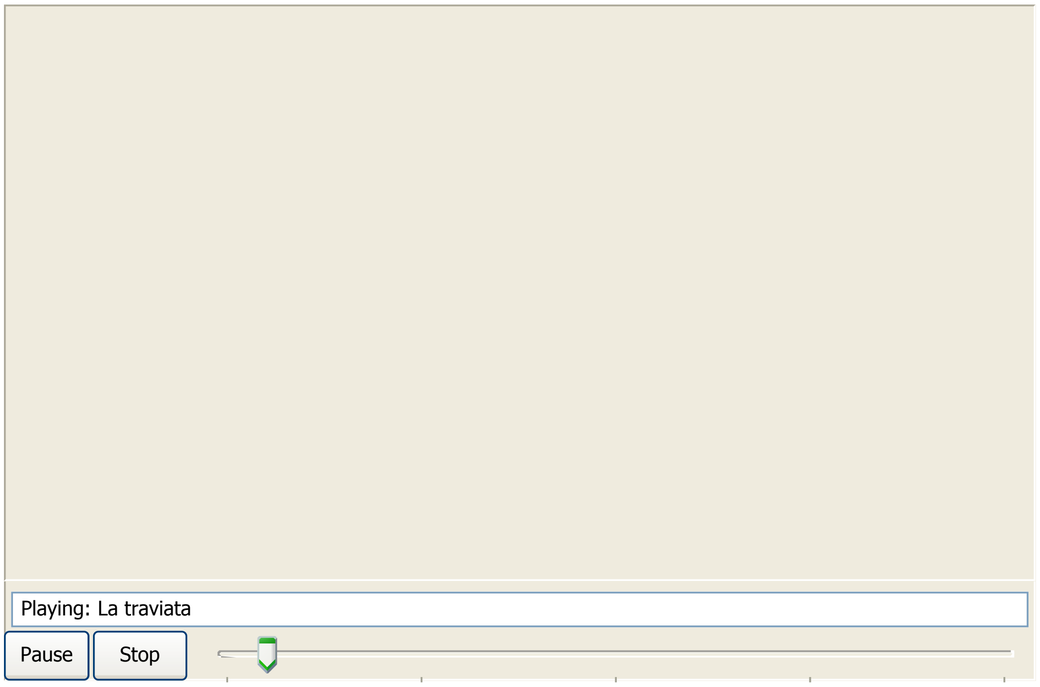
Module Profile		
Internal Audio Player		
Executable or Library(Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Responsible Name	Bellini	
Responsible Partner	DSI	
Status (proposed/approved)		
Platforms supported	Windows	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
MPEG3, WAVE, AAC		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
WxWIDGET	C++	
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
WxWIDGET		LGPL
splay		LGPL

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
- extract a portion of the audio to a stream

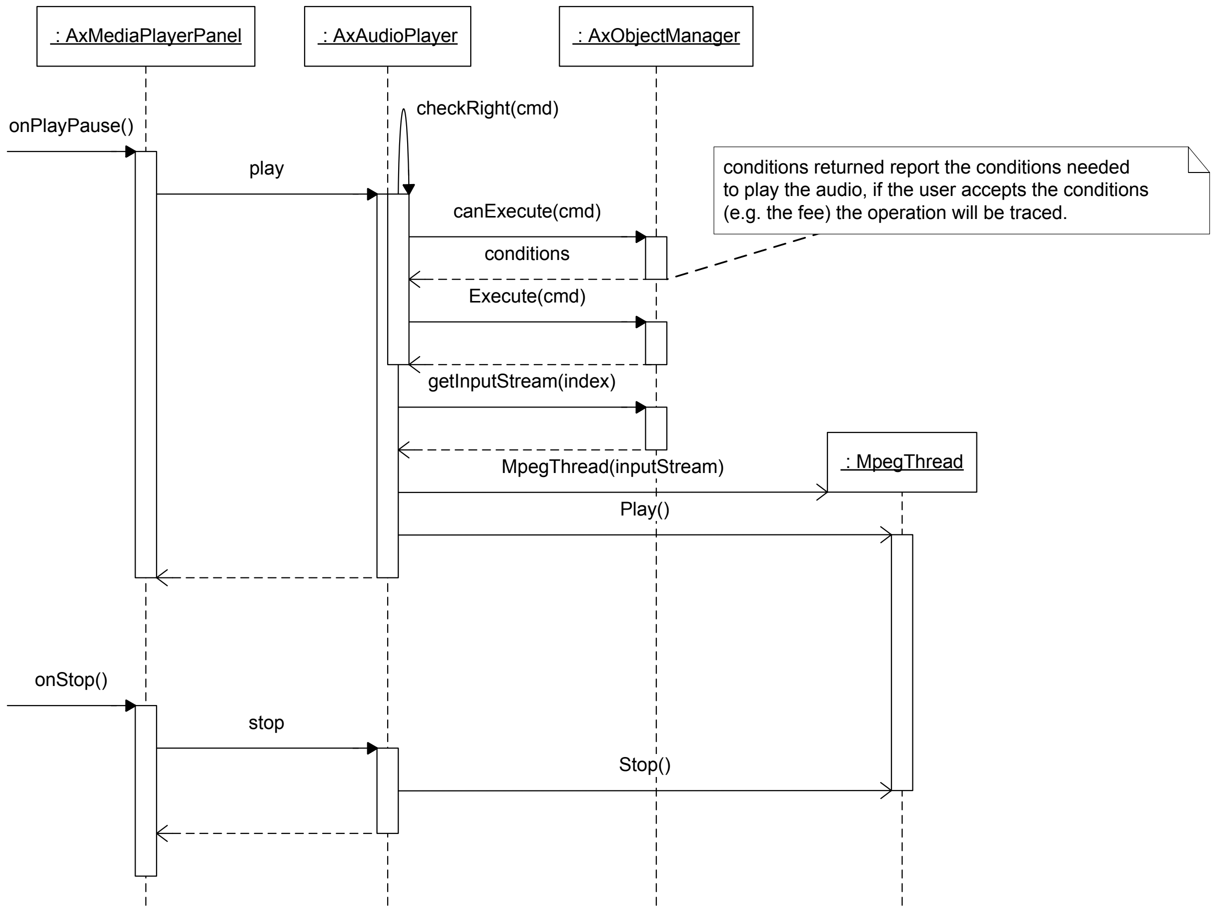


Class **axMediaPlayerPanel** is a panel containing the basic user interface for a media player allowing the basic operations on media files (play/pause, stop, seek), see the following picture for an example on how it may look like.


Playing: La traviata

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 *AxMedisPalyer* is specialized in **AxAudioPlayer, AxVideoPlayer** and **AxImageViewer.**
Class **AxAudioPlayer** implements the functionalities for audio using a library (splay) for MP3 and other code developed in WEDELMUSIC for WAV files. Another library to support AAC audio can be used. However this player may be substituted by the VideoPlayer since this one can also do audio playing only.

The following sequence diagram depicts what happens when the Play/Pause button is pressed:



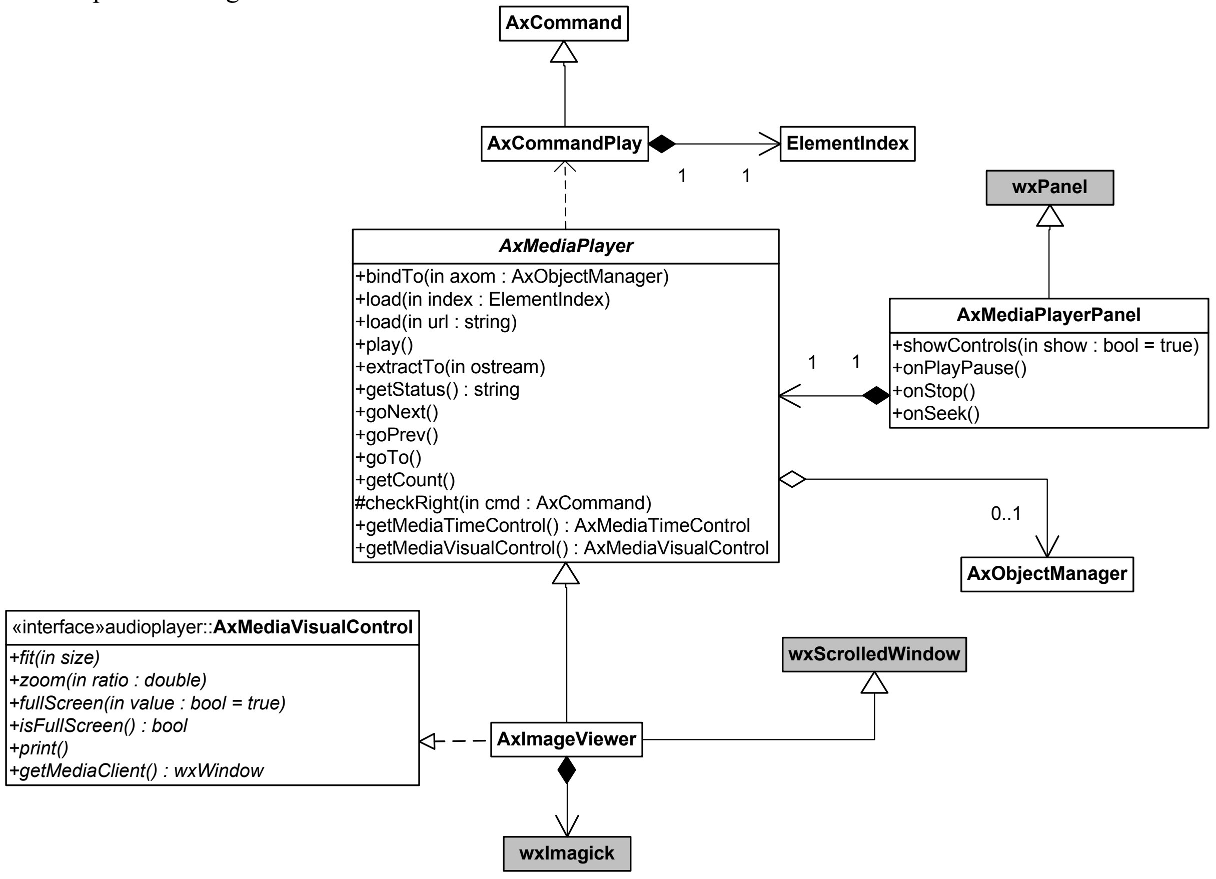
6.2 Internal Image Viewer (DSI,…………)

Module Profile		
<name……..>		
Executable or Library(Support)	Library	
Single Thread or Multithread	Multiple	
Language of Development	C++	
Responsible Name	Ivan Bruno	
Responsible Partner	DSI	
Status (proposed/approved)		
Platforms supported	Windows	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
Many, practically all		

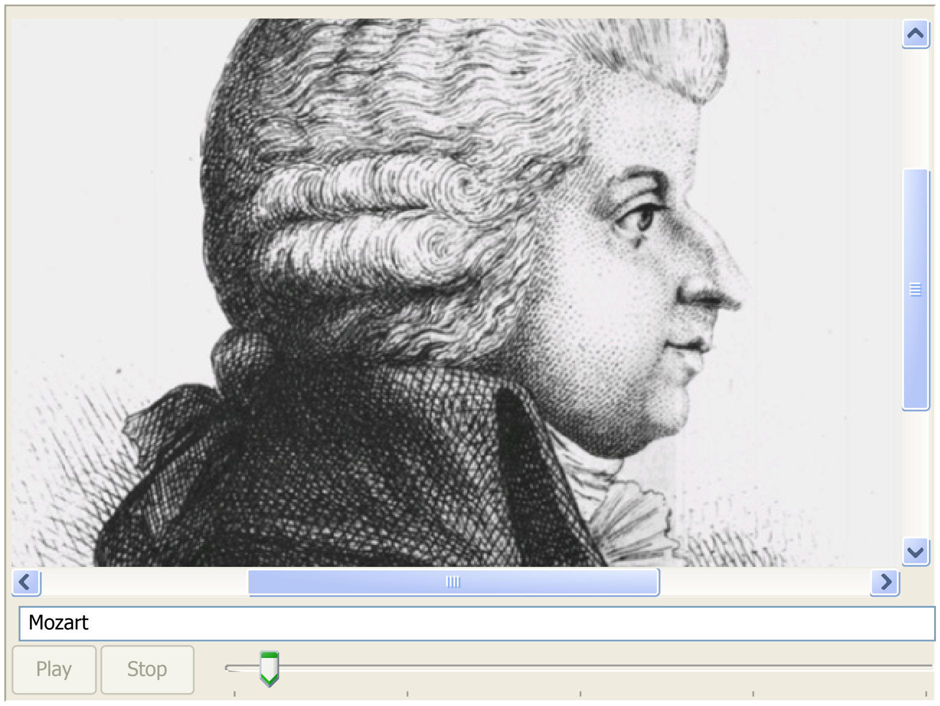
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
ImageMagik	ImageMagick	LGPL
WxWIDGET	WxWIDGET	LGPL
wxImagick		LGPL

The Internal Image Viewer will allow to display images and sequences of images (like multi image TIFF format), 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
- display the next/previous image in a sequence or display a specific one
- print an image



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.
The following is a possible user interface to view images.



