Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE3.1.2.2.5
Specification of External Editors, Viewers, and Players first update of DE3.1.2 part B

Version: 2.3
Date: 21-08-2006
Responsible: DSI (EPFL up to the July 2006 meeting)

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

Deliverable Number: DE3.1.2.2.5
Contractual Date of Delivery: M18
Actual Date of Delivery: 21-08-2006
Title of Deliverable: Specification of External AXMEDIS Editors/Viewers and Players, first update of DE3.1.2 part B
Work-Package contributing to the Deliverable: WP3.1
Task contributing to the Deliverable: WP3, WP2
Nature of the Deliverable: report
Author(s): DSI, EPFL, SEJER, ILABS, TISCALI

Abstract: this part includes the specification of components, formats, databases and protocol related to the AXMEDIS Framework area including External AXMEDIS Editors/Viewers and Players

Keyword List: External Editor/Viewer Activation Manager, AXMEDIS PC player, AXMEDIS PDA player, AXMEDIS Mobile player, AXMEDIS Tablet PC Player, Active X Control, plug in.

Table of Content

1 Executive Summary and Report Scope

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

DE number	Deliverable title	responsible
DE3.1.2.2.1	Specification of General Aspects of AXMEDIS framework, first update of DE3.1.2 part A AXMEDIS-DE3-1-2-2-1-Spec-of-AX-Gen-Asp-of-AXMEDIS-framework-upA-v1-0.doc	DSI
DE3.1.2.2.2	Specification of AXMEDIS Command Manager, first update of DE3.1.2 part B AXMEDIS- DE3-1-2-2-2-Spec-of-AX-Cmd-Man-upB-v1-0.doc	DSI
DE3.1.2.2.3	Specification of AXMEDIS Object Manager and Protection Processor, first update of DE3.1.2 part B AXMEDIS-DE3-1-2-2-3-Spec-of-AXOM-and-ProtProc-upB-v1-0.doc	DSI
DE3.1.2.2.4	Specification of AXMEDIS Editors and Viewers, first update of DE3.1.2 part B AXMEDIS-DE3-1-2-2-4-Spec-of-AX-Editors-and-Viewers-upB-v1-0.doc	DSI
DE3.1.2.2.5	Specification of External AXMEDIS Editors/Viewers and Players, first update of DE3.1.2 part B AXMEDIS-DE3-1-2-2-5-Spec-of-External-Editors-Viewers-Players-upB-v1-0.doc	EPFL
DE3.1.2.2.6	Specification of AXMEDIS Content Processing, first update of DE3.1.2 part C AXMEDIS-DE3-1-2-2-6-Spec-of-AX-Content-Processing-upC-v1-0.doc	DSI
DE3.1.2.2.7	Specification of AXMEDIS External Processing Algorithms AXMEDIS-DE3-1-2-2-7-Spec-of-AX-External-Processing-Algorithms-v1-0.doc	FHGIGD
DE3.1.2.2.8	Specification of AXMEDIS CMS Crawling Capabilities, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-8-Spec-of-AX-CMS-Crawling-Capab-v1-0.doc	DSI
DE3.1.2.2.9	Specification of AXMEDIS database and query support, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-9-Spec-of-AX-database-and-query-support-v1-0.doc	EXITEC H
DE3.1.2.2.10	Specification of AXMEDIS P2P tools, AXEPTool and AXMEDIS, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-10-Spec-of-AXEPTool-and-AXMEDIA-tools-v1-0.doc	DSI with its subcontra ctor
DE3.1.2.2.11	Specification of AXMEDIS Programme and Publication tools, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-11-Spec-of-AX-Progr-and-Pub-tool-v1-0.doc	UNIVLE EDS
DE3.1.2.2.12	Specification of AXMEDIS Workflow Tools, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-12-Spec-of-AX-Workflow-Tools-v1-0.doc	IRC
DE3.1.2.2.13	Specification of AXMEDIS Certifier and Supervisor and networks of AXCS, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-13-Spec-of-AXCS-and-networks-v1-0.doc	DSI
DE3.1.2.2.14	Specification of AXMEDIS Protection Support, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-14-Spec-of-AX-Protection-Support-v1-0.doc	FUPF
DE3.1.2.2.15	Specification of AXMEDIS accounting and reporting, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-15-Spec-of-AX-Accounting-and-Reporting-v1-0.doc	EXITEC H

1.1 This document concern

DE3.1.2.2.5 Specification of External AXMEDIS Editors/Viewers and Players, first update of DE3.1.2 part B

This document includes the specification of a set of players and tools of AXMEDIS and in particular of:

- **EXEVAM**: a system and tools for activating external players and viewers in connection with the AXMEDIS Editor. The connection is with a specific protocol. The mechanism also includes the presence of an AXMEDIS plug in into the external Player to guarantee the DRM aspects.
- **AXMEDIS player based on Active X**, it includes AXOM and all the other internal players of AXMEDIS and the PMS client. This permits the realization of AXMEDIS player based on Internet Explorer and other tools such as those of Macromedia: Authorware, etc.
- **AXMEDIS plug in of Mozilla,** it includes AXOM and all the other internal players of AXMEDIS and the PMS client. This permits the realization of AXMEDIS player based on Mozilla Browser.
- **AXMEDIS Plug in into Multimedia Players**: an plug in of AXMEDIS including AXOM, and PMS client to allow enforcing DRM aspects into third party Players.
- **AXMEDIS PDA player**. A player capable of playing quite complex AXMEDIS objects based on video, audio, images, documents, etc., and also with MPEG-4 and SMIL inside. Please note that both OSMO MPEG-4 player and SMIL AMBULANT player will be used for realizing this player porting them in the Pocket Pc 2003 platform. For the support of AXMEDIS model the AXOM library and related tools will be ported, to this end, a reduction of functionalities is foreseen.
- **AXMEDIS Mobile player**: An AXMEDIS player with low foot print for mobiles phone capable of playing simple resources, mainly single Video or single Audio files with related metadata. For the support of AXMEDIS model the AXOM library and related tools will be ported, to this end, a reduction of functionalities is foreseen.
- **AXMEDIS player for Tablet PC**: an AXMEDIS player based on AXMEDIS Mozilla Plug in presented before.

This deliverable, initially in charge of EPFL (up to July 2006), has been taken in charge of the DSI (coordinator) to solve some pending aspects and reallocating some of the activities among them: the specification and the realization of the AXMEDIS players for PDA with the related profiling and thus of porting of libraries.

1.2 List of Modules or Executable Tools Specified in this document

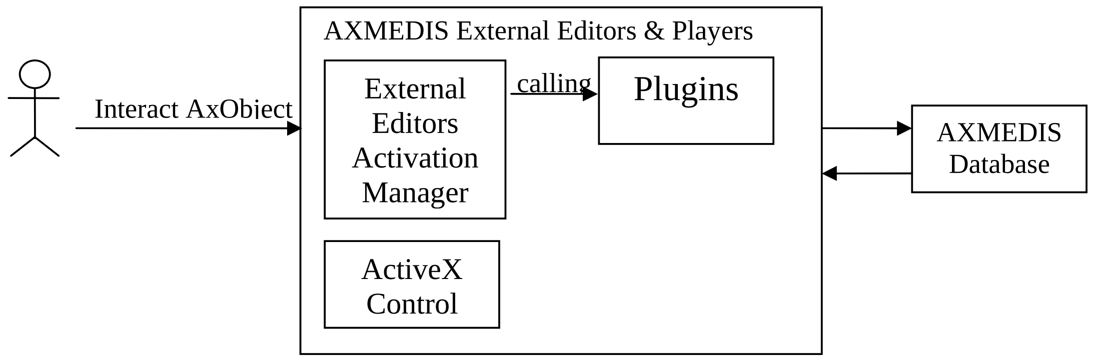
A module is a component that can be or it is reused in other cases or points of the AXMEDIS framework or of other AXMEDIS based solutions.
The modules/tools have to include effective components and/or tools and also testing components and tools.

Module/tool Name	Module/Tool Description and purpose, state also in which other AXMEDIS area is used	Standards exploited if any
External Editor/Viewer Activation Manager, EXEVAM	enables AXMEDIS Editors the capability of viewing/modifying resources by using external application which have not ActiveX/COM interface;	
AXMEDIS ActiveX Control	enables the plug of an AXMEDIS player inside any compliant executable	
AXMEDIS plug-in into Mozilla	enable to rendering of AXMEDIS content inside Mozilla's web browser and let the user interact with it.	