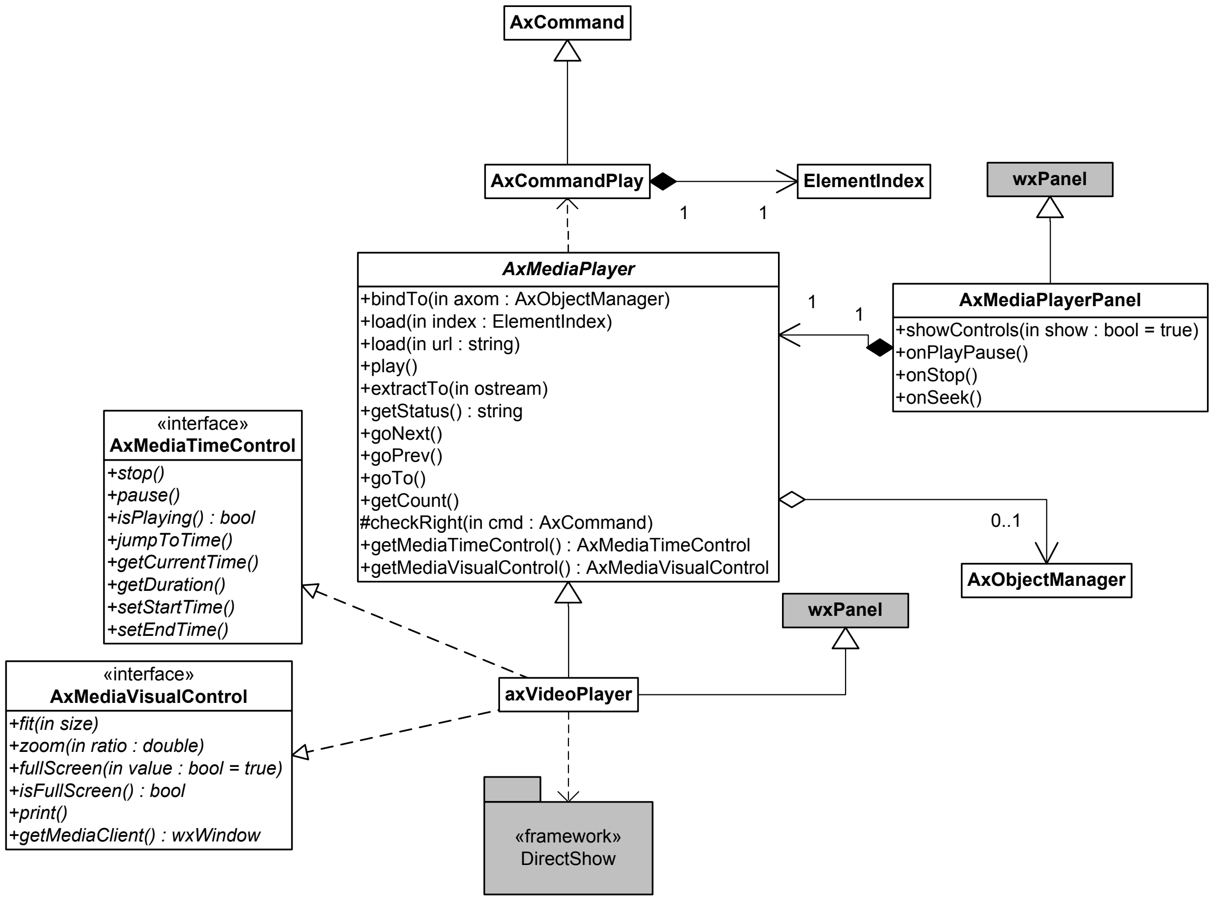
6.3 Internal Video Player (DSI, ………)

Module Profile		
<name……..>		
Executable or Library(Support)	Library	
Single Thread or Multithread	Multiple	
Language of Development	C++	
Responsible Name	Bellini	
Responsible Partner	DSI	
Status (proposed/approved)		
Platforms supported	Windows	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
AVI, MPEG, other video depending on the Codecs		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
WxWIDGET	C++	

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

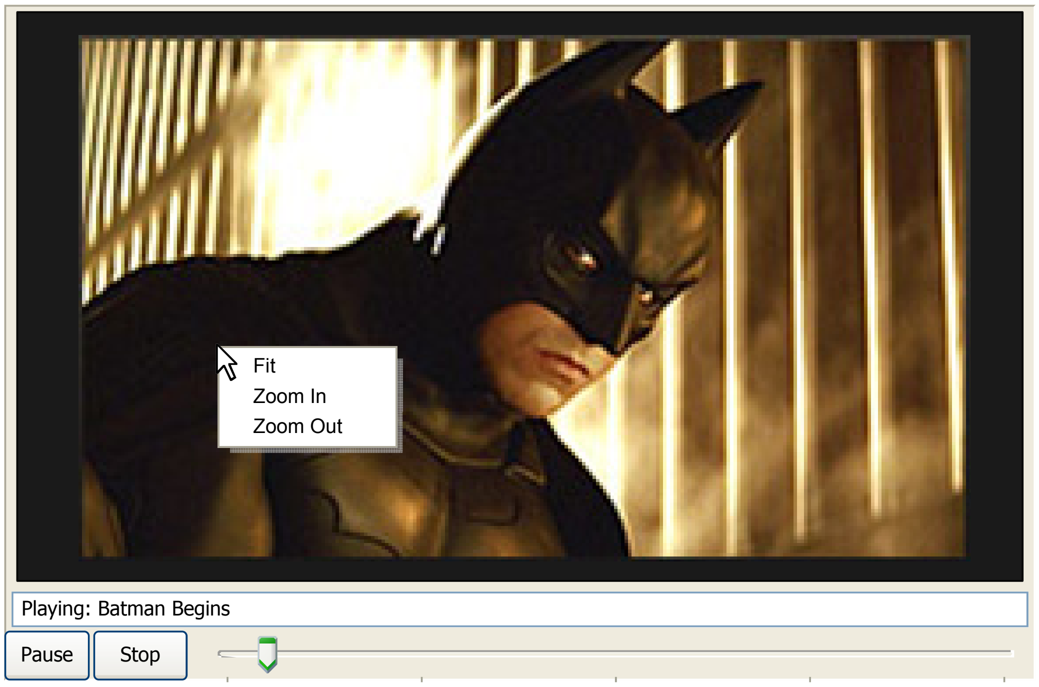
The Internal Video Player will be used to display video resources. It wiil provide functionalities to:

- 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



Class **AxVideoPlayer** implements the functionalities of **AxMediaPlayer, AxMediaTimeControl** and **AxMediaVisualControl** for playing a video.
Class ***AxVideoPlayer*** will be 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 a 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.

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

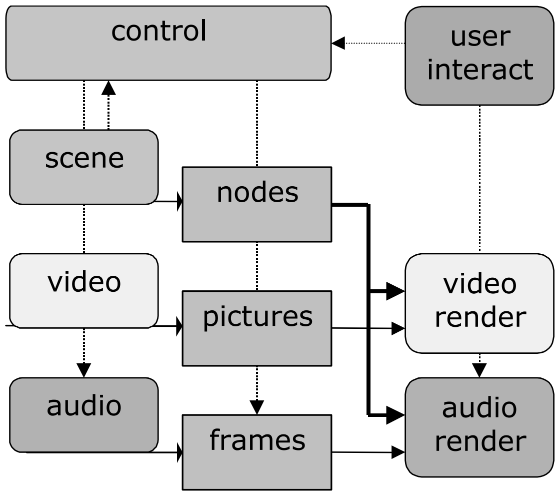


6.4 Internal MPEG-4 Player (EPFL)

Module Profile		
MPEG-4 Player		
Executable or Library(Support)	Executable or library (current status: executable)	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Responsible Name		
Responsible Partner	EPFL	
Status (proposed/approved)	Proposed	
Platforms supported	Windows 2k/XP	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
mp4 (MPEG-4 File Format)		
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		Proprietary (bSoft). Limited access to the source code and Use for the project can be negotiated with bSoft directly as done during IST CARROUSO. Also agreement with ENTHRONE may be investigated

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.

6.5 Internal SMIL Player (EPFL)

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.

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

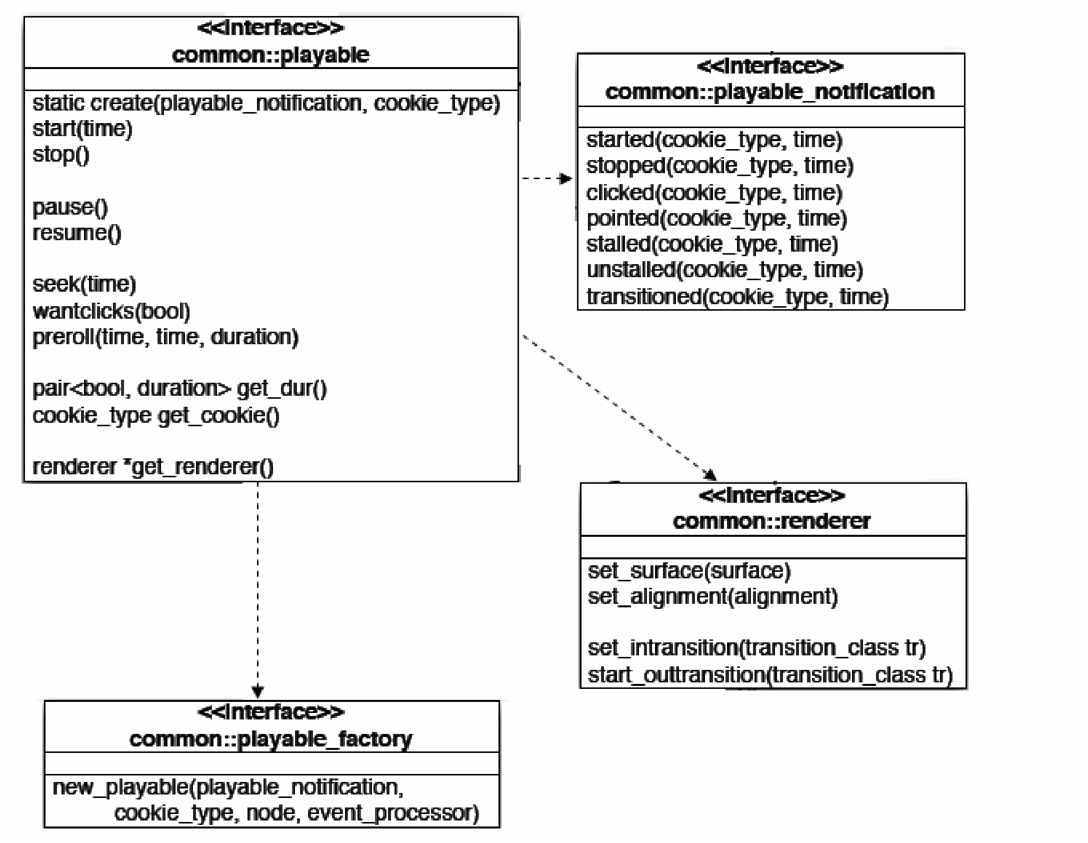
The **bindTo** function will be called on the SMIL player to attach it an AXMEDIS Object Manager. The SMIL player needs a reference to an Object Manager because the player does not have direct access to the AXMEDIS document. The SMIL player will use the reference to the Object Manager to read the AXMEDIS 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)

6.6 Document Viewer (DSI)

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

6.6.1 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)

To host ActiveX controls inside wxWidgets applications a wrapper class is provided in a library called wxActiveX (http://sourceforge.net/projects/wxactivex).

This library also provides an interface to host Internet Explorer as an ActiveX in wx applications.

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
- load from a wxString
- load from 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 to catch 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 allow to avoid to save 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.

The other solution could be to use wxMozilla which is a wrapper of mozilla for wx applications. It does not use the ActiveX technology thus allowing to use it also under other platforms (Linux, MACOS) but it seems to not provide a load from a stream object but the fact that both the library and mozilla are in open source gives the possibility to inspect and to adapt both to handle protection aspects.

6.6.2 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).

6.6.3 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 to open 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 plugins allowing to define 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

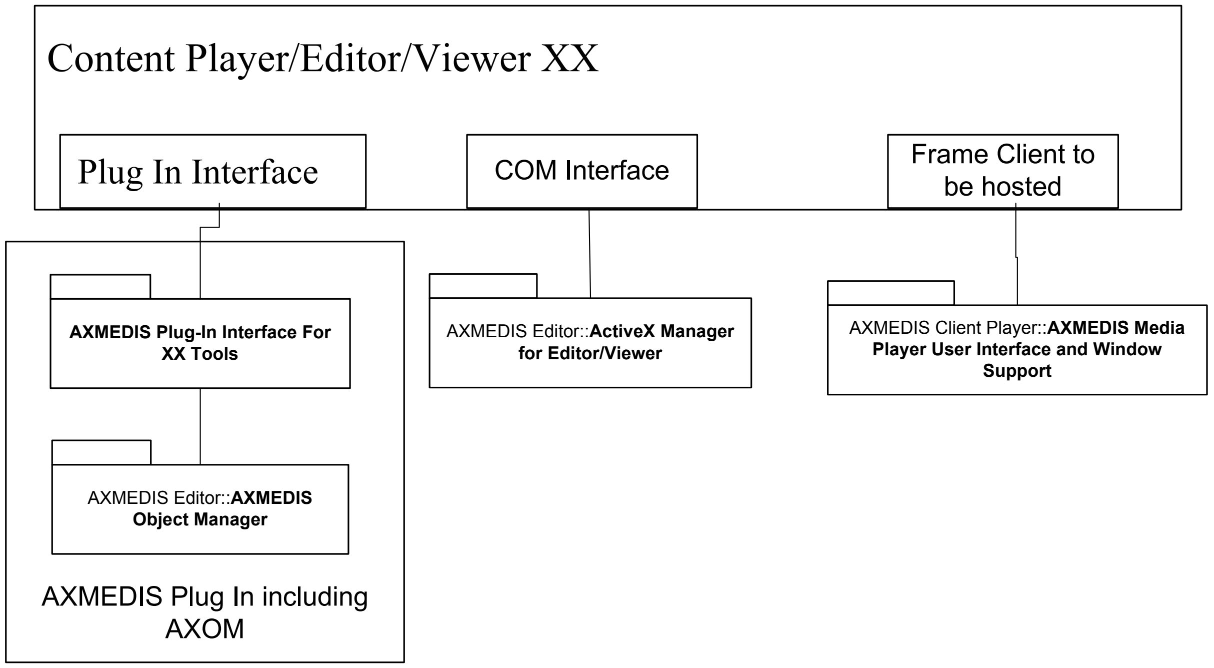
object also this key is put inside and protected. When the protected pdf file is opened it is stored on disk (unusable without the key) and the Acrobat ActiveX is used to open it, the AXMEDIS specific protection plugin for Acrobat will acquire the key from the AXEditor (e.g. via a protected channel) 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.

6.6.4 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.

7 External Editor/Viewer AXMEDIS Plug-ins and ActiveX Editor/Viewer AXMEDIS Plug-ins (WP4.1.3: DSI, WP4.1.4: EPFL)

AXMEDIS Other Players/Editors/Viewers

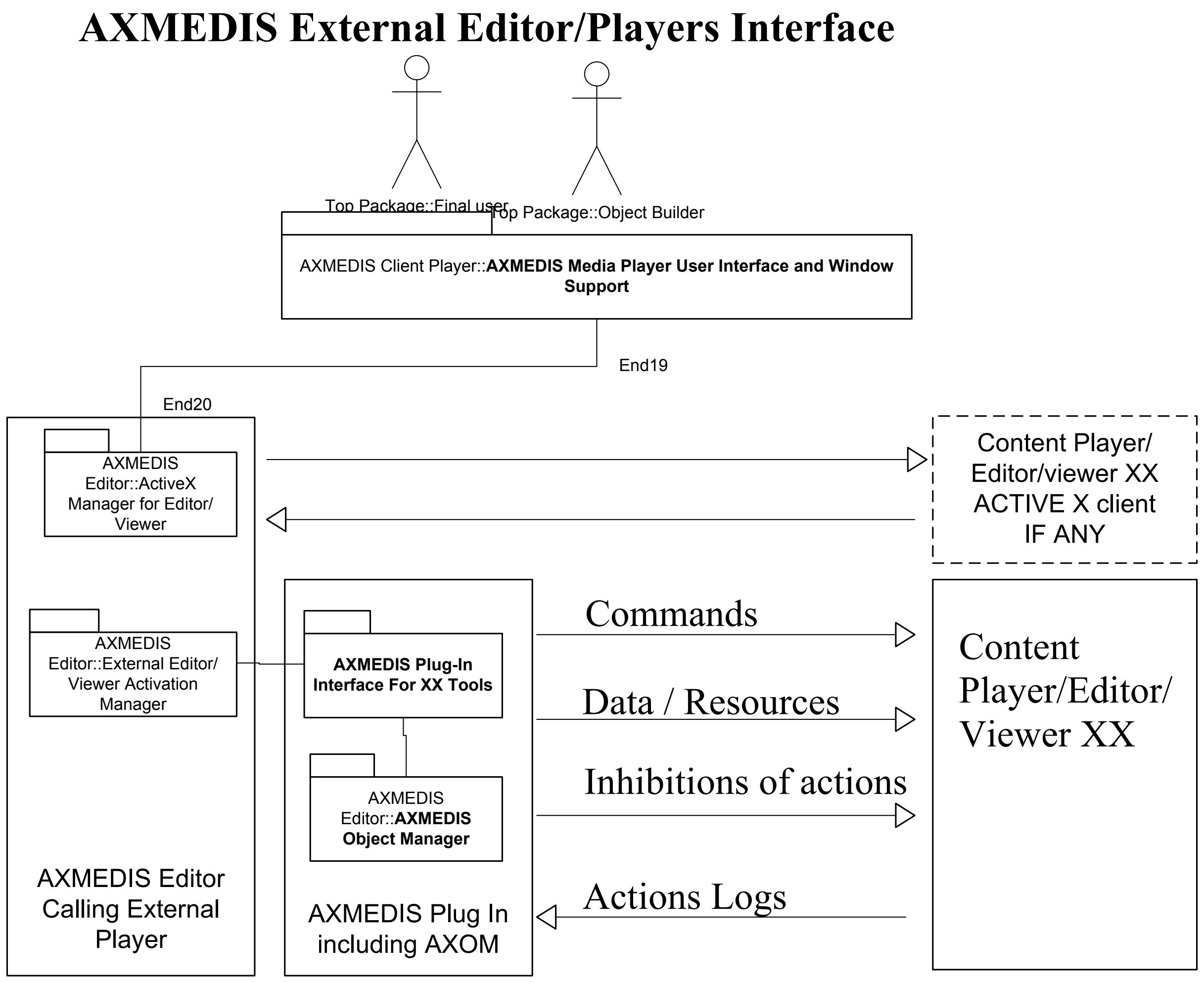

Content Player/Editor/Viewer XX
Plug In Interface
COM Interface
Frame Client to be hosted
AXMEDIS Plug-In Interface For XX Tools
AXMEDIS Editor::AXMEDIS Object Manager
AXMEDIS Plug In including AXOM
AXMEDIS Editor::ActiveX Manager for Editor/Viewer
AXMEDIS Client Player::AXMEDIS Media Player User Interface and Window Support

AXMEDIS Plug-In Interface For XX Tools module has to be realized for each Editor and Viewer in which we would like to insert an AXMEDIS plug in.

AXMEDIS editor for images could be realized in this manner into third party editors (such as XX above) if some DRM control is present via Plug In Interface in the those tools.

Some of the XX editor viewers can be available as ActiveX objects (such as ACROBAT reader). Those cases the Content Editor/Viewer XX can be controlled even via ACTIVE X Manager for Editor/Viewer of AXMEDIS. In these cases, the frame of the XX application can be hosted into the AXMEDIS Media Player User Interface and Windows Support

The AXMEDIS plug-in for each device has to respect basic control features on the target external player/editor/viewer.
Since the plug-in need to control the external player/editor/viewer in order to maintain the same security level. It is not acceptable to lost the protected assets just to be edited in specific tools.


AXMEDIS External Editor/Players Interface
Top Package::Final user
Top Package::Object Builder
AXMEDIS Client Player::AXMEDIS Media Player User Interface and Window Support
End19
End20
AXMEDIS Editor::ActiveX Manager for Editor/ Viewer
Content Player/ Editor/viewer XX ACTIVE X client IF ANY
AXMEDIS Editor::External Editor/ Viewer Activation Manager
AXMEDIS Plug-In Interface For XX Tools
AXMEDIS Editor::AXMEDIS Object Manager
Commands
Data / Resources
Inhibitions of actions
Actions Logs
Content Player/Editor/ Viewer XX
AXMEDIS Editor Calling External Player
AXMEDIS Plug In including AXOM

The EXEVAM uses a table to identify the accessible External Players, Editors and viewers with their profile if any, in alternative it can use the MIME if the resource is not protected
The EXEVAM is responsible of verifying if the External Editor is a secure environment according to the actions allowed by the license of the object under editing/playing, etc. If the external editor is not trusted the demand is not possible.

The communication from the EXEVAM and the AXMEDIS Plug in Interface for XX Tool is performed by means of a P2P communication layer with Discovery mechanism. This allows to see if a specific Editor is Open to do a given activity.

The AXMEDIS Plug in for some XX Editor/Viewer has to be developed on the basis of:

- AXMEDIS Development Tool Kit for External Players
- Software Development Tool Kit of the External Content Player/Editor XX

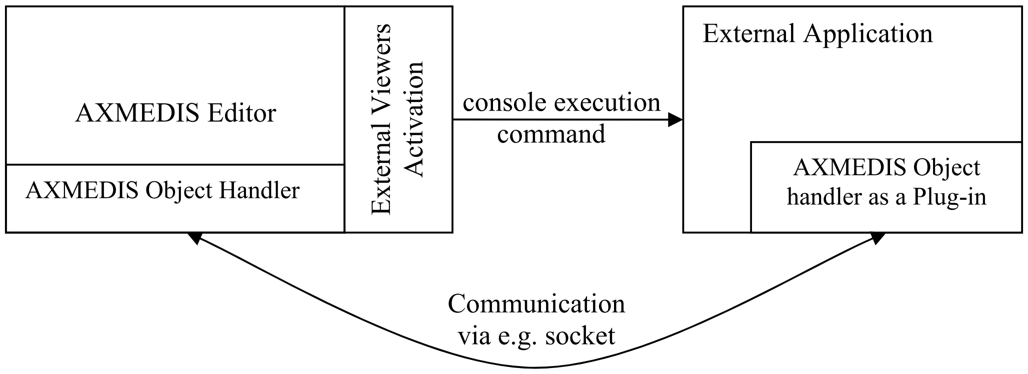
7.1 External Editor/Viewer Activation Manager, EXEVAM, (DSI)

Module Profile	
External Editor/Viewer Activation Manager	
Executable or Library(Support)	Library (Support)
Single Thread or Multithread	Single Thread

Language of Development	C++	
Responsible Name	Davide Rogai	
Responsible Partner	DSI	
Status (proposed/approved)	Proposed	
Platforms supported	All	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
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

External Editor/View 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;
The following figure depicts how the activation proceeds:

1. Once the AXMEDIS Editor receive a request by the user to open a give resource in the external editor; it can run a system command to launch the proper executable (External Editor) eg. Word to edit document, Photoshop to edit images
2. The external application installed the AXMEDIS Object Handler Plug-in; as soon as it is instantiated by the application it activates a communication channel to receive requests or to report notification/errors or to give back results.
3. The AXMEDIS Editor after the invocation it scan the local network interface in order to discover the Object Handler Plug-in communication channel (more than one at time can be activated).
4. When the AXMEDIS Editor discover the plug-in the initialization process is commenced in this phase several checks can be performed in order to assess the security level of the counter part plug-in in performing specific operations.
5. If the security requirements are fulfilled by the External editor Object Handler plug-in capabilities, than the object content is passed from the AXOM in the AXMEDIS Editor to the AXOM counter part in the plug-in.

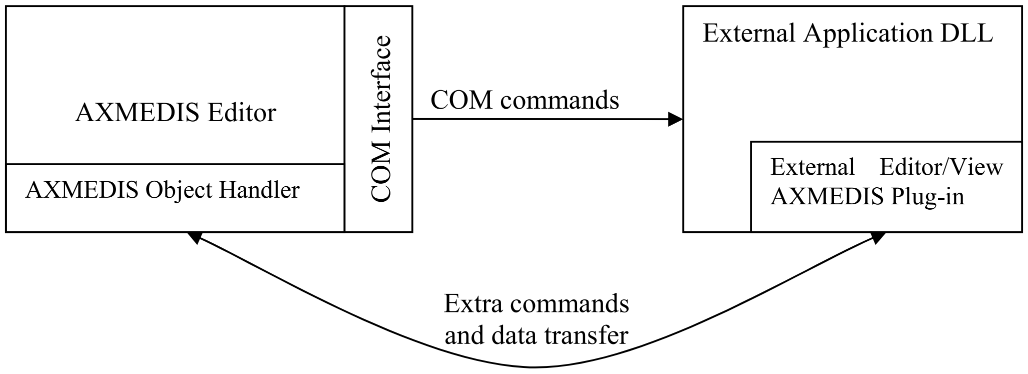

AXMEDIS Editor
AXMEDIS Object Handler
External Viewers Activation
console execution command
External Application
AXMEDIS Object handler as a Plug-in
Communication via e.g. socket

The same utilization scenario can be adopted on the ActiveX external editor/viewer/players; the basic functionalities are replicated for this architecture that simplifies the communication mechanism, as the latter reuses the COM features to interoperate with the ActiveX object. The behaviour of the ActiveX has to expose the same guaranties concerning security.

7.1.1 ActiveX Server

To host ActiveX controls inside wxWidgets applications a wrapper class is provided in a library called wxActiveX (http://sourceforge.net/projects/wxactivex).
It provides functionalities to:

- construct an ActiveX inside a wxWindow using the ProgId (e.g. "ShockwaveFlash.ShockwaveFlash") or the ClsId (e.g. CLSID_WebBrowser) and giving the id, position, size, style and name of the ActiveX Control.
- access to properties of the ActiveX
- call methods of the ActiveX
- receive and manage events generated from the ActiveX
- inspect properties, methods and events published by the ActiveX

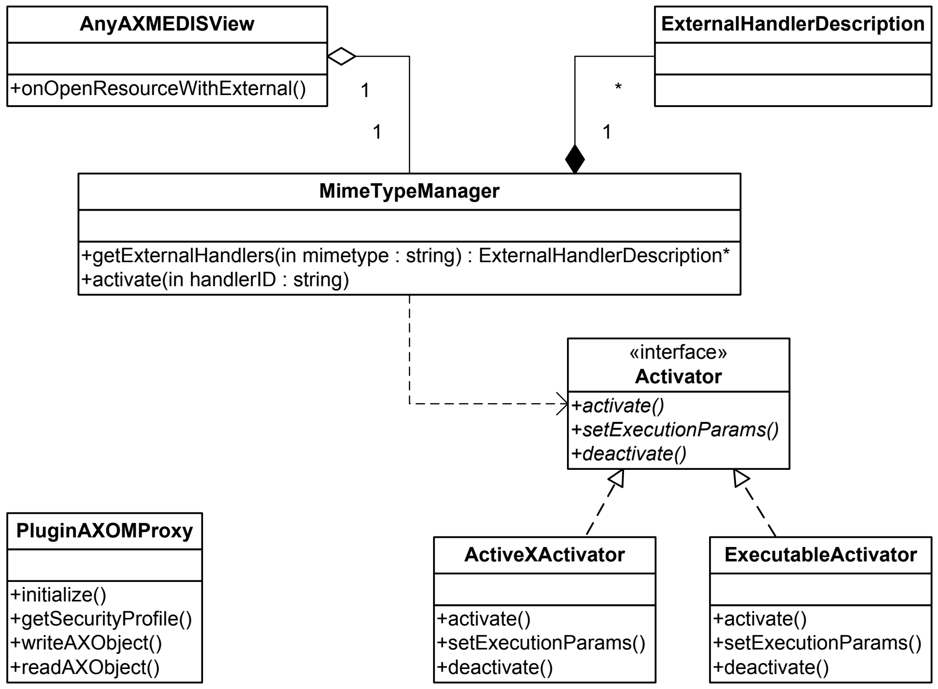


The communication channel is not a strong assumption: it is possible to arrange different solutions in order to make data available on both sides. The security does not depend on the communication channel, but is maintained at the required level with the communication between the two AXOM instances.

7.1.2 AXMEDIS Object Handler on the AXMEDIS Editor

A specific group of classes inside AXMEDIS editor provides different services in order to:

1. store a list of association between well-known mime-type and external editors (like what is provided by an operating system; in such a list every mime-type is related to a group of external player/viewer/editors descriptors: this descriptors collects relevant information about the related tool like executable location in the file system, command line parameters…
2. manage the AXMEDIS Editor event of "Open this resource with…" triggered by the proper gui.
3. activate a communication able object in order to discover the plug-in counter part in the external editor.