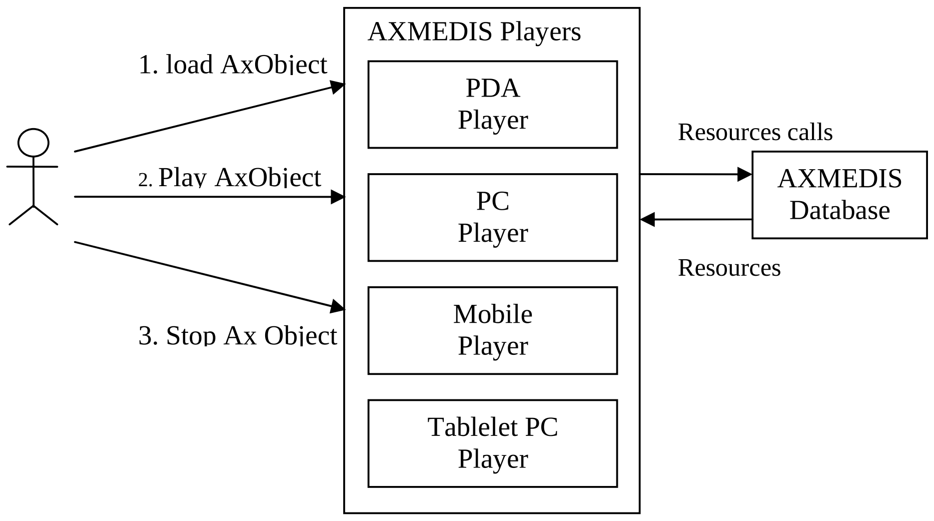
AXMEDIS plugin into Multimedia Player	Enable to direct rendering the resource of AXMEDIS content by other multimedia players	
AXMEDIS PC Player	enable to rendering of AXMEDIS content on PC and let the user interact with it.	W3C XML MPEG21 DIDL W3C SMIL 2.0 MPEG-4 Systems, MPEG-4 IPMPX
AXMEDIS PDA Player	enable to rendering of AXMEDIS content on PDA and let the user interact with it.	W3C XML MPEG21 DIDL W3C SMIL 2.0 MPEG-4 Systems, MPEG-4 IPMPX
AXMEDIS Mobile Player	enable to rendering of AXMEDIS content on Mobile Devices and let the user interact with it.	W3C XML MPEG21 DIDL W3C SMIL 2.0 MPEG-4 Systems, MPEG-4 IPMPX
AXMEDIS Tablet PC Player for School Bag on MozillA	enable to rendering of AXMEDIS content on Tablet PC Player and let the user interact with it.	

2 General Use Cases and scenarios

2.1 Use Case Activate external editors and players

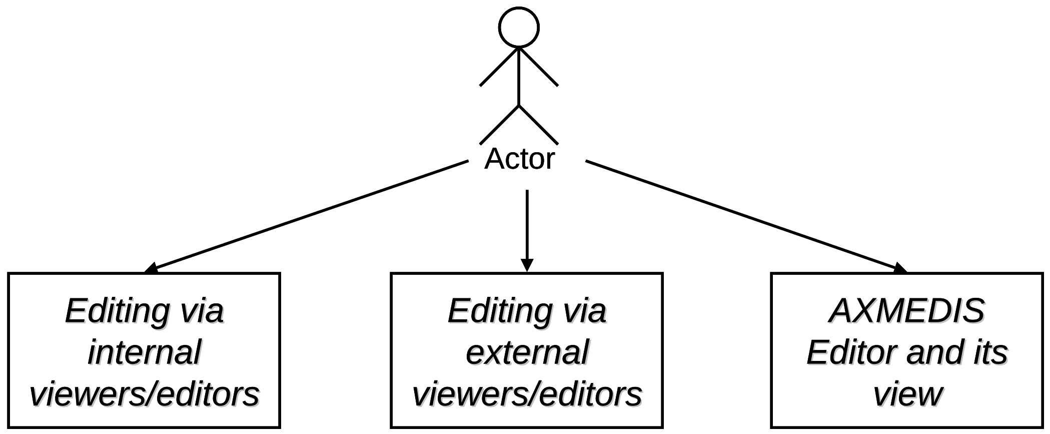

AXMEDIS External Editors & Players
Interact AxObject
External Editors Activation Manager
calling
Plugins
ActiveX Control
AXMEDIS Database

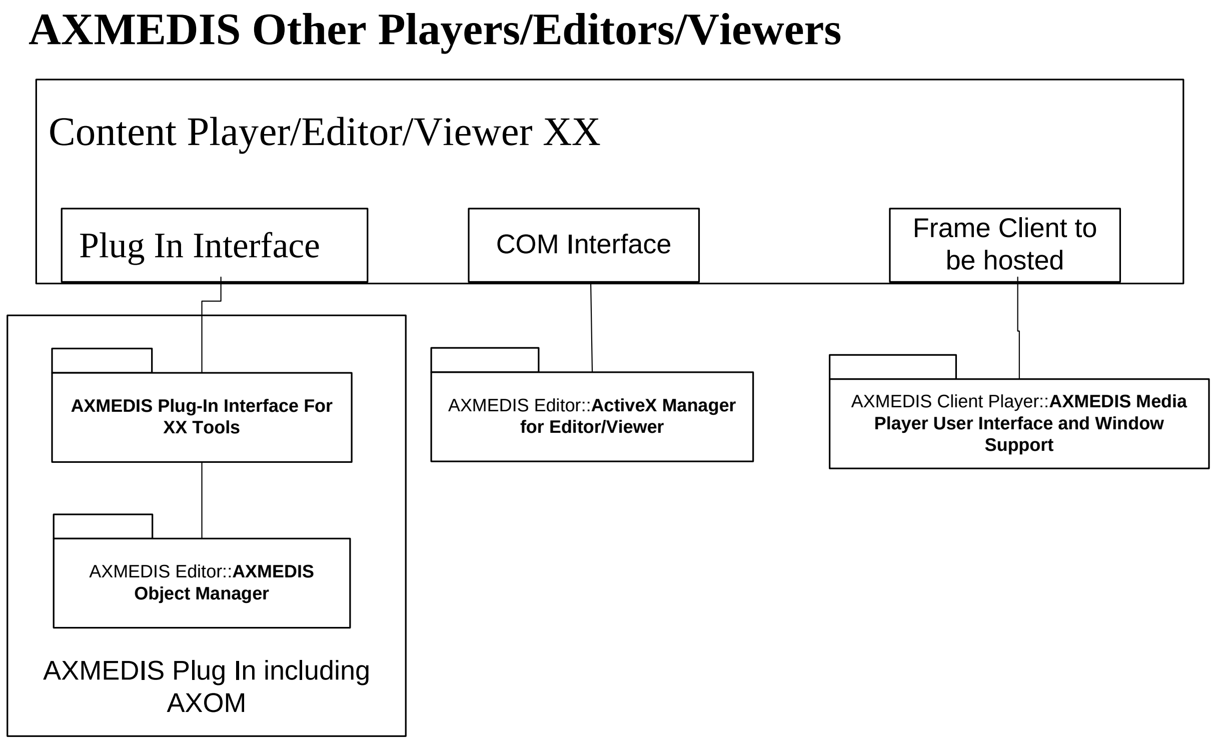
2.2 Use Case Activate different platform based AXMEDIS players



3 General architecture and relationships among the modules produced

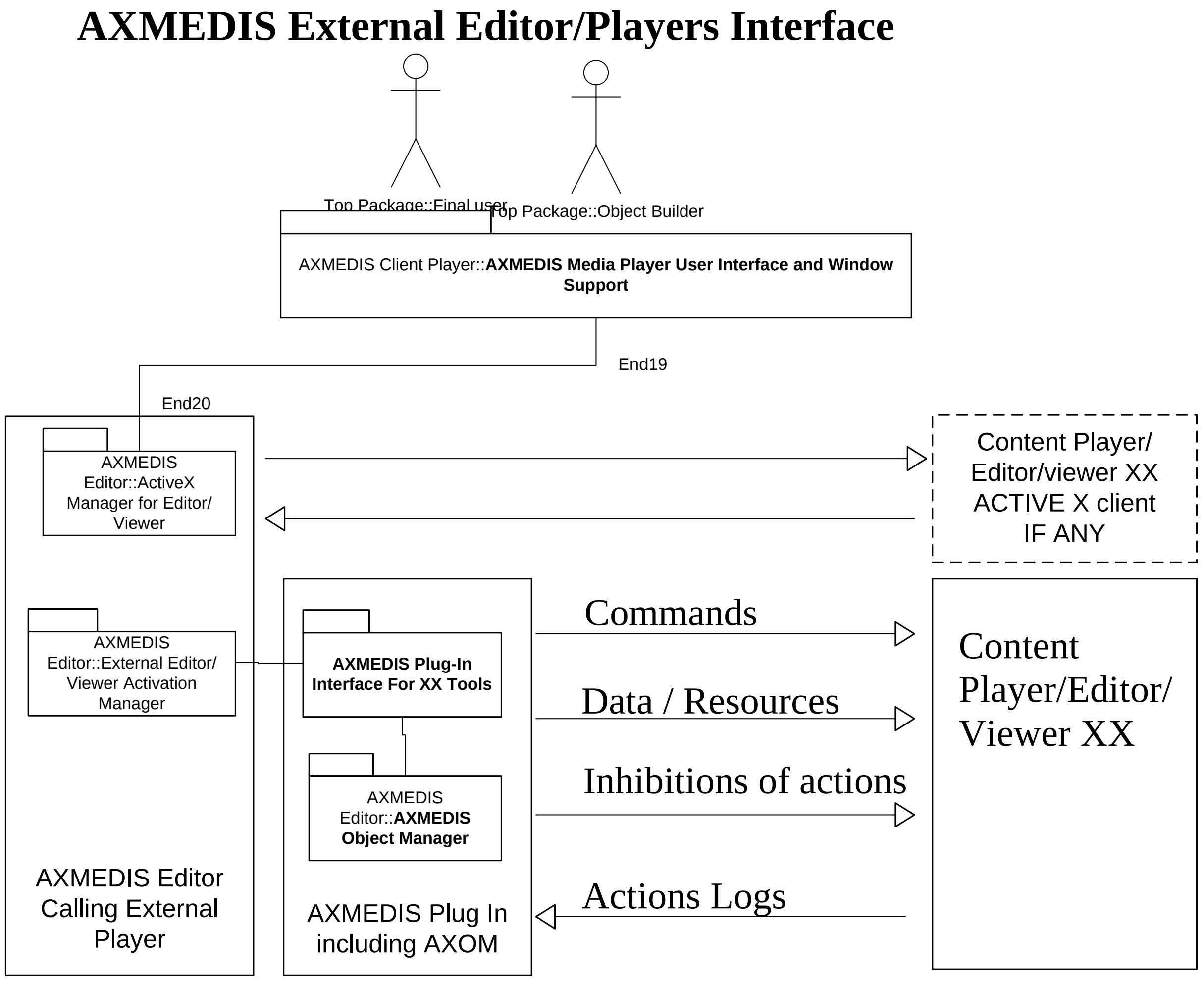
This document concerns with the scenario for the interoperability of the AXMEDIS Editor with other commonly used tools for content production; in the External Editor/Viewer AXMEDIS plug-ins the capability of activating other tools directly from the AXMEDIS Editor is specified. The specification details how the AXMEDIS editor can master the communication with the external tools and how the DRM and the security requirements are preserved in the data transfer.
The plug-in analysis is also carried out 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.


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


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). 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.

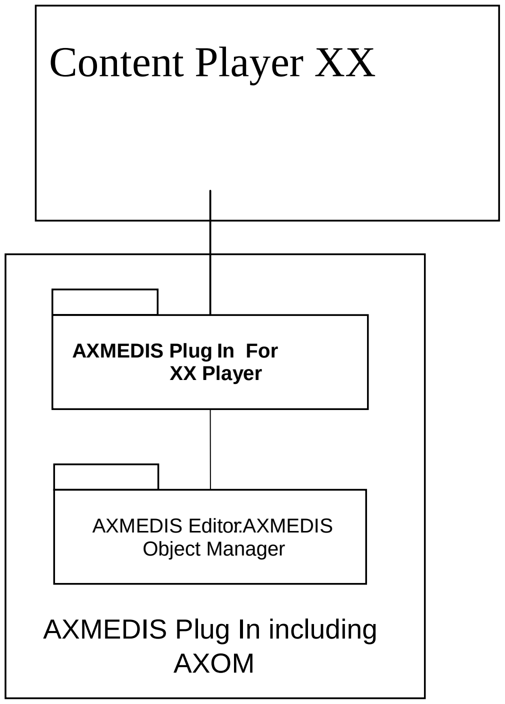

AXMEDIS External Editor/Players Interface
Top Package::Final user
Top Package::Object Builder
AXMEDIS Client Player::AXMEDIS Media Player User Interface and Window Support
End19
End20
AXMEDIS Editor::ActiveX Manager for Editor/ Viewer
Content Player/ Editor/viewer XX ACTIVE X client IF ANY
AXMEDIS Editor::External 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

3.1 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

3.1.1 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:

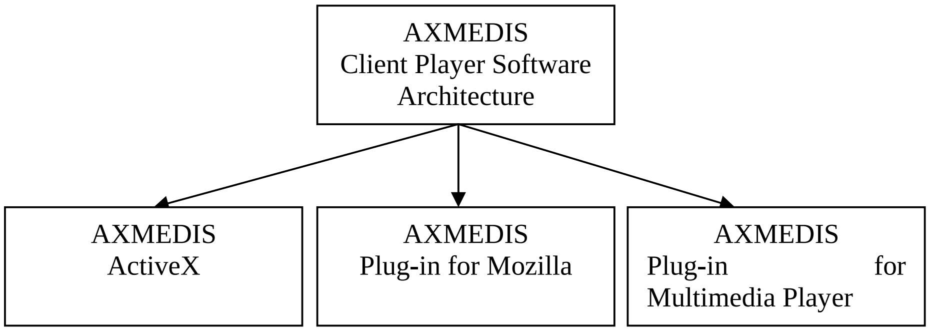
- browsers like IE, Mozilla etc,
- multimedia players like WindowsMedia technology etc…

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:

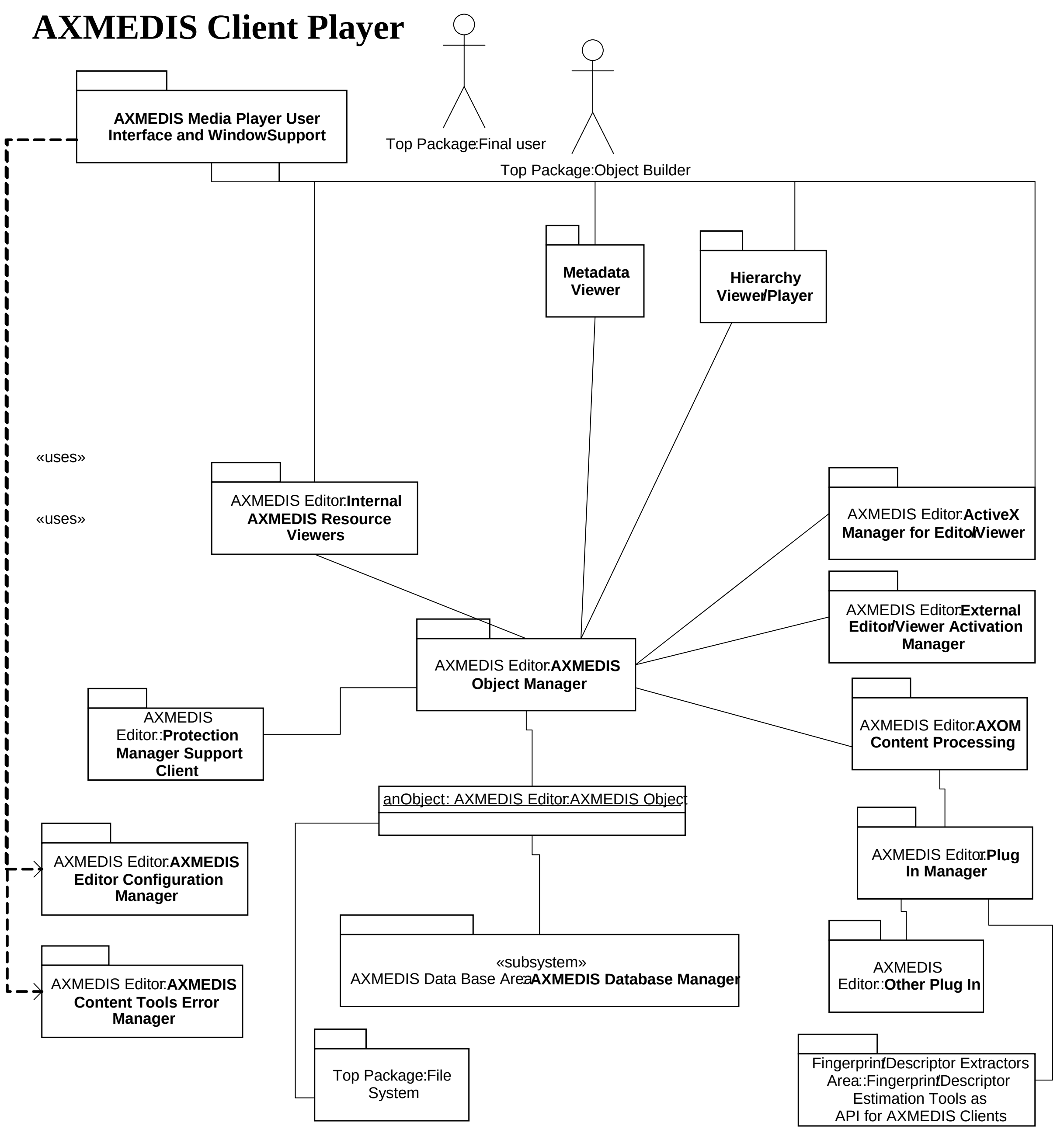
1. AXMEDIS ActiveX
2. AXMEDIS Plug-in for Mozilla
3. AXMEDIS Plug-in for Multimedia Players