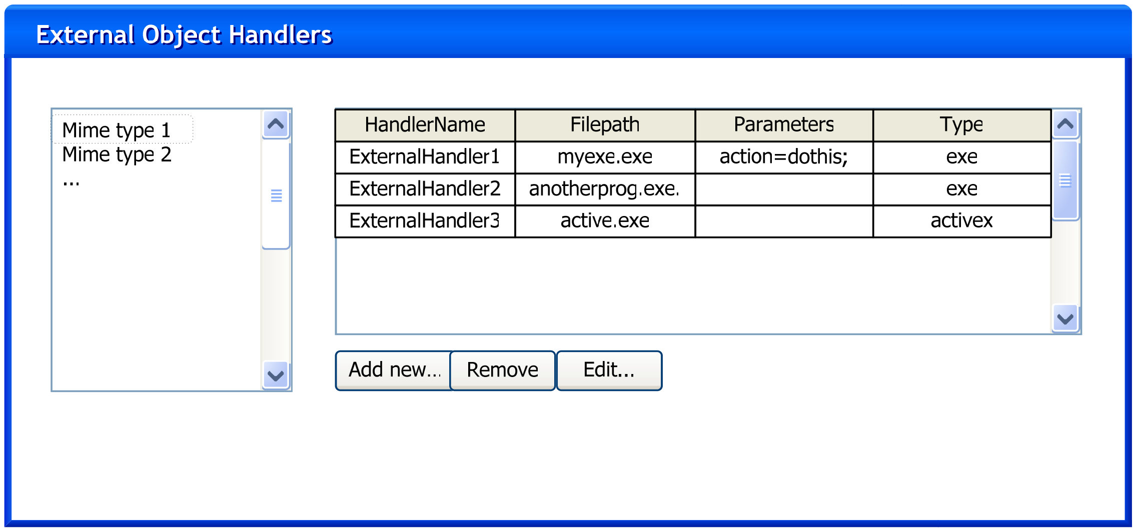


From any view in the AXMEDIS editor a resource of an AXMEDIS object could be opened with an external "handler" (with handler both cases of independent executable processes and ActiveX controls have been modelled).

The MimeTypeManager class is a support entity to select the proper external handler and it operates in two phases:

1. it allows to get a list of registered external handlers that can be activated for a specific resource mime type;
2. it acts as the gateway of the activation since it associates on the basis of the description the proper activator to invoke the external handler (Activator interface)

The MimeTypeManager exposes the interface to previously register the associations between mime type e by a suitable configuration GUI.



HandlerName	Filepath	Parameters	Type
ExternalHandler1	myexe.exe	action=dothis;	exe
ExternalHandler2	anotherprog.exe.		exe
ExternalHandler3	active.exe		activex

The detailed description of the external handlers is stored on a ExternalHandlerDescription.
Two different types of activators are conceived respectively with the aim to invoke the proper entity:

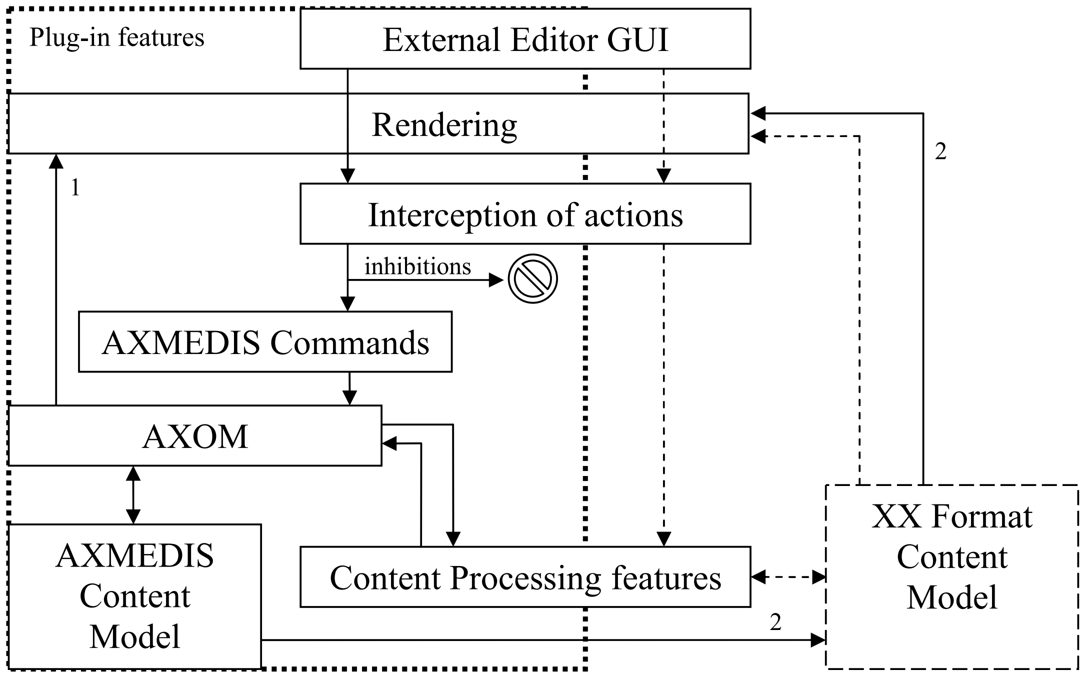
- basically the ExecutableActivator realizes the Activator interface to implement a console command activation of the desired executable a class has been conceived in order to support different platforms;
- the ActiveXActivator implements the activation via COM mechanism and host the ActiveX control inside the AXMEDIS editor executable (which acts as a COM server). It is based on the wxAutomationObject which provides functionalities to interact with OLE objects.

It is possible that, after the object manipulation in an external handler, that acts as a slave with respect to the invoker (AXMEDIS editor), the handler is terminated by the AXMEDIS editor automatically; this is an optional feature that should be managed by the activators.
Please note that the external editor (as an executable) could be already opened, when it is requested to activate in order to run as a AXMEDIS Editor slave. In this case it is possible to proceed in two manners:

- do not open a new instance of the external editor and try to discover the plug-in activated in the already opened
- open a new instance of the same external handler and discover the plug-in discovered in the new instance just opened.

7.1.3 AXMEDIS Object Handler as Plug-in for external Editor XX

On the external handler side another Object Handler has been conceived to act as the protection counter part in the secure architecture. In fact the security is based on the same protection mechanism of the AXMEDIS Object: the AXOM and its ProtectionProcessor are hosted in the Object Handler Plug-in and the protected object and its content are opened only inside the external handler memory context.
Ideally the plug-in should have the structure depicted in the figure below.



The dashed lines represent the normal flow of the information for content handling.
The solid lines represent how the AXMEDIS plug-in can check the authorization of every action performed by the user when it operates with the familiar external handler GUI.
The interception of the actions has the main role of enforcing only the rights licensed to the operating user; according to the different operations that the external editor GUI can initiate, the plug-in has to redirect all of them to the suitable "DRM-compliant" operation:

- an action can be directly inhibited, because is not bounded to any right described by a license
- an action can be realized by the instantiation of the required commands executable by the AXOM module.

Please note that new commands can be added to the basic set provided by the AXOM library itself. Actually it is the best way to include in the AXOM processing the specific content processing features that the external handler exposes.

The commands to be implemented for the plug-in are the basic blocks of the secure manipulation of contents: they must declare the required rights before the execution. The certification of plug-in is strongly based on the well definition of such rights.

After the content encapsulated by the AXOM has been changed the rendering phase is needed in order to feedback to the user what he has changed in the content. In the above diagram two different solution have been depicted:

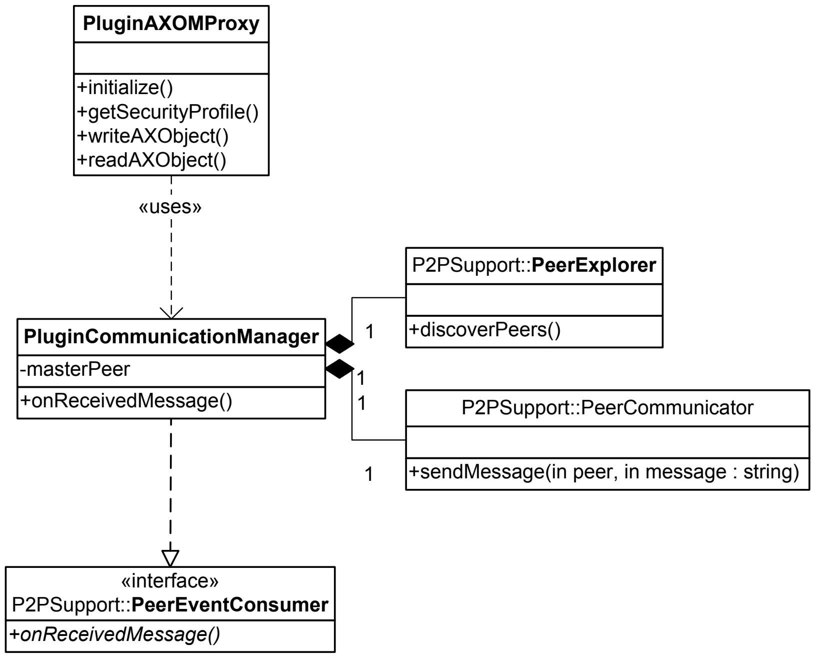
1. the content is extracted from the model and rendered by the plug-in (hard-coding solution, but decoupled from external editor limitations)
2. the content is extracted from the model in the plug-in to update the usual model for content modelling of the external editor; in this way the way of rendering the content model is totally reused.

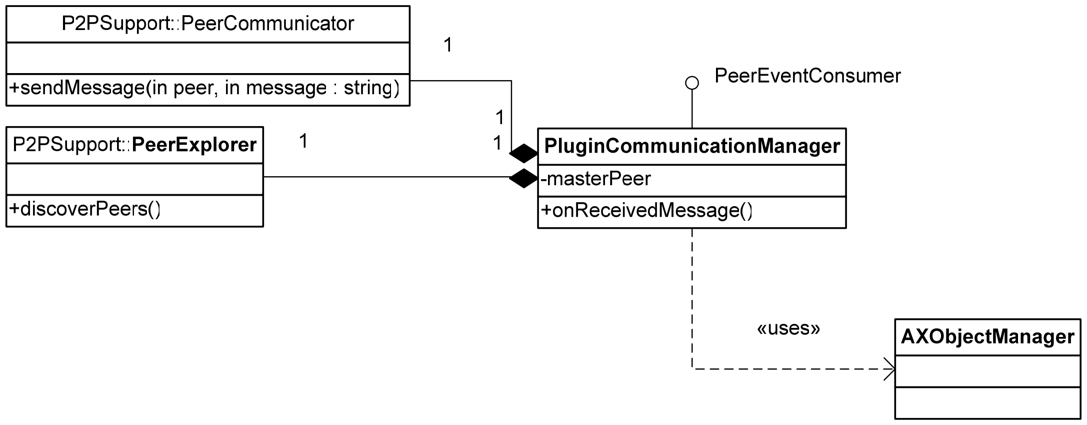
7.1.4 Communication and discovery of plug-in counter-parts

Some basic functionalities are included in the transport mean of information regarding security, content, commands, synchronization. A P2P framework can be used to allow several entities to communicate in a decentralized manner. On this ad-hoc networking some basic services are provided like:

- Discovery of joining peers
- Send/receive text messages

A specific handler of the communication events has to be set up to deal with protocol notification such as "a new message has been received": after this notification a command can be acquired or new content information can be transferred.





On the plug-in side the handling of messages is similar. It is trivial to model the reference peer as a master in the plug-in structure. The communication manager has the role of instantiating the content in a new AXOM object inside the plug-in. The transferred object could come from the PeerCommunicator or it could be read from a well-know file path: it is the resource that must be processed in the external handler.

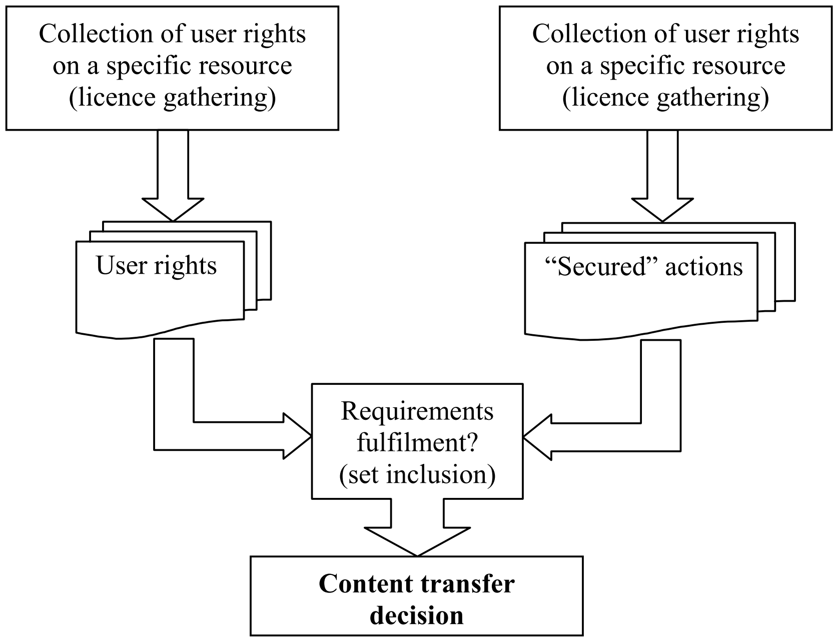
7.1.5 Security level negotiation

In some cases the supposed architecture depicted in the above diagram cannot be provided by the plug-in technology of the hosting program. If not all the user operation can be "handled" in the sense of avoided or redirected to the secure command execution, this limitation could compromise the security framework built on controlled content manipulation.

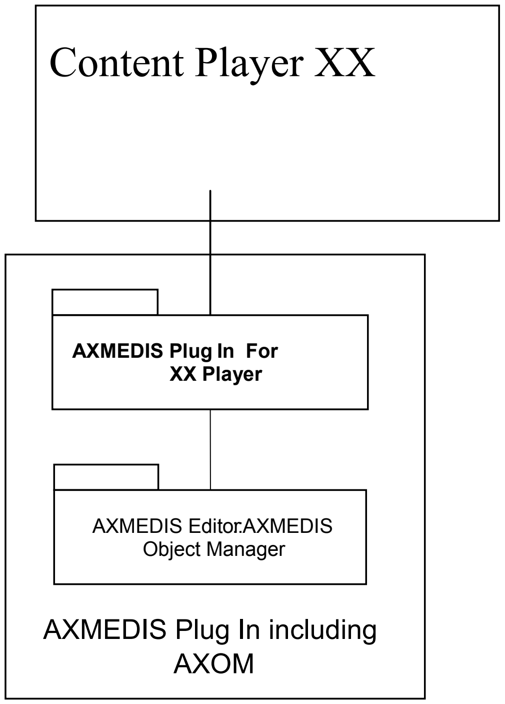
Once the content (e.g. an image or a video) has reached the non-secure program the user can perform un-authorized actions in an easy manner (e.g. save the content on the disk in another format).

Some incomplete secure architecture have to be declared by the plug-in and managed by the security framework. The decision to transfer a content or not must be taken on the basis of matching the security requirements of a particular content manipulation.
For example: An user has granted by a specific license the editing/adapting of a resource, but not to save it on the disc in an unprotected form; he wants to edit outside of AXMEDIS editor that resource. If the external editor cannot inhibit the save actions on its GUI, it has to be considered that the resource cannot be transferred in the external editor, because this transfer compromises the DRM enforcement on the resource.