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

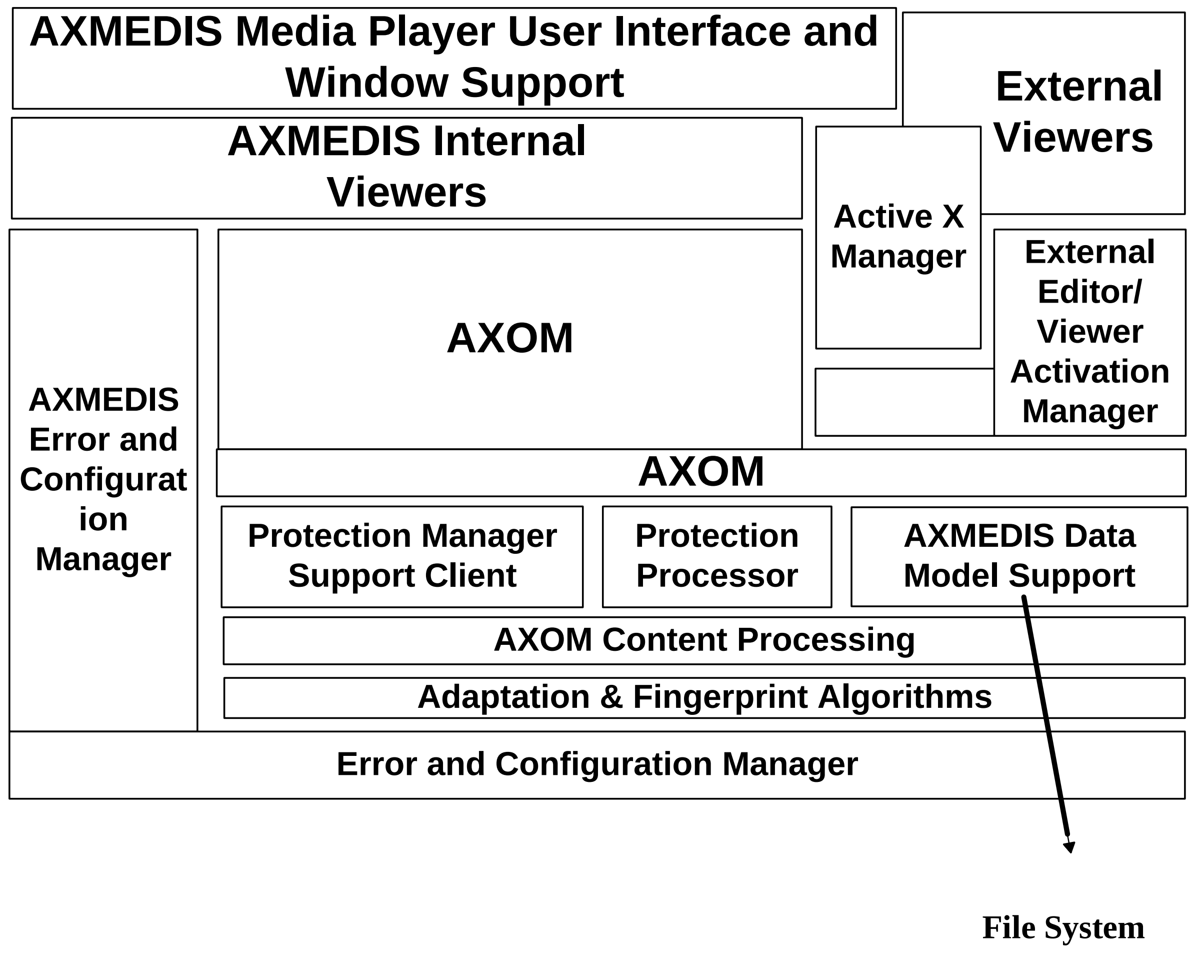

AXMEDIS
Client Player Software Architecture
AXMEDIS
ActiveX
AXMEDIS
Plug-in for Mozilla
AXMEDIS
Plug-in for Multimedia Player

3.2 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
AXOM
AXMEDIS Error and Configurat ion Manager
AXOM
Protection Manager Support Client
Protection Processor
AXMEDIS Data Model Support
AXOM Content Processing
Adaptation & Fingerprint Algorithms
Error and Configuration Manager
File System

4 Module - External Editor/Viewer Activation Manager, EXEVAM, (DSI)

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

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

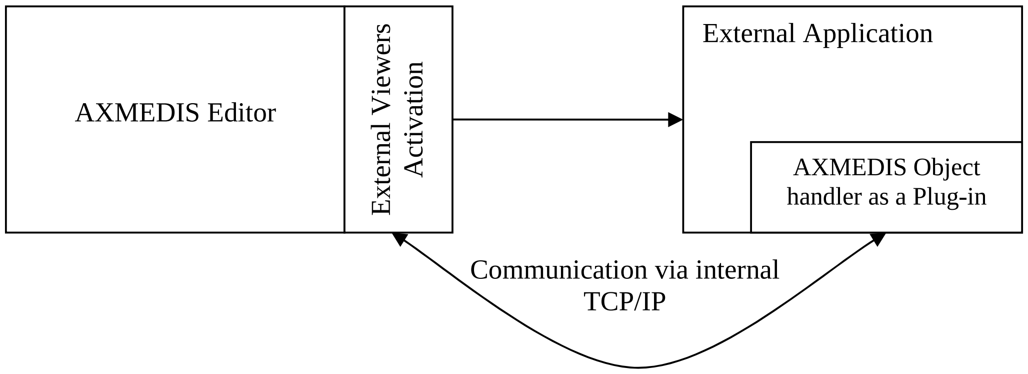
4.1 General Description of the Module

main purpose and description, major functionalities, etc.

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 contact 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.

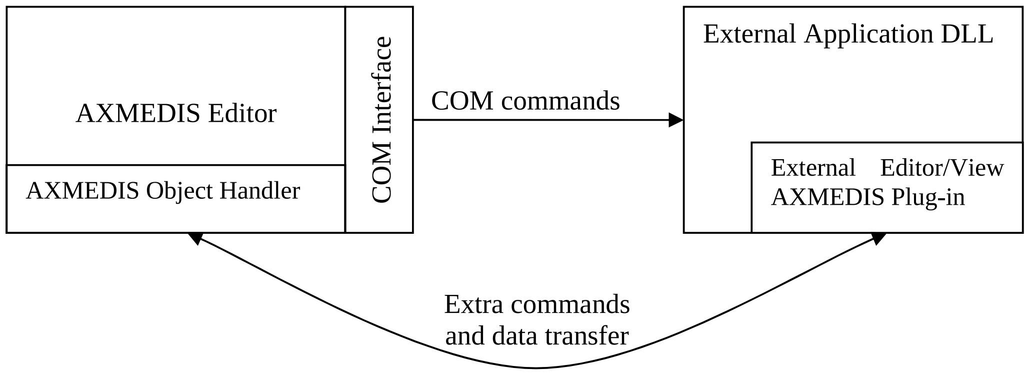

AXMEDIS Editor
External Viewers Activation
External Application
AXMEDIS Object handler as a Plug-in
Communication via internal TCP/IP

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.

4.1.1 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

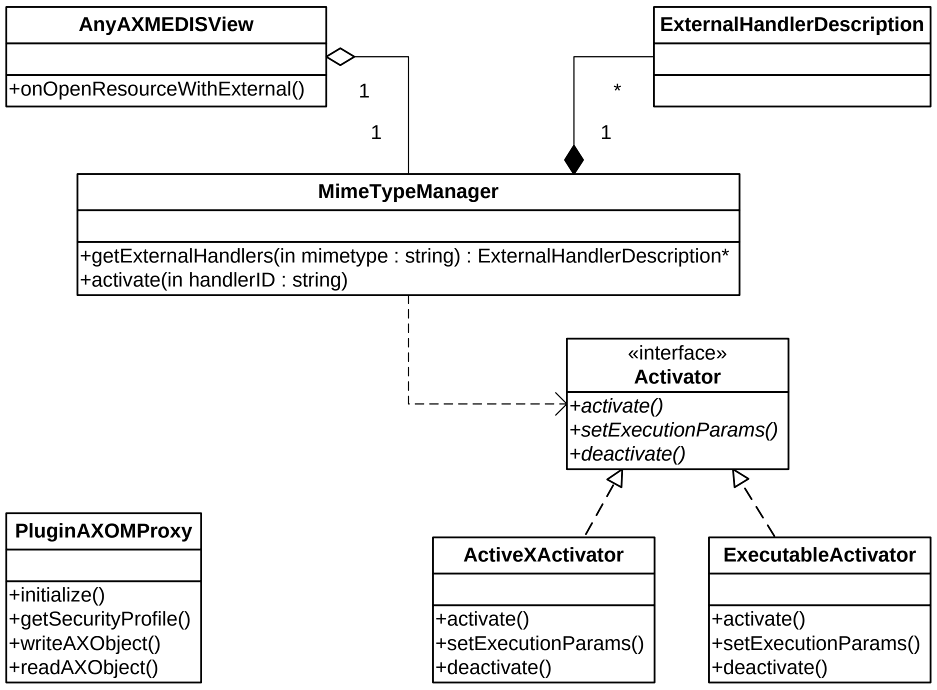


The communication channel is not a strong assumption: it is possible to arrange different solutions in order to make data available on both sides. The security does not depend on the communication channel, but is maintained at the required level with the communication between the two AXOM instances.