The decision can be taken automatically by a preference settings or asked to the used (resource editor).
A profile of the plug-in is needed for any external editor plug-in. The secure profile of any plug-in must be certified, since it declares which action are (are not) inhibited by the plug-in development.
A policy has to be define in order to manage the rights/actions matching.


Collection of user rights on a specific resource (licence gathering)
Collection of user rights on a specific resource (licence gathering)
User rights
"Secured" actions
Requirements fulfilment? (set inclusion)
Content transfer decision

8 Plug ins in other Players (DSI, EPFL, UNIVLEEDS)


Content Player XX
AXMEDIS Plug In For XX Player
AXMEDIS Editor.AXMEDIS Object Manager
AXMEDIS Plug In including AXOM

8.1 General Plug-in Aim and Support

AXMEDIS Plug-In module has to be realized for each Player in which we would like to insert the capability of processing content packaged in AXMEDIS objects.

It is reasonable that AXMEDIS players could add value to the AXMEDIS content if they are pluggable in:

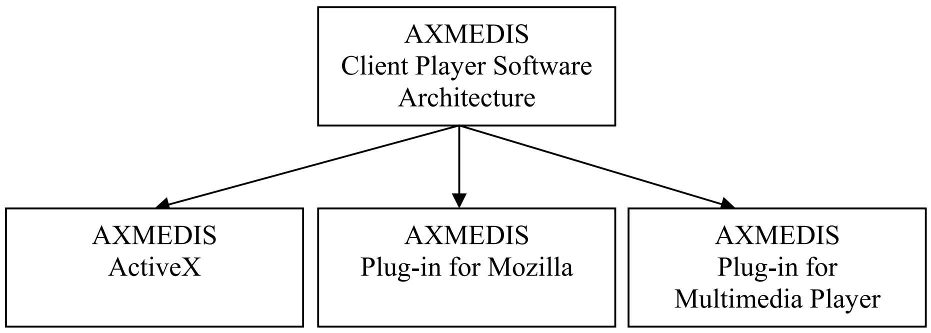
- browsers like IE, Mozilla etc,
- multimedia players like WindowsMedia technology etc…

The different plug-in share the common feature that they has to manage AXMEDIS content; in some case they must also provide the ability to render the content, in other cases they just have to extract from the object and redirect to the target media format.

Three main architecture has been identified that cover most of the plug-in scenario of required integrated players:

1. AXMEDIS ActiveX
2. AXMEDIS Plug-in for Mozilla
3. AXMEDIS Plug-in for Multimedia Players

In the following section the AXMEDIS client software architecture is tailored to fit these three different utilization of AXMEDIS technology.

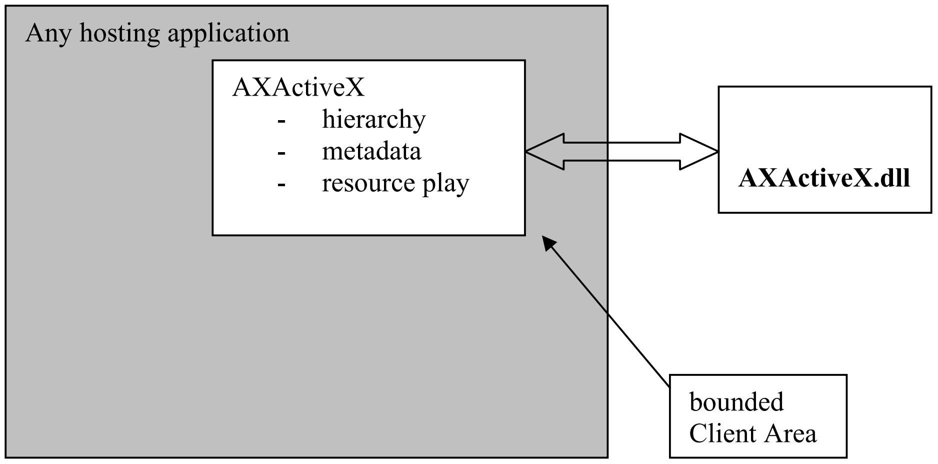


8.2 AXMEDIS ActiveX Control (DSI)

Module Profile		
AXMEDIS plug-in into Microsoft Internet Explorer		
Executable or Library(Support)	Library	
Single Thread or Multithread	Multhread	
Language of Development	C++	
Responsible Name	Davide Rogai	
Responsible Partner	DSI	
Status (proposed/approved)	Proposed	
Platforms supported	Windows	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
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
ActiveX development provided by Visual Studio		

The ActiveX technologies allows the plug of an AXMEDIS player inside any compliant executable. First of all the AXMEDIS content can be viewed in the most common interface, which is the Microsoft Internet Explorer browser.
Several products designed for the windows platform have the capability of hosting ActiveX controls: Microsoft Office Suite, Macromedia Tools, Authorware, ToolBook.

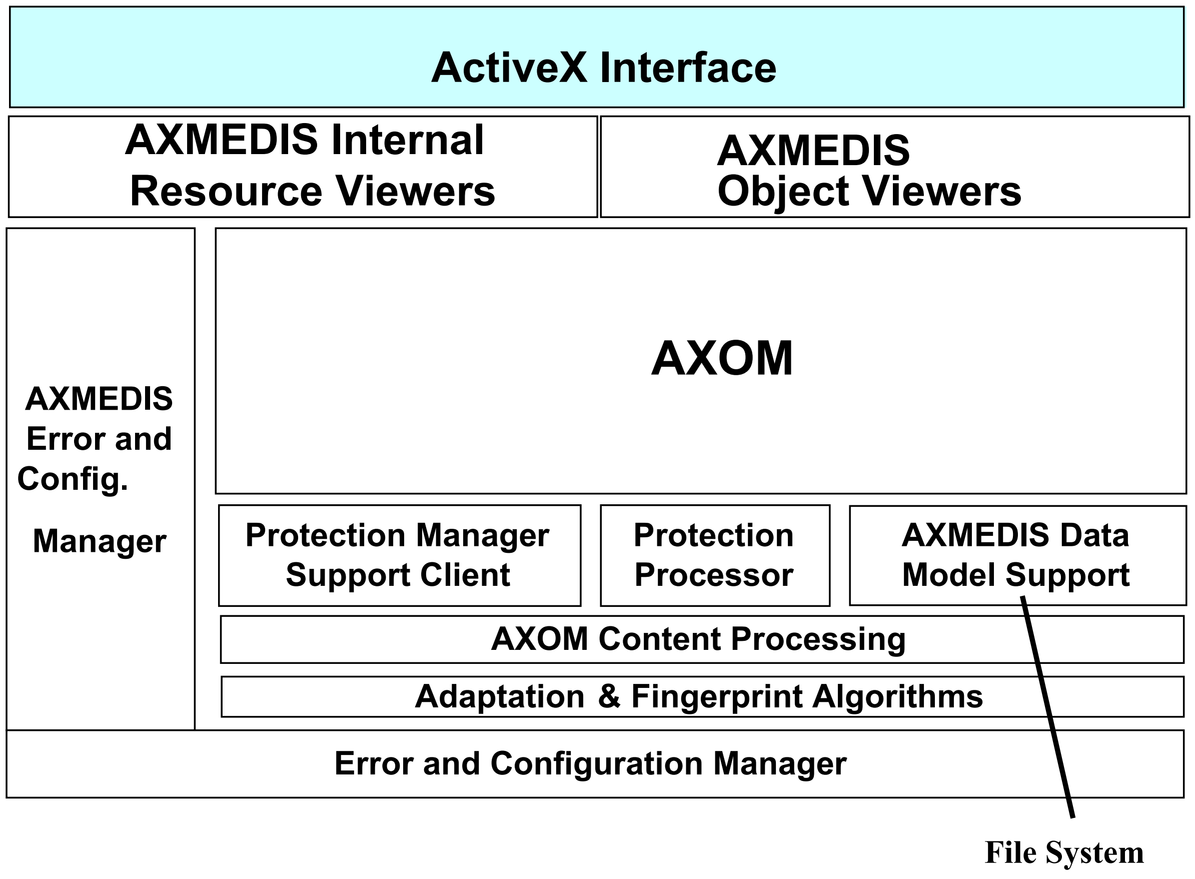
In the main client view of the target application the AXMEDIS ActiveX can be rendered in a specific bounded client area. The control takes the responsibility of render information and manage user interactions regarding such an area.



Since the AXMEDIS ActiveX has to render itself on the screen portion it needs:

- some of the AXMEDIS viewers to be able to show hierarchy and metadata about content
- all the internal resource viewers (audio, video, image, document…)

AXMEDIS Active X Software Architecture

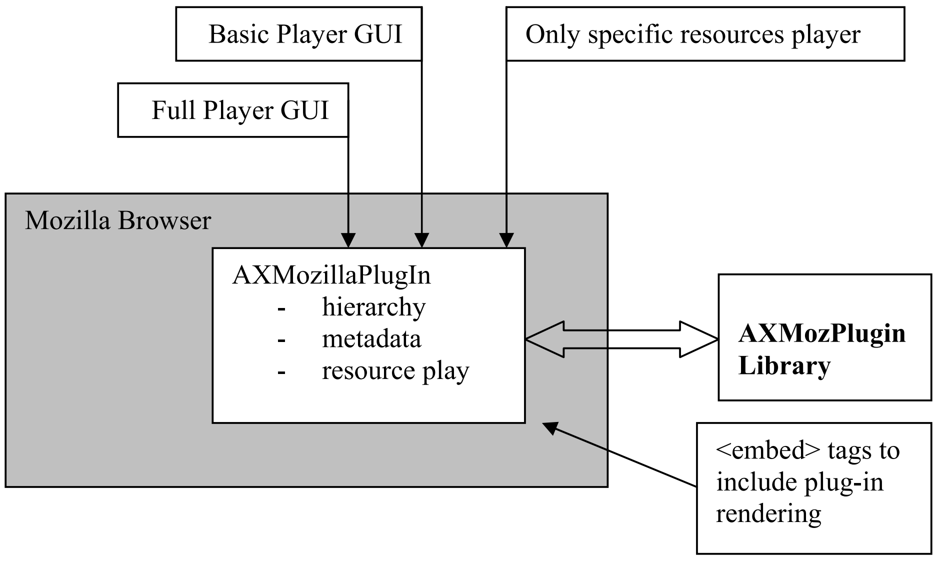

ActiveX Interface
AXMEDIS Internal Resource Viewers
AXMEDIS Object Viewers
AXMEDIS Error and Config. Manager
AXOM
Protection Manager Support Client
Protection Processor
AXMEDIS Data Model Support
AXOM Content Processing
Adaptation & Fingerprint Algorithms
Error and Configuration Manager
File System

8.3 AXMEDIS plug-in into Mozilla (SEJER)

Module Profile		
AXMEDIS plug-in into Mozilla		
Executable or Library(Support)	Library	
Single Thread or Multithread	Single Thread (NPAPI Is NOT Thread Safe)	
Language of Development	C++	
Responsible Name		
Responsible Partner	Sejer	
Status (proposed/approved)		
Platforms supported	Windows, Linux, MacOsX	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
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

8.3.1 Introduction

In this section, a Mozilla Plug-in of the AXMEDIS Player is specified. This is a piece of code that knows how to display an AXMEDIS object, at the request of Mozilla browser and inside the window of this Mozilla Browser.
The AXMEDIS player is considered as being able to render the AXMEDIS content into its own window.



A Mozilla plug-in is a dynamic code module that is native to the specific platform on which the Mozilla browser is running. It is a code library, rather than an application or an applet, and runs only from the browser.

The AXMEDIS plug-in may have to render many kind of content type (video, audio, text etc.). Some times, it may be necessary for the plug-in to render a GUI allowing the user to manipulate the AXMEDIS content (ala Acrobat Reader), sometimes it may be desirable to only display the AXMEDIS content without anything around so that it integrates himself seamlessly into the hosting content. Thus, depending on the parameters provided on the <embed> tag calling for the plug-in, the pug-in may choose to :

- Display the whole AXMEDIS Player GUI
- Display a lightweight, plug-in specific version of the Player GUI
- Display the content without any kind of GUI.

8.3.2 The Mozilla Plug-in Technology

With the Mozilla Plug-in API, one can create dynamically loaded plug-ins that can:

- register one or more MIME types
- draw into a part of a browser window
- receive keyboard and mouse events
- obtain data from the network using URLs
- post data to URLs
- add hyperlinks or hotspots that link to new URLs
- draw into sections on an HTML page
- communicate with Javascript/DOM from native code

The Mozilla Plug-in API has two levels. The first level is constituted by the basic, old Netscape Plug-in API named NPAPI, which comes from the first ages of the Netscape browser and is recognized by practically every browser on the market. Even MS Internet Explorer prior 5.5 understands this API, and for latter version, after SP2 in which Microsoft has removed support for Netscape Plug-ins, there is a work around.

Thus the AXMEDIS Player plug-in implements this API. With some care the plug-in should then be able to compile into a valid Opera Browser, Safari Browser and others plug-in, depending and the platforms the Player can be compiled to.

At a second level, this API has been improved, in collaboration with Opera software, Sun, Apple, Adobe etc. to extended interaction capacities between the plug-in and the hosting browser, that is:

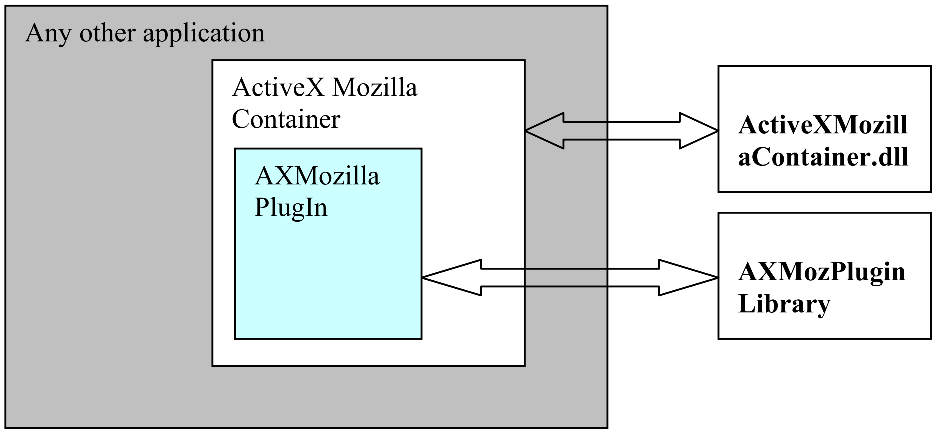
- allowing the hosting browser to call some methods the plug-in exposes to him, that is making the plug-in scriptable
- allowing the plug-in to access the DOM of the hosting Window to manipulate it.