4.1.2 EXEVAM 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.

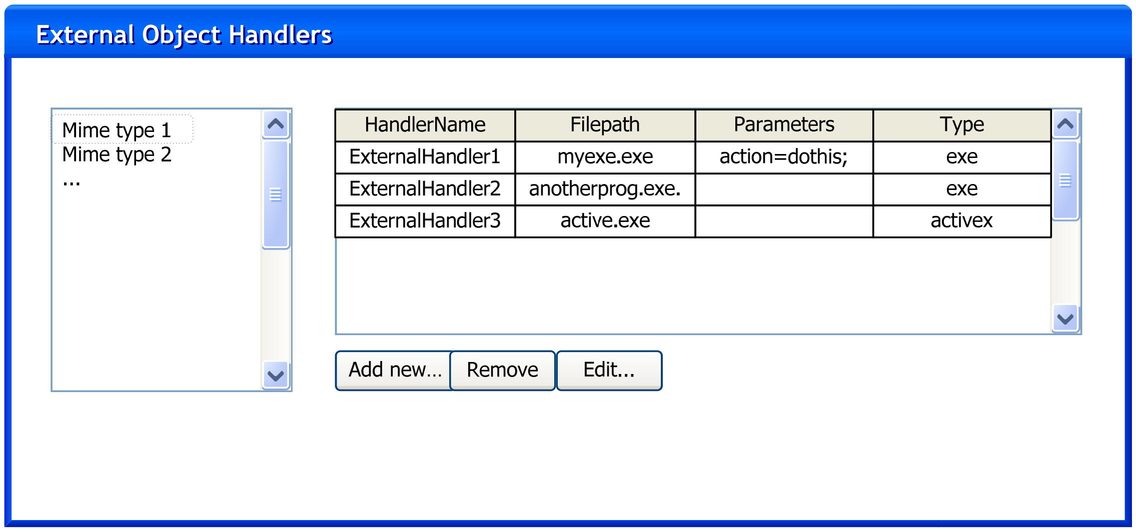


From any view in the AXMEDIS editor a resource of an AXMEDIS object could be opened with an external "handler" (with handler both cases of independent executable processes and ActiveX controls have been modelled).

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.



Mime type 1
Mime type 2
…

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:

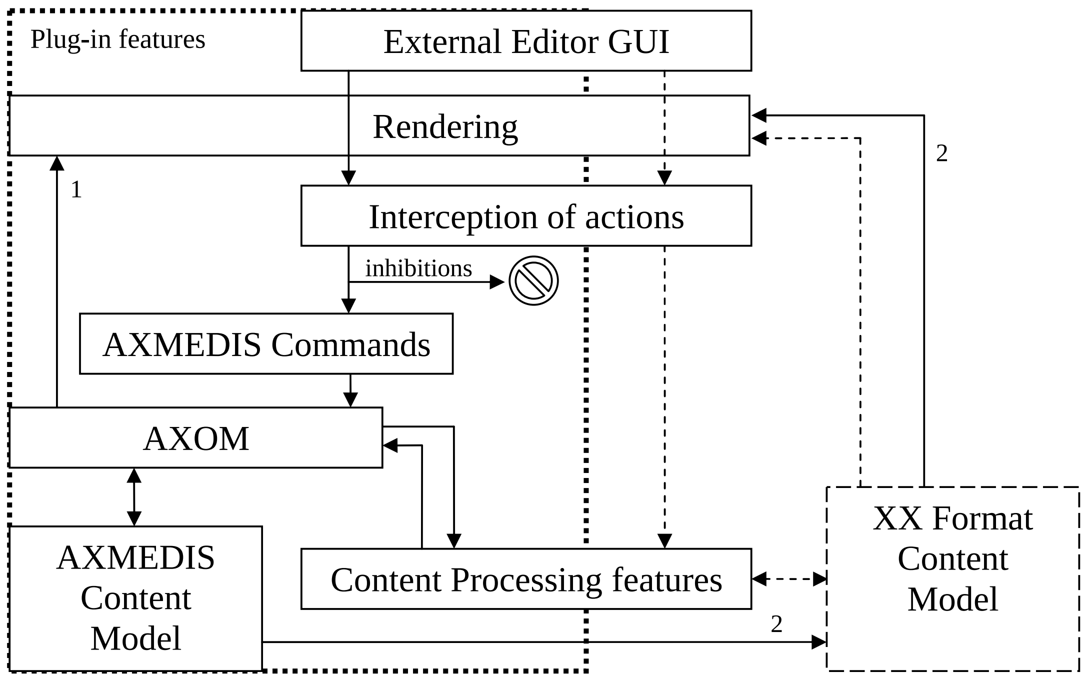
- basically the ExecutableActivator realizes the Activator interface to implement a console command activation of the desired executable a class has been conceived in order to support different platforms;
- the ActiveXActivator implements the activation via COM mechanism and host the ActiveX control 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.

4.1.3 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.

Another strategy could be to simply instantiate a "empty" command which has only the purpouse of exercising the right against the Authorization Support. Than the usual editor action can be commanded without violating the resource license.

4.1.4 Communication and localization of plug-in counter-parts

Some basic functionalities are included in the transport mean of information regarding security, content, commands, synchronization. A SOAP protocol can be used to allow AxEditor and AXMEDIS Plug-in of an editor to communicate. A list of well-known port exposed by the plug-in for different editors can be produced. The AxEditor can acquire such information form an external repository or the configuration can be changed at installation time of a certain plug-in.

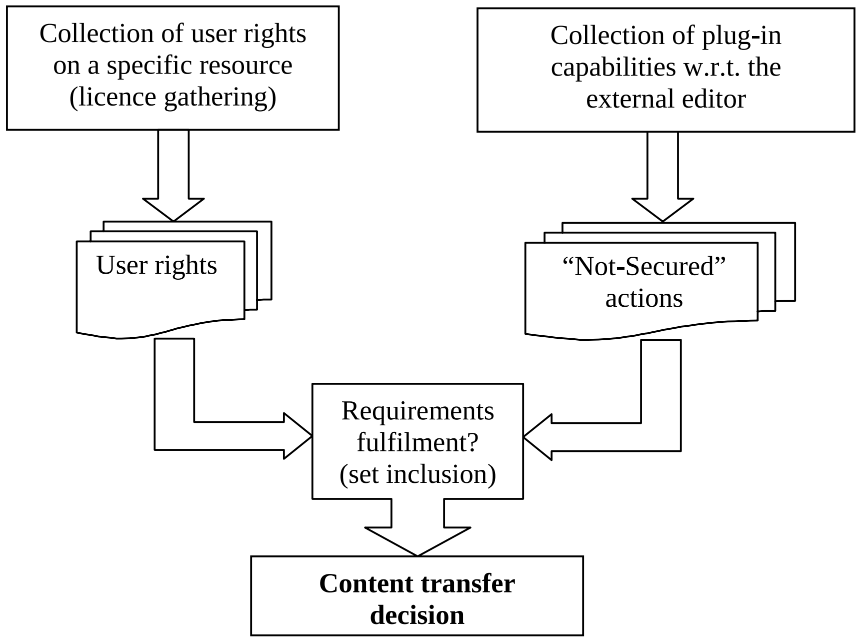
Since basically resources or object subparts will be passed to external editor. The protected content can be passed via file, packaging it in the present protection status. Since AXOM is present in both applications,

4.1.5 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 plug-in capabilities w.r.t. the external editor
User rights
"Not-Secured" actions
Requirements fulfilment? (set inclusion)
Content transfer decision

5 Module - AXMEDIS ActiveX Control (DSI)

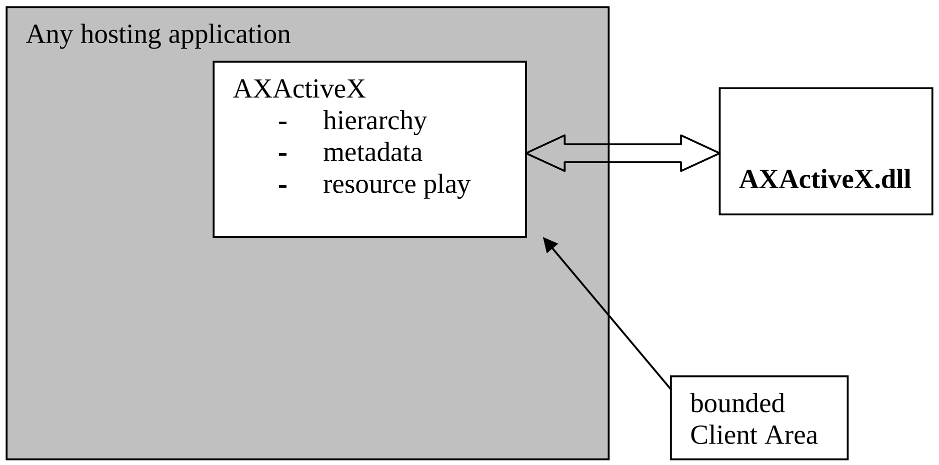
Module/Tool Profile		
AXMEDIS ActiveX Control		
Responsible Name	Bellini	
Responsible Partner	DSI	
Status (proposed/approved)		
Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)	DLL library (OCX)	
Single Thread or Multithread	Multithread	
Language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

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

5.1 General Description of the Module

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.

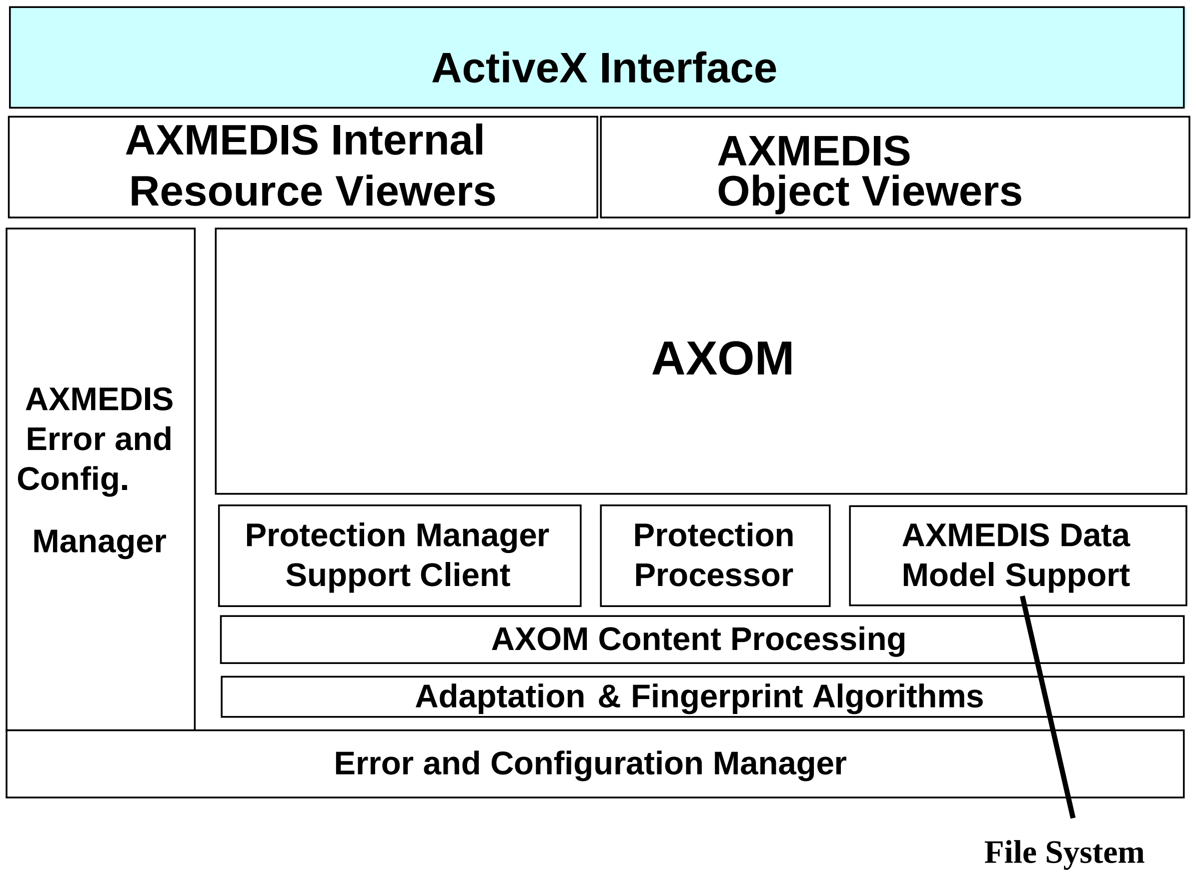
In the main client view of the target application the AXMEDIS ActiveX can be rendered in a specific bounded client area. The control takes the responsibility of render information and manage user interactions regarding such an area.



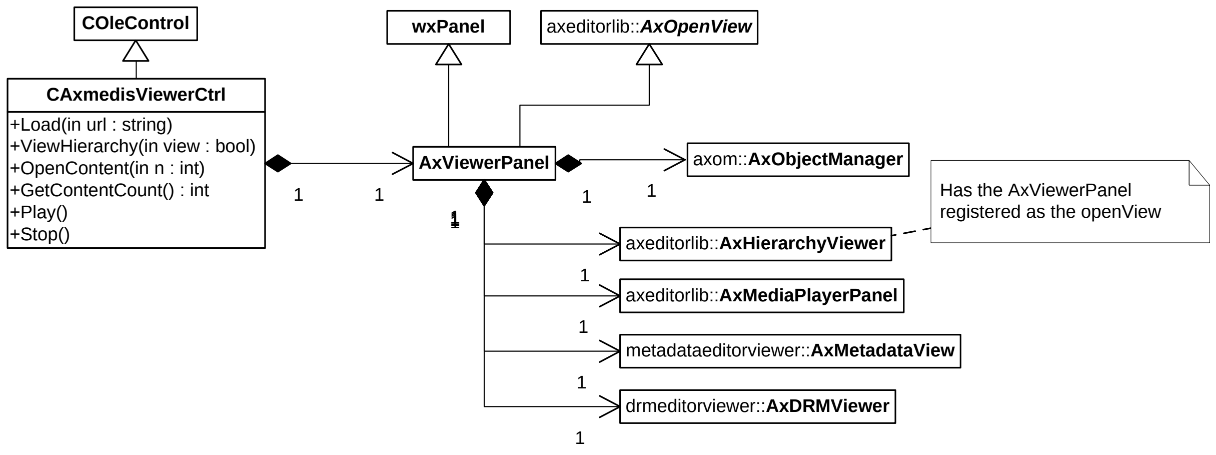
Since the AXMEDIS ActiveX has to render itself on the screen portion it needs:

- 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

5.2 Module Design in terms of Classes


COleControl
CAxmedisViewerCtrl
+Load(in url : string)
+ViewHierarchy(in view : bool)
+OpenContent(in n : int)
+GetContentCount() : int
+Play()
+Stop()
wxPanel
axeditorlib::AxOpenView
AxViewerPanel
axom::AxObjectManager
Has the AxViewerPanel registered as the openView
axeditorlib::AxHierarchyViewer
axeditorlib::AxMediaPlayerPanel
metadataeditorviewer::AxMetadataView
drmeditorviewer::AxDRMViewer
1
1
1
1
1
1
1
1
1

The *CAXMEDISViewerCtrl* class (generated using the MSVisualStudio automation) represents the activeX interface, it uses the *AxViewerPanel* class which implements the real user interface and hosts the *AxObjectManager* which is used to access to the AXMEDIS object to be shown.
The *AxViewerPanel* uses:

- the *AxHierarchyViewer* class to display the hierarchy of the object,
- the *AxMediaPlayerPanel* class to display the resources inside the object,
- the *AxMetadataView* class to display the metadata of the object
- the *AxDRMViewer* class to display the license owned on the object