The implementation status (stable or unstable) of this API is not clear for now but it should become stable rapidly from now and it seems safe to rely on it, as it is said that the API is close to being frozen..

The API allows defining windowless or windowed plug-in, which mean respectively: plug-in that draw themselves directly into the window of the hosting page, or plug-in that draws themselves in their own window within the hosting web page. The second solution is used for the AXMEDIS plug-in, because, it is safer.

Also, the id of the plug-in, which is an URI following a specific format to ensure unicity, must be in the form: ***@Domain/ProductName,version=[?versionInfo&versioninfo=],*[ModuleIdentifier.*** @domain, ProductName and Version are mandatory.
See http://www.mozilla.org/projects/plugins/plugin-identifier.html for further details.

Last, but not least, an "ACTIVEX Control for Hosting Netscape plug-ins in IE" project can be found on the Mozilla web site at http://www.mozilla.org/projects/plugins/plugin-host-control.html. This control is able to embed a Mozilla Plug-in to make it available to IE. This feature could be used to not do twice the same coding effort at least for an initial validation of plug-in concept.



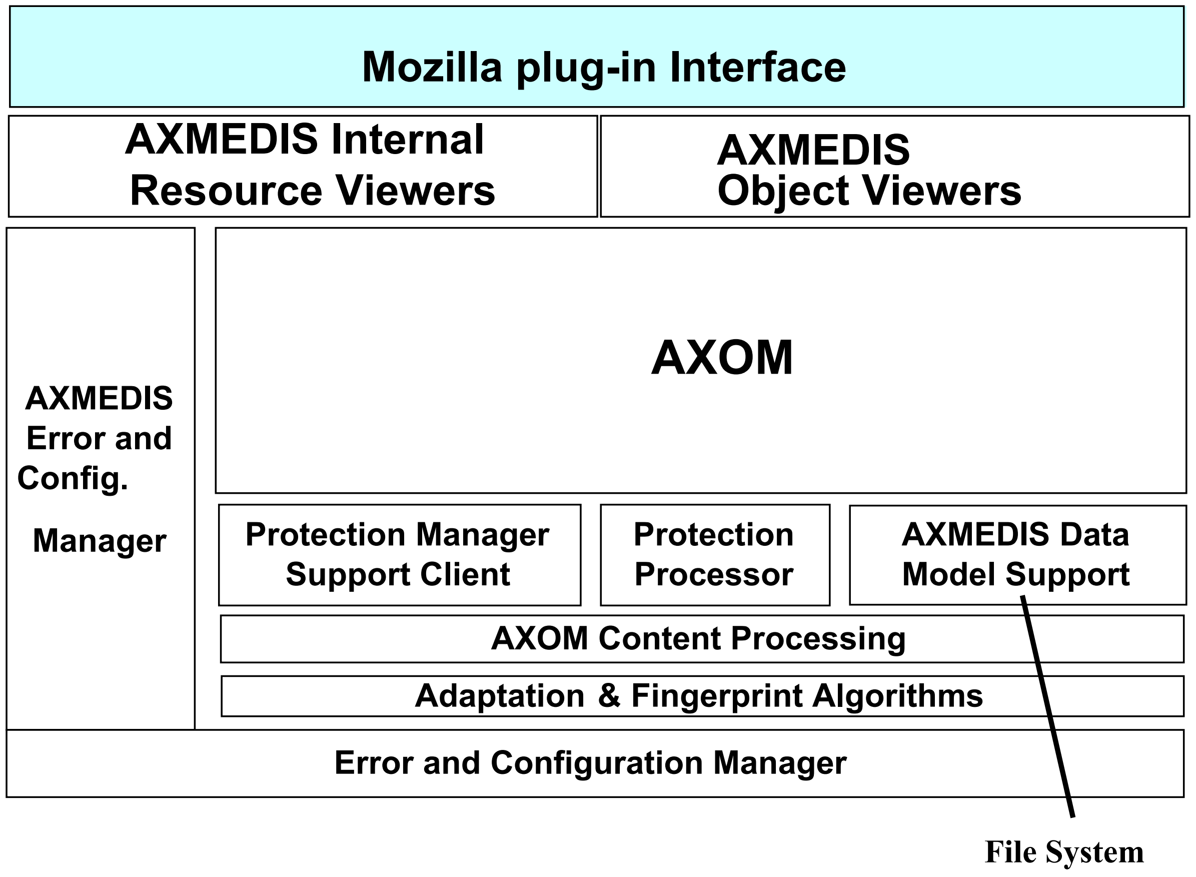
8.3.3 Authentication

For the plug-in to be able to manipulate AXMEDIS object and react to user requests through the browser & player GUI, the user must be authenticated. When being instantiated, the plug-in:

- It pops up a dialog box asking the user to authenticate himself and request authentication through the AXOM
- If authentication fails, display a message instead of the content
- If authentication succeeds, start processing the requested AXMEDIS content
- After a successful identification all the user licence are at disposal to consume the requested content.

8.3.4 Architecture

AXMEDIS Mozilla Plug-in


Mozilla plug-in Interface
AXMEDIS Internal Resource Viewers
AXMEDIS Object Viewers
AXMEDIS Error and Config. Manager
AXOM
Protection Manager Support Client
Protection Processor
AXMEDIS Data Model Support
AXOM Content Processing
Adaptation & Fingerprint Algorithms
Error and Configuration Manager
File System

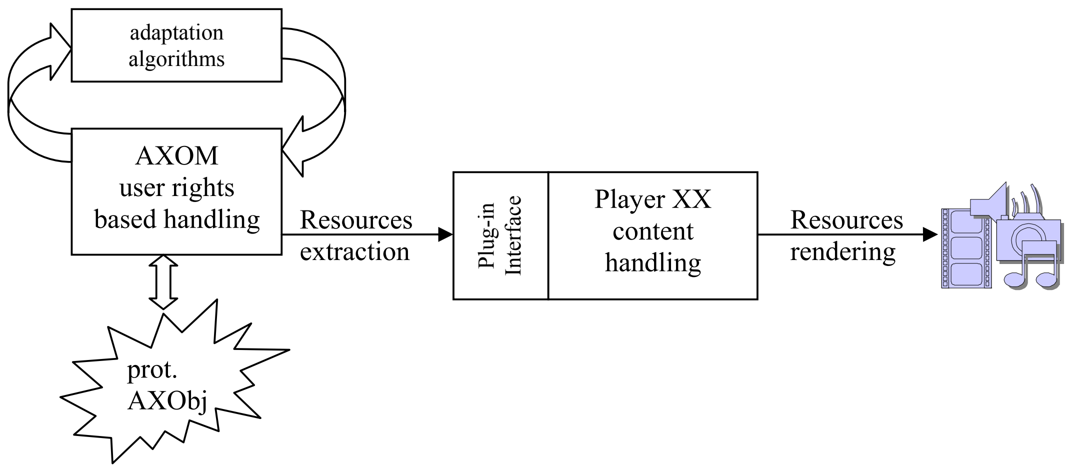
Moz docs & API : http://www.mozilla.org/projects/plugins/

8.4 AXMEDIS plug-in into Multimedia Players (DSI)

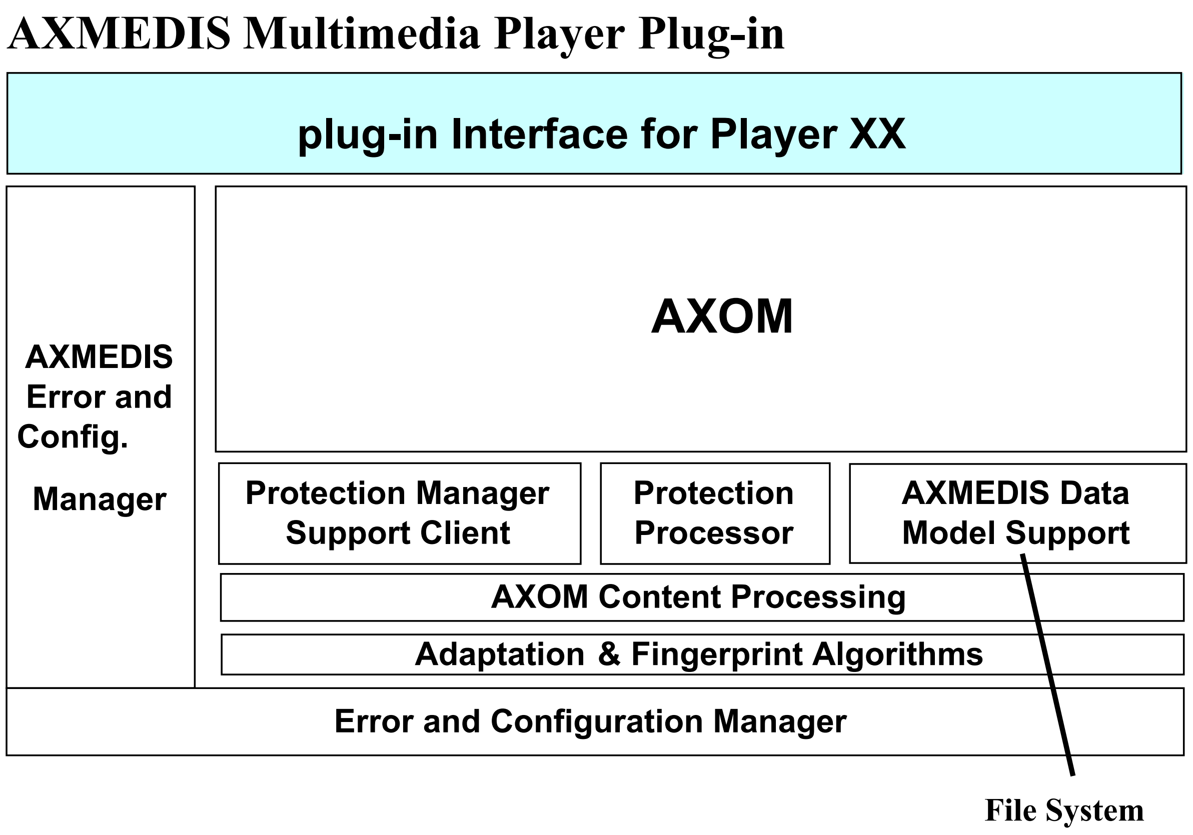
Module Profile		
AXMEDIS plug-in into Microsoft Windows Media		
Executable or Library(Support)	Library (Support)	
Single Thread or Multithread	Singlethread	
Language of Development	C++	
Responsible Name	Davide Rogai	
Responsible Partner	DSI	
Status (proposed/approved)	Proposed	
Platforms supported	Windows, Linux, MacOsX	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
Multimedia players plug-in technologies		DLLs – communication interfaces dependent on the specific player
File Formats Used	Shared with	File format name or reference to a section
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

The main difference among AXMEDIS ActiveX/Mozilla Plug-in and AXMEDIS Plug-in for Multimedia players is the rendering capabilities. The first manages all the client area assigned to it, full of controls, scrollbars, options, tabs… etc. In the Plug-in for Multimedia the most important part is rendering of the content. Since the content in AXMEDIS is a composition of most used multimedia formats, the resource can directly be rendered by other multimedia players (i.e. Windows Media Player). Two important issues have to be managed:

1. Adaptation capabilities of AXOM can play a great role inside other players as the plug-in capability of play a content in a preferred player also if it was not designed explicitly for it
2. Since the plug-in of AXMEDIS in other player can really increase the AXMEDIS utilization in the consumers' world; the DRM has to be considered with much attention.



The AXOM module and its protection processor grants the proper manipulation of content with respect to the user licence. When this secure environment collides with external plug-in technology such a warranty can only be treated at level of certification.
The best solution is that the plug-in acts as certified viewer/player of the AXMEDIS content.


AXMEDIS Multimedia Player Plug-in
plug-in Interface for Player XX
AXMEDIS Error and Config. Manager
AXOM
Protection Manager Support Client
Protection Processor
AXMEDIS Data Model Support
AXOM Content Processing
Adaptation & Fingerprint Algorithms
Error and Configuration Manager
File System

8.4.1 Possible AXMEDIS plug-in into Multimedia Players

- AXMEDIS plug-in into
- Adobe Photoshop (DSI)
- Adobe Acrobat Reader (DSI)
- AXMEDIS plug-in into Macromedia tools (ILABS)

8.4.2 Analyzed plug-in technologies

Niagara (EPFL)

Niagara Streaming Systems is a component set that enables for the live video capturing and streaming over the Internet. The system consists of a hardware card and an associated software package. The card captures the audio and video and can perform real-time encoding. The software controls the hardware and allows the internet broadcasting. The system is integrated with the RealNetworks and the Microsoft Media encoding softwares and is compatible with common streaming formats.

Since AXMEDIS does not focus on streaming this system does not fit the best with the AXMEDIS architecture. In addition it seems rather impossible to develop suitable plug-ins through which it may be possible to send/receive specific commands and or data, especially because functionality is rather implemented in hardware. Additional tools for video acquisition/manipulation may be investigated if needed, fitting better into the overall AXMEDIS concept.

Adobe Premiere for video capture, editing (EPFL)

Adobe Premiere is a video editing tool. Premiere is used by video producers to manipulate video content. Premiere is a tool oriented to professional users in contrast to Windows Movie Maker. This software is designed to allow the use of third-party plug-ins. These plug-ins are used to enlarge Premiere capabilities. The different plug-ins that Premiere allows are: filters for processing, transition schemes, additional file-format supports, device controls (reduced capability), titling, or VST Audio plug-ins.

For the AXMEDIS project, first it is important to have file-format support plug-ins. An AXMEDIS plug-in can be developed to use an AXMEDIS Objects inside Adobe Premiere. In Adobe terminology, this is an importing plug-in. The plug-in may use the AXMEDIS blocks to extract and decrypt multimedia content that can be used by Premiere –audio, video, and images-. The extracted and decrypted material would be conveyed to the Premiere tool, etc. The complementary of the importing plug-in would be the exporting plug-in. The exporting plug-in would allow e.g. saving the edited movie in AXMEDIS format.