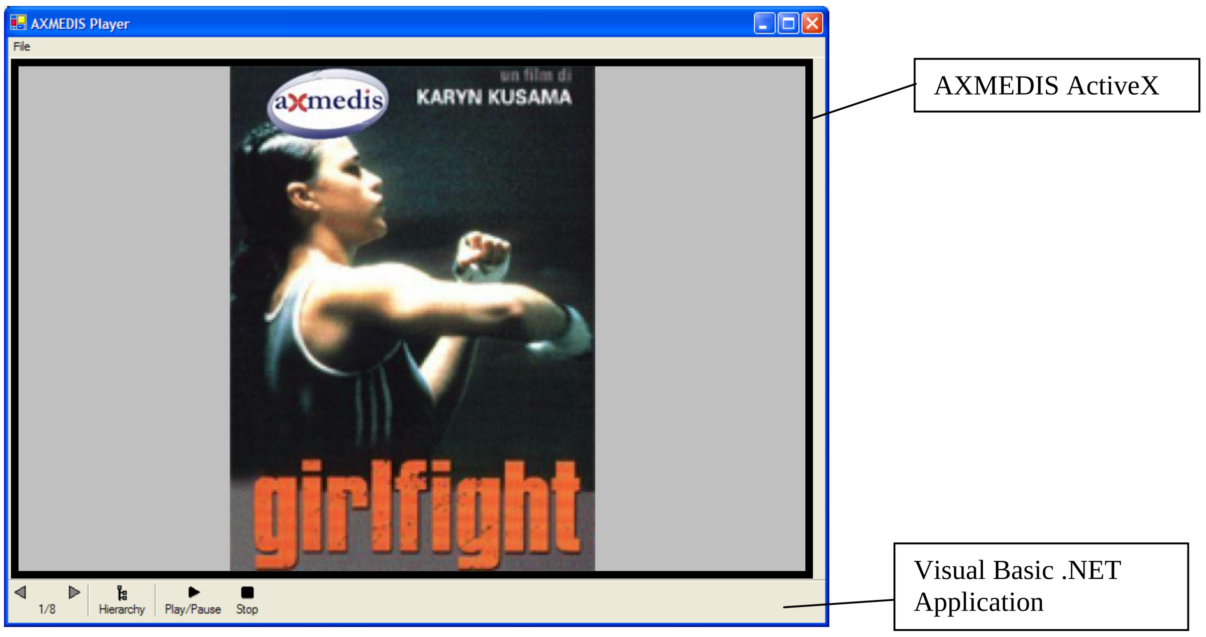
The IDL file defining the ActiveX is:

```
#include <olectl.h>
#include <idispids.h>

[ uuid(C3003391-36FC-4C85-A7FD-3CE0D86E1CF7), version(1.0),
  helpfile("axactivex.hlp"),
  helpstring("Control module for ActiveX axactivex"),
  control ]
library axactivexLib
{
    importlib(STDOLE_TLB);

    //  Dispatch interface for CAXMEDISViewerCtrl

    [ uuid(159FAE48-7BC3-4C10-8F27-772426A51DB2),
      helpstring("Dispatch interface for AXMEDISViewer Control")]
    dispinterface _Daxactivex
    {
        properties:
             [id(5), helpstring("property ViewHierarchy")] VARIANT_BOOL ViewHierarchy;
            [id(6), helpstring("property ContentCount")] SHORT ContentCount;
            [id(10), helpstring("property BackgroudColor")] BSTR BackgroudColor;
            [id(11), helpstring("property Src")] BSTR Src;
            [id(27), helpstring("property SupportTimeControl")] VARIANT_BOOL SupportTimeControl;
            [id(13), helpstring("property IsPlaying")] VARIANT_BOOL IsPlaying;
            [id(15), helpstring("property CurrentTime")] ULONG CurrentTime;
            [id(16), helpstring("property Duration")] ULONG Duration;
            [id(17), helpstring("property StartTime")] ULONG StartTime;
            [id(18), helpstring("property EndTime")] ULONG EndTime;
            [id(28), helpstring("property SupportVisualControl")] VARIANT_BOOL SupportVisualControl;
            [id(22), helpstring("property FullScreen")] VARIANT_BOOL FullScreen;
            [id(25), helpstring("property Zoom")] DOUBLE Zoom;
            [id(26), helpstring("property AutoFit")] VARIANT_BOOL AutoFit;
            [id(30), helpstring("property NeedsLicense")] VARIANT_BOOL NeedsLicense;
            [id(31), helpstring("property HaveLicense")] VARIANT_BOOL HaveLicense;
        methods:
            [id(DISPID_ABOUTBOX)] void AboutBox();
             [id(1), helpstring("method Load")] SHORT Load(BSTR filename);
             [id(4), helpstring("method GetContentType")] BSTR GetContentType(SHORT contentRef);
            [id(34), helpstring("method GetContentMimeType")] BSTR GetContentMimeType(SHORT
contentRef);
```

```
            [id(7), helpstring("method OpenContent")] void OpenContent(SHORT contentRef);
            [id(8), helpstring("method Play")] void Play(void);
            [id(9), helpstring("method Stop")] void Stop(void);
            [id(12), helpstring("method Pause")] void Pause(void);
            [id(14), helpstring("method JumpToTime")] void JumpToTime(ULONG time);
            [id(19), helpstring("method Fit")] void Fit(DOUBLE width, DOUBLE height);
            [id(24), helpstring("method FitToWindow")] void FitToWindow(void);
            [id(20), helpstring("method ZoomIn")] void ZoomIn(DOUBLE ratio);
            [id(21), helpstring("method ZoomOut")] void ZoomOut(DOUBLE ratio);
            [id(23), helpstring("method Print")] void Print(void);
            [id(29), helpstring("method ShowLicense")] void ShowLicense(void);
            [id(32), helpstring("method AcquireLicense")] VARIANT_BOOL AcquireLicense(BSTR licID);
            [id(33), helpstring("method ShowMetadata")] void ShowMetadata(void);
            [id(35), helpstring("method execCommand")] HRESULT execCommand(BSTR cmdID, VARIANT_BOOL
showUI, VARIANT value, VARIANT_BOOL* pfRet);
            [id(36), helpstring("method queryCommandEnabled")] HRESULT queryCommandEnabled(BSTR
cmdID, VARIANT_BOOL* pfRet);
            [id(37), helpstring("method queryCommandIndeterm")] HRESULT queryCommandIndeterm(BSTR
cmdID, VARIANT_BOOL* pfRet);
            [id(38), helpstring("method queryCommandStatus")] HRESULT queryCommandStatus(BSTR cmdID,
VARIANT_BOOL* pfRet);
            [id(39), helpstring("method queryCommandSupported")] HRESULT queryCommandSupported(BSTR
cmdID, VARIANT_BOOL* pfRet);
            [id(40), helpstring("method queryCommandValue")] HRESULT queryCommandValue(BSTR cmdID,
VARIANT* pCmdValue);
    };

    //  Events dispatch interface for CAXMEDISViewerCtrl

    [ uuid(D87580AA-D50A-4CF5-8086-67BA99A92859),
      helpstring("Events interface for AXMEDISViewer Control") ]
    dispinterface _DaxactivexEvents
    {
        properties:
            //  L'interfaccia eventi non ha proprietà

        methods:
            [id(DISPID_CLICK)] void Click(void);
    };

    //  Informations on class CAXMEDISViewerCtrl

    [ uuid(B30B789D-2378-45BD-A1B5-AFC0B43919D3),
      helpstring("AXMEDISViewer Control"), control ]
    coclass axactivex
    {
        [default] dispinterface _Daxactivex;
        [default, source] dispinterface _DaxactivexEvents;
    };

};
```

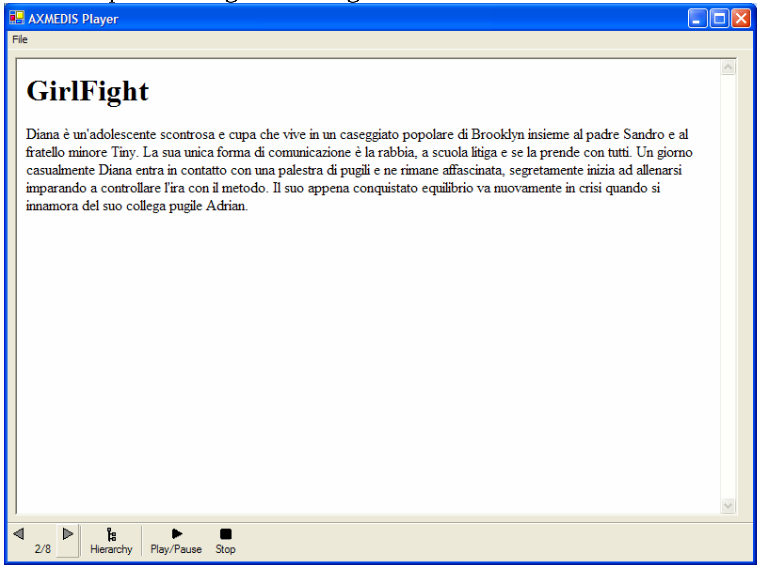
5.3 User interface description

The following pictures show the use of the AXMEDIS ActiveX inside a Visual Basic Application implementing a basic player. The ActiveX user interface does not present any button controls to allow the customization of the user interface, the visual area is used to display the resources.