The Premiere plug-in technology does not provide full control over the software (in AXMEDIS it would be desirable to have full control over Premiere to ensure that the AXMEDIS Object and the resources it contains are used according to the rights of the user). However, in the MS Windows version of Adobe Premiere it is possible to apply some external control and to monitor over the tool by intercepting user commands. For instance, it is possible to intercept a "Save as" user command and report it to the AXMEDIS OM. The plug-in can be able to look up the user rights through the AXMEDIS OM and PMS and decide if the user is allowed or not to do a "Save as".

The Adobe Premiere Pro software development kit (SDK) contains examples of plug-in implementations. The documentation recommends writing new plug-ins by modifying a sample plug-in. To program an importing plug-in one can start by modifiying the sample named SDK_File_Import. This sample consists of a few source code files and a project file for VisualC++ .NET. The output of this project is a dll library renamed as prm. This prm file has to be copied to the plug-in directory of Adobe Premiere; the software loads all the plug-ins located in this directory when the application is started. The software calls the function **xImportEntry** to communicate with the plug-in. See the declaration of **xImportEntry**:

```
xImportEntry (
      int              selector,
      imStdParms       *stdParms,
      long             param1,
      long             param2)
```

The **selector** variable is the action Premiere wants the importer plug-in to perform.

stdParams provides callbacks to obtain additional information from Premiere or to have Premiere perform tasks.

Parameters param1 and param2 vary with the selector; they may contain a specific value or a pointer to a structure.

Some examples of the selector variable are:

- **imInit**: Sent during application startup.
- **imImportAudio**: the plug-in has to give Premiere the specified amount of audio data in the format specified.
- **imImportImage**: the plug-in has to give Premiere a frame of video by populating a buffer.

A simplified **xImportEntry** implementation would look like this:

```
DllExport xImportEntry (int selector, ...)
{
      switch (selector)
      {
            case imInit:
                  //initialization of the plug-in
                  break;
            case imImportImage:
                  //decode image file and write raw output to a buffer
                  break;
            case imImportAudio:
                  //decode audio file and write raw output to buffer
                  break;
            case imOpenFile:
                  //open file and return handle
                  break;
            case imCloseFile:
                  break;
      }
      return OK;
}
```

When the user clicks on the menus of Premiere Windows messages are sent to the application. These messages are processed by a window procedure. These messages can be intercepted by subclassing the Premiere window or, what is the same, by changing the window procedure. The following snipped of code shows how to do window subclassing.

```
//old window proc
long g_oldWndPrc = 0;
//window handle
HWND g_hWnd = 0;

//find window handle of Premiere window
g_hWnd = FindWindow(PREMIERE_MAIN_WINDOW, NULL);

//substitute Premiere window proc by our window proc
g_oldWndPrc = SetWindowLong(g_hWnd, GWL_WNDPROC, (long)WindowProc);

//our window proc
LRESULT CALLBACK WindowProc(
  HWND hwnd,
  UINT uMsg,
  WPARAM wParam,
  LPARAM lParam)
{
    if(uMsg == BUTTON_MENU)
      {
             //intercept only a few messages
            if(wParam == SAVE || wParam == SAVE_A_COPY || wParam == SAVE_AS)
             {
                   MessageBox(g_hWnd, "Save Message Intercepted", "", MB_OK);
                   return 0;
             }
       }
       //let the other messages to be processed normally
      return CallWindowProc((WNDPROC)g_oldWndPrc, g_hWnd, uMsg, wParam, lParam);
}
```

Interaction with Adobe Premiere

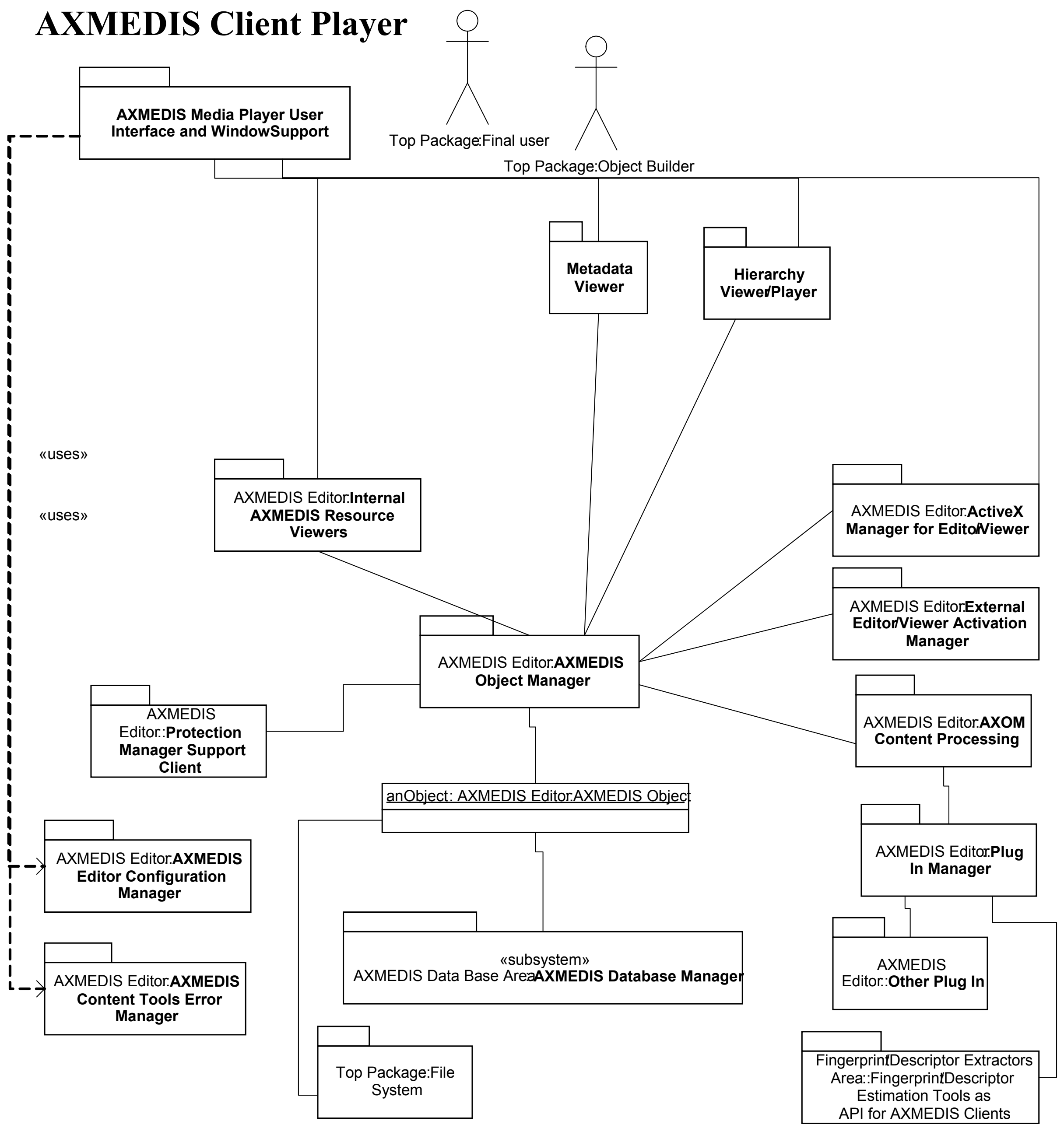
Sending commands for content processing	NO
Sending inhibitions, controls of DRM	In Windows, when the user clicks on a menu, a message is sent to the application. This message can be intercepted by using “windows sublcassing” as explained above. One way of doing inhibition is by intercepting and not releasing the desired windows messages.
Sending data, receiving back processed data	NO
Receiving logs (event reporting)	Some events can be logged by using the same “windows subclassing” method explained above. When, for instance, the user clicks on the Open File menu, a message is sent to the application. This message can be intercepted and logged and then released.

Microsoft Windows Media Encoder (EPFL)
The Windows Media Encoder Software Developer Kit is a set of libraries compiled as automation servers. This SDK can be used to encode multimedia content –mostly audio and video- in Windows compatible formats. The SDK is designed to be usable by script languages like Visual Basic Script or high level languages like C++. It also provides support for streaming and DRM management.
Rather than a plug-in technology, the Windows Media Encoder SDK is a library used by the AXMEDIS Editor. The Windows Media Encoder 9 Series SDK can be used by the AXMEDIS Editor to encode multimedia content in Windows compatible formats in form of internal components.

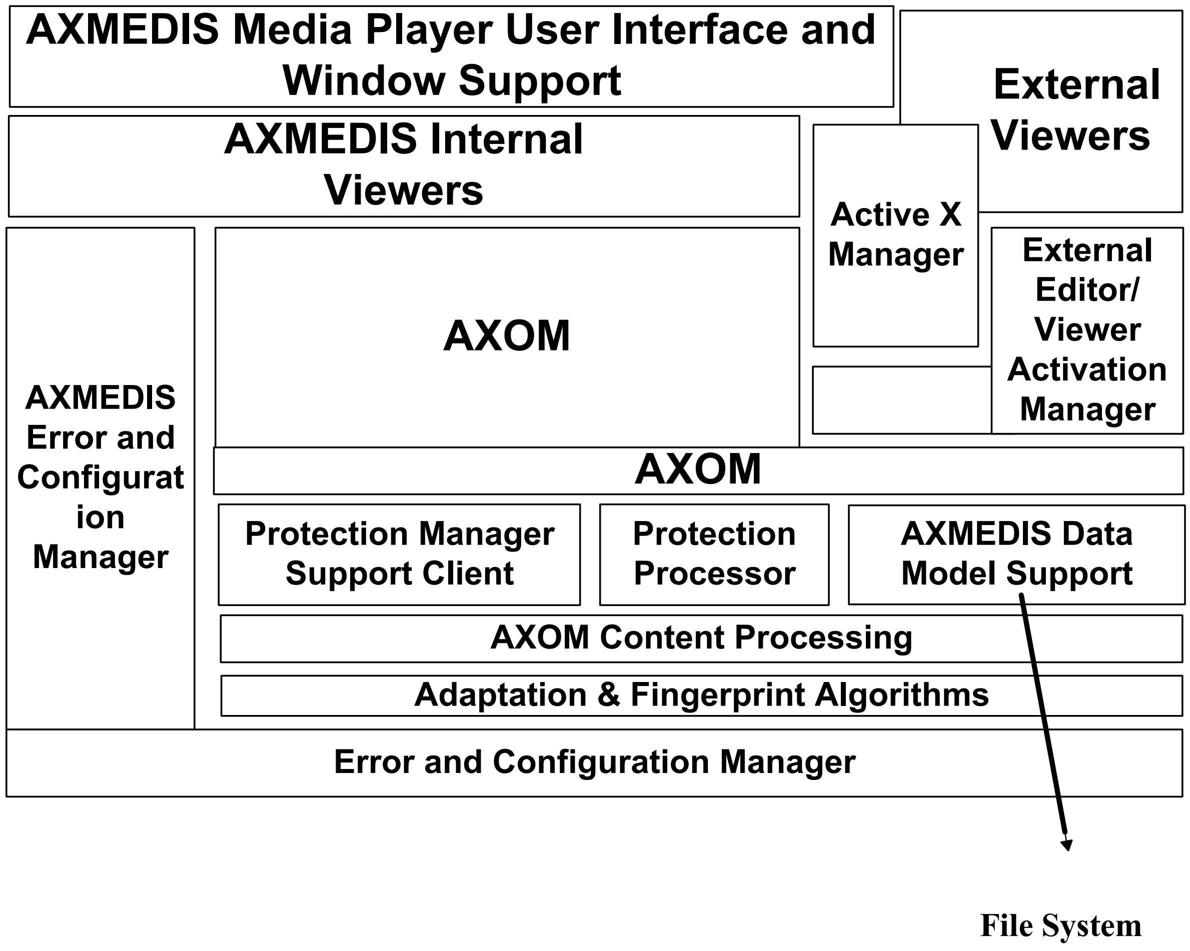
Windows Movie Maker (EPFL)
Windows Movie Maker is a simple video editing tool for home/end users. It is free and it is delivered so far together with Windows XP operating system. The tool allows the user to perform all the basic video editing operations: import a video into the tool, create scene transitions, add subtitles, add credits, cut scenes, and save the resulting video in windows compatible formats. The same functions are available for the sound-track of the film.
Up to the last version 5.1, that has been investigated, the Windows Movie Maker does not allow the creation of a plug-in to communicate with AXMEDIS other blocks. Plug-ins can be developed only for additional effects, transitions etc. (simple processing functions).
Windows Movie Maker is not an automation server and thus it cannot be controlled by another application through an ActiveX interface. This means that Movie Maker cannot receive commands for processing, cannot receive inhibitions, and cannot report event logs.

9 AXMEDIS Players (EPFL, DSI)

Although AXMEDIS focuses on content production, a reduced number of players (and MPEG-21 terminals) will be implemented in order to provide to the AXMEDIS Framework tools to be downloaded by end users to play back AXMEDIS Objects. This may be required in cases when the target user is a consumer (B2C) and especially when target platforms are such kind of devices that an Editor or other kind of sophisticated tool is not supported or envisaged due to either computational, or functional or display capabilities/limitations: such kind of tools may include PDAs, Mobiles, devices with embedded Multimedia DSP processors, etc. In the following tool specifications for PC and similar platforms are reported (first phase); then, specifications for a few mobile or lightweight devices which may be selected for the second phase (WP 4.1.4) will be briefly discussed, keeping in mind that a precise description of some of these aspects are rather difficult to define without the above mentioned selection of the devices themselves.


AXMEDIS Client Player
AXMEDIS Media Player User Interface and WindowSupport
Top Package:Final user
Top Package:Object Builder
Metadata Viewer
Hierarchy Viewer/Player
«uses»
«uses»
AXMEDIS Editor:Internal AXMEDIS Resource Viewers
AXMEDIS Editor:ActiveX Manager for Editor/Viewer
AXMEDIS Editor:External Editor/Viewer Activation Manager
AXMEDIS Editor:AXMEDIS Object Manager
AXMEDIS Editor::Protection Manager Support Client
AXMEDIS Editor:AXOM Content Processing
anObject: AXMEDIS Editor:AXMEDIS Object
AXMEDIS Editor:Plug In Manager
AXMEDIS Editor:AXMEDIS Editor Configuration Manager
«subsystem» AXMEDIS Data Base Area:AXMEDIS Database Manager
AXMEDIS Editor::Other Plug In
AXMEDIS Editor:AXMEDIS Content Tools Error Manager
Top Package:File System
Fingerprint/Descriptor Extractors Area::Fingerprint/Descriptor Estimation Tools as API for AXMEDIS Clients

AXMEDIS Client Player Software Architecture


AXMEDIS Media Player User Interface and Window Support
External Viewers
AXMEDIS Internal Viewers
Active X Manager
External Editor/ Viewer Activation Manager
AXMEDIS Error and Configuration Manager
AXOM
AXOM
Protection Manager Support Client
Protection Processor
AXMEDIS Data Model Support
AXOM Content Processing
Adaptation & Fingerprint Algorithms
Error and Configuration Manager
File System

9.1 AXMEDIS PC Player (DSI, EPFL)

Module Profile		
AXMEDIS PC Player		
Executable or Library(Support)		
Single Thread or Multithread		
Language of Development		
Responsible Name		
Responsible Partner		
Status (proposed/approved)		
Platforms supported		
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)

File Formats Used	Shared with	File format name or reference to a section
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

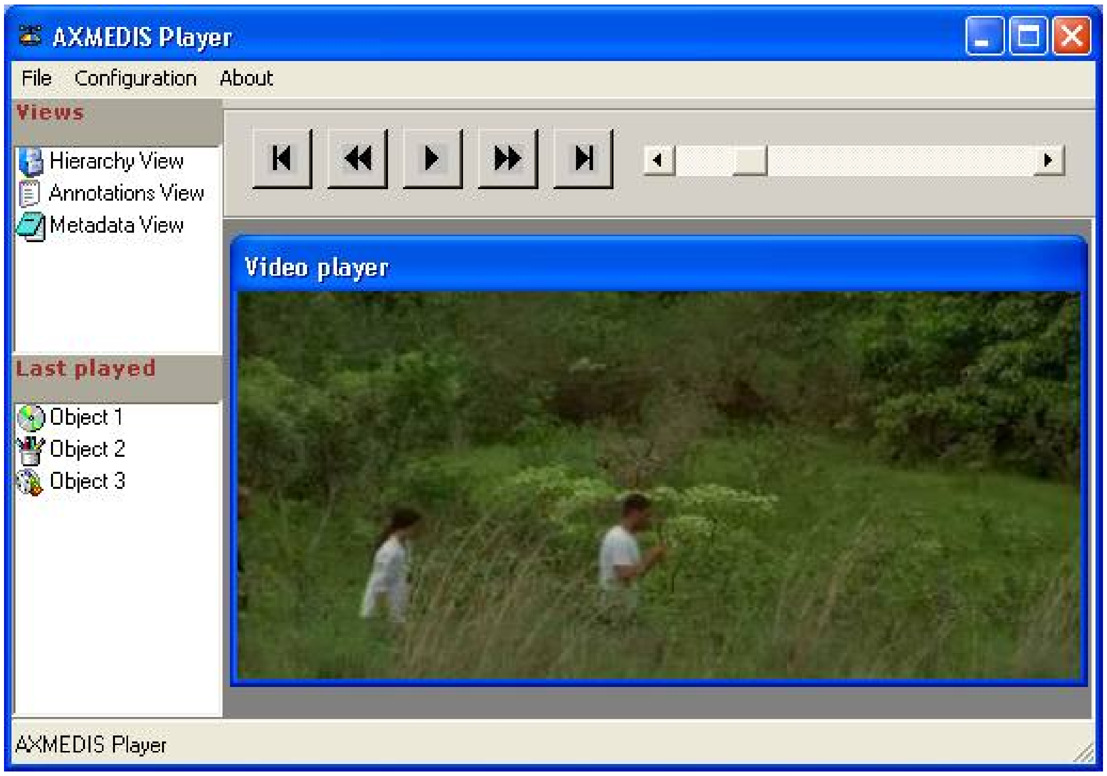
The AXMEDIS Player for PC platform has a strong architectural relationship with the AXMEDIS Editor, in the sense that, being the Editor mainly targeted at WindowsOS and MacOS for their established platforms and tools, the Player functionality may be seen as a subset of the Editor functionality; this subset only allows playback according to DRM AXMEDIS rules and interaction according to the same rules and to the rules embedded in the multimedia composited scenes. The Player may nevetheless allow, for his limited capabilities (in comparison to the Editor), a simpler porting to other platforms like e.g. Linux OS. At the same time, the Player may support additional media formats for which playback-only code or libraries are available. The AXMEDIS Player for PC is in relationship with the content distribution on PC (over the internet). Exact requirements, issues, specifications for this are also expected by discussion with Tiscali at the meeting.

The AXMEDIS Player user interface should work through usual devices such as screen, mouse, joystick, while the usage of the keyboard should be limited to a minimum; it should also work through touch screens, if these are available. Interaction is needed when this is specifically described and coded in interactive multimedia scenes such as those implemented by MPEG-4, VRML, Flash and the like.
The AXMEDIS Player contains an XML-parsing block able to parse AXMEDIS Objects and selects additional tools necessary for the protection management and playback of the different media components (???, see Editor for additional details ???). The AXMEDIS Player shall respect the protection of AXMEDIS objects, by means of the Protection Support module.
The AXMEDIS Player shall allow Digital Content playback for several formats (WAV, MP3, MPEG-2/4, AVC, AVI, PDF, etc…); this will be mainly accomplished through decoders and media players available as source code and/or precompiled libraries and SDKs: media decoders which are not already available are to be considered outside the scope of AXMEDIS (???). Then, the AXMEDIS Player shall use external applications/ActiveX for Digital Content playback or it will provide interfacing mechanisms for communicating with external applications in form of API.
The AXMEDIS Player also contains a module to communicate item adaptation requests to the media provider, whenever this may be necessary for terminal limitations or overloading conditions.

9.1.1 AXMEDIS Player GUI

The AXMEDIS Player GUI for PC may look like the picture below. The user will be able to select among the different views: Hierachy View, Annotations View, and Metadata View. These views will show the Hierachy Editor, the Annotations Editor, and the Metadata Editor. However, these editors will not permit the user to modify the AXMEDIS object, only to view it. This is because the AXMEDIS Player is meant to play AXMEDIS objects while the AXMEDIS Editor is used to edit AXMEDIS objects.
To open an AXMEDIS object the user will click the File menu and then Open or he will drag and drop the AXMEDIS file.

The Configuration menu will be used to tune the behaviour of the Player according to the user preferencs. The user will be able to specify the default view to use when an AXMEDIS object is loaded, default volume, the types of content allowed to reproduce –for instance disallowing the reproduction of content rated as Restricted- etc
The AXMEDIS player will use the AXMEDIS internal Players to render the different types of content – audio, video, text, and SMIL presentations.
If a single AXMEDIS object contains multiple media contents the user will have to use the Hierachy Viewer to select which content to reproduce.



9.1.2 AXMEDIS PDA Player (EPFL, DSI)

Module Profile		
AXMEDIS PDA Player		
Executable or Library(Support)	Executable	
Single Thread or Multithread		
Language of Development	C/C++	
Responsible Name	Giorgio Zoia	
Responsible Partner	EPFL	
Status (proposed/approved)	Proposed	
Platforms supported	Windows PocketPC	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a

		section
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

The AXMEDIS Player may be ported on at least one of the most widespread PDA device with well supported OS and platform.

In the case of PDAs, many of the existing and well supported devices are running on PalmOS or Windows (WindowsCE and PocketPC) operating systems. Whereas Palm usually stress on application design (most Palm applications are designed specifically and exclusively for the mobile use, in order to provide best fit for user needs) WindowsCE derives huge Windows application base, concentrating on conversion of existing applications. In the case of AXMEDIS, since the first platform and version of the Player will be developed during the first phase for WindowsOS, the choice will most probably be a limited porting of the Player software to a PocketPC device; some of the AXMEDIS partners are for instance familiar with the iPaq device.

As mentioned, in terms of development the most characterizing issue is the limitation of resources due to device constraints. A second additional issue is adaptation of the functionality to a rather different interface and interaction devices.

It is difficult at this time to provide an exact specification for the AXMEDIS Player in PDA, but in fact some guidelines can be stressed since now. In terms of platform features, current off-the-shelf PDAs (PocketPCs) can be characterized as follows:

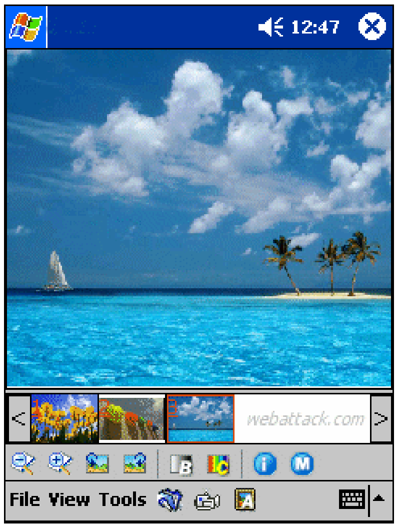
- Operating system: PalmOS or Windows Mobile
- Processor: 320 to 624 MHz clock speed (Intel, StrongARM…)
- Display: from CIF 64k colors to 4" Transflective type VGA 64K colors,
- Memory: 64MB to 192 MB total memory (ROM and SDRAM in different ratios)
- Audio: Integrated microphone, speaker and one 3.5 mm stereo headphone/headset jack, MP3 stereo through audio jack and speaker, sometimes 5-band equalizer for playback through audio jack
- Other typical features: 4 shortcut programmable buttons, vavigation Touch-pad, touch-sensitive display for stylus or fingertip, voice record button.

In terms of Player functionality and user interaction, it appears that the most limiting factors will be for the overall processing power and screen size and resolution. The second issue partially or mainly masks limits due to the first as it is hardly conceivable to play multiple audiovisual objects at the same time in a CIF or VGA screen. Implementation of Player views may also be limited by graphic aspects. Nevertheless the AXMEDIS Player will allow digital content playback for several audiovisual formats (AAC, MP3, MPEG-1/4, AVC, AVI, etc…) at the most convenient profiles and levels, including scalable codecs and as far as possible computational graceful degradation for complex bitstreams. In addition to already existing or announced codecs and tools, PocketPC allows fast creation of some applications, because it supports converting existing Windows applications to pocket platform. PocketPC development requires Microsoft tools: the most important Windows APIs are available for Windows CE/PocketPC, sometimes in limited form. For this, regular MSDN library is used for reference. For each API or feature, MSDN articles explain differences or limitations on Windows CE/PocketPC, so portability issues can be considered already under development of the PC Player.

For development, Windows CE edition of the Visual Studio package is needed. The package allows compiling an application for Windows CE/PocketPC and testing it on PC in emulated environment (it is not an emulation of the device, it is instead another compiler for PC with APIs redirected).

Portability of the AMBULANT or equivalent SMIL Player to PDA platform will be investigated in a successive phase. Indeed some porting is announced under development already, as it is the case for the MPEG-4 Player.

The AXMEDIS Player user interface should work almost exclusively through touch devices limiting to the minimum interaction via keys or other interfaces. This will require a certain reworking in comparison to the PC Player. A typical GUI for a PocketPC viewer is in the following picture



The AXMEDIS Player will be also capable to respect the playback protection of AXMEDIS objects, as contained the content and decoded and dealt with through the MPEG-21/AXMEDIS client terminal.

9.2 AXMEDIS Mobile Player (EPFL, COMVERSE, DSI)

The AXMEDIS Player will be ported on at least one of the most widespread mobile devices with well supported hardware and software platform. At the moment it is not clear how far will a mobile device (smartphone like) from a PDA during the second phase of AXMEDIS in terms of functionality, but we can safely assume a device characterized by an even more reduced display size and computational resources, even if in comparable overall functionality.

The issue of porting the overall AXMEDIS Player concept to a Mobile platform is even more challenging than in the case of PDA, it requires important compromises and an attentive investigation of the actual possibilities, as resources are in this case extremely limited in comparison even to a PDA and development almost surely does not allow simple porting of good portions of code.

As for PDAs some possibilities for adaptation exist; one possibility is to address Smartphones based on Windows Mobile operating system; another possible solution is to use a SymbianOS device. In the case of mobile devices it is in any case necessary to plan a considerable rework of the player and cutoff in functionality, so the fact of remaining in the Windows environment may not constitute an advantage anymore. In terms of platform features, current off-the-shelf Smartphones can be characterized as follows:

- Operating system: Symbian or Windows Mobile
- Processor: 120 to 200 MHz clock speed
- Display: in the order of 176x220 pixels, 64K colors
- Memory: 32MB to 96 MB total memory (ROM and SDRAM in different ratios)
- Multimedia: Integrated microphone, speaker and one 3.5 mm stereo headphone/headset jack, digital camera, MP3 player, etc.

In terms of Player functionality and user interaction, even more than for the PDA the most limiting factors will be for the processing power and screen size and resolution.
The second issue masks limits due to the first as it is not conceivable to play multiple audiovisual objects at the same time on such a small display. In addition, it may be hard to have one audiovisual content decoded in real time so that multiple objects may result impossible for yet another reason.
Implementation of Player may be limited by graphic aspects and by a single view allowing simple access and control of audiovisual content and its metadata. The AXMEDIS Player will allow digital content playback for several audiovisual formats (AAC, MP3, H263, H264/AVC, etc…) at the most convenient profiles and levels, including low complexity codecs that may be available at the moment of development. Porting of a SMIL player is probably not convenient, if useful at all, as simple synchronization of one audiovisual object should be enough and no complex multimedia synchronization can be necessary or achieved.

In the case of Windows Mobile, a Mobile Application Development Toolkit provides development utilities and easy integration and simulation inside Visual Studio .NET 2003 and the .NET Compact Framework environment. In the case of SymbianOS, the SDK provides both native development in C++ as well as inside PersonalJava runtime, including JavaPhone APIs. Choice of one of the two platforms may be based on different factors but mainly availability of suitable players for the most important media object types used by partners in AXMEDIS for distribution.

The AXMEDIS Player user interface should work almost exclusively through a limited number of keys and/or interaction devices (pointing devices, joystick-like devices, etc.). This will imply some rework of the GUI.
The AXMEDIS Player will also be capable to respect the playback protection of AXMEDIS objects, exploiting some management strategy that may be adapted for the mobile platform.

9.3 AXMEDIS Tablet PC Player for School Bag on Mozilla (SEJER)

This AXMEDIS Player which can run on a TabletPC is based on the internet browser provided with this technology: Mozilla. The AXMEDIS objects are rendered with AXMEDIS plug in for Mozilla.