The Left and Right arrow buttons (defined in the VB application) use the *OpenContent(n)* method of the ActiveX to open a specific content stored in the AXMEDIS object, the Hierarchy button use the *ViewHierarchy* property to view/hide the AXMEDIS Hierarchy, the Play/Stop button use the *Play/Stop* methods to control the execution of Audio/Video resources.

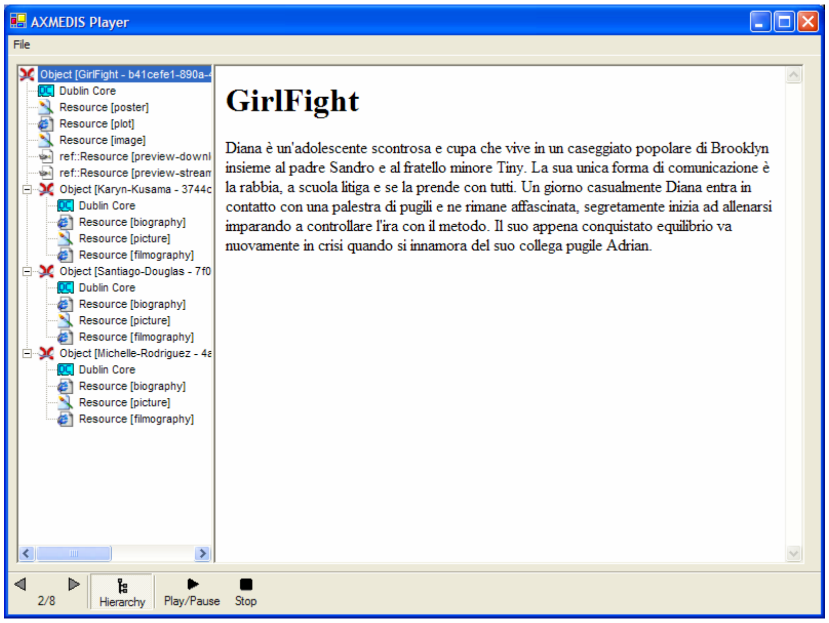
For example clicking on the Right arrow the second resource is shown (the movie plot):



GirlFight

Diana è un'adolescente scontrosa e cupa che vive in un caseggiato popolare di Brooklyn insieme al padre Sandro e al fratello minore Tiny. La sua unica forma di comunicazione è la rabbia, a scuola litiga e se la prende con tutti. Un giorno casualmente Diana entra in contatto con una palestra di pugili e ne rimane affascinata, segretamente inizia ad allenarsi imparando a controllare l'ira con il metodo. Il suo appena conquistato equilibrio va nuovamente in crisi quando si innamora del suo collega pugile Adrian.

and clicking on the Hierarchy Button the hierarchy is shown:

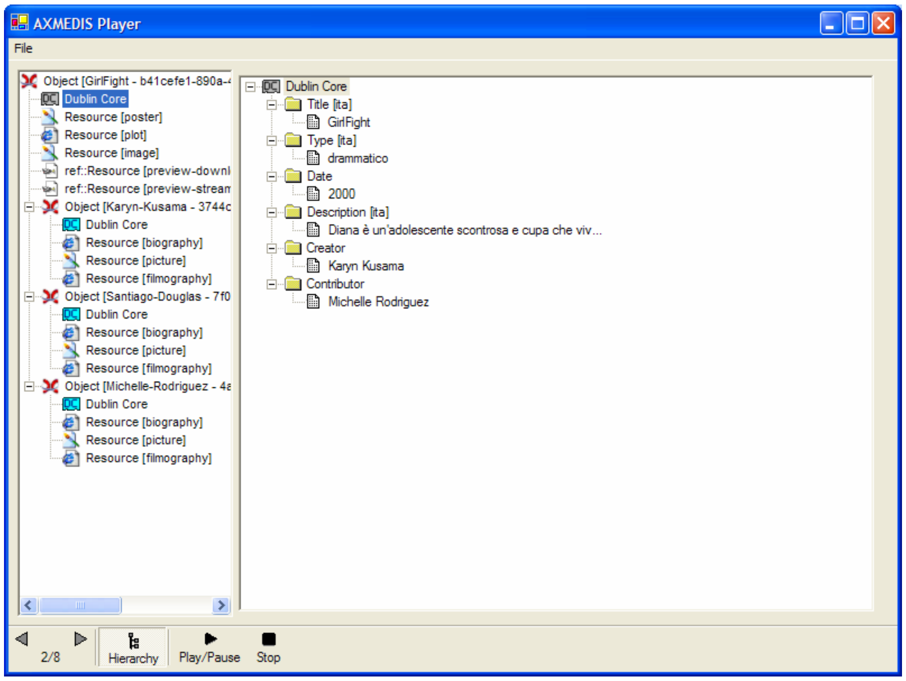


```
Object [GirlFight - b41cefe1-890a-
  Dublin Core
  Resource [poster]
  Resource [plot]
  Resource [image]
  ref::Resource [preview-downl
  ref::Resource [preview-stream
  Object [Karyn-Kusama - 3744c
    Dublin Core
    Resource [biography]
    Resource [picture]
    Resource [filmography]
  Object [Santiago-Douglas - 7f0
    Dublin Core
    Resource [biography]
    Resource [picture]
    Resource [filmography]
  Object [Michelle-Rodriguez - 4a
    Dublin Core
    Resource [biography]
    Resource [picture]
    Resource [filmography]
```

GirlFight

Diana è un'adolescente scontrosa e cupa che vive in un caseggiato popolare di Brooklyn insieme al padre Sandro e al fratello minore Tiny. La sua unica forma di comunicazione è la rabbia, a scuola litiga e se la prende con tutti. Un giorno casualmente Diana entra in contatto con una palestra di pugili e ne rimane affascinata, segretamente inizia ad allenarsi imparando a controllare l'ira con il metodo. Il suo appena conquistato equilibrio va nuovamente in crisi quando si innamora del suo collega pugile Adrian.

Double clicking on the Dublin Core element the Metadata is shown:



```
Dublin Core
  Title [ita]
    GirlFight
  Type [ita]
    drammatico
  Date
    2000
  Description [ita]
    Diana è un'adolescente scontrosa e cupa che viv...
  Creator
    Karyn Kusama
  Contributor
    Michelle Rodriguez
```

5.4 Technical and Installation information

References to other major components needed	AXMEDIS Object Manager AXMEDIS Internal Players AXMEDIS Hierarchy Editor & Viewer AXMEDIS Metadata Editor & Viewer AXMEDIS DRM Editor & Viewer AXMEDIS Object Editor & Viewer
Problems not solved	•

Configuration and execution context	

5.5 Draft User Manual

AXMEDISViewerCtrl

5.5.1 Basic Interface

Method Load	
short Load(string url)	
Description	Loads and AXMEDIS object from an URL or from file system, it tries to open the first SMIL resource available or otherwise the first Resource/AXObject available
Input parameters	• *string url* – the url or file to be opened (e.g. "http://AXMEDIS.org/demo/demo1.axm", "c:\AXMEDIS-objects\test.axm"), protocols supported will be http and ftp,
Return value	returns *0* if the object can be successfully loaded, *1* if the object cannot be loaded, 2 if the content cannot be opened (missing license)
Remarks	In case of load failure an empty object is present
Property Src (read write)	
string Src	
Description	the URL of the object loaded, when set the object is loaded (not opening content if the object is protected)
DefaultValue	""
Property ContentCount (read only)	
short ContentCount	
Description	number of content elements (resources/axobjects) at the first level
DefaultValue	0
Property BackgroudColor (read write)	
string BackgroundColor	
Description	the color to be used as background (e.g. "#ffffff" for white)
DefaultValue	"#ffffff"
Property ViewHierarchy (read write)	
bool ViewHierarchy	
Description	indicates if the AXMEDIS Hierarchy view has to be shown or not.
DefaultValue	false
Method GetContentType	
string GetContentType(short contentPos)	
Description	allows to get which kind of content is present at a specific position (1..ContentCount)
Input parameters	• *short contentPos* – position of the content to be analyzed
Return value	returns *"RESOURCE" or "AXOBJECT"*
Remarks	In case of invalid position returns ""
Method GetContentMimeType	
string GetContentMimeType(short contentPos)	
Description	allows to get the mimetype of content at a specific position (1..ContentCount)
Input parameters	• *short contentPos* – position of the content to be analyzed (1..ContentCount)
Return value	returns a mimetype for a resource and the mimetype of the resource really played in case of an embedded AXMEDIS object (e.g. it can return "application/smil" if the object contains a SMIL resource)
Remarks	In case of invalid position returns ""
Method OpenContent	

void OpenContent(short contentPos)	
Description	allows to open a content at a specific position (0..ContentCount), position 0 means the object itself. If the content to be opened is an AXMEDIS object it opens the first SMIL resource present otherwise it opens the first resource present
Input parameters	• *short contentPos* – position of the content to be opened (0..ContentCount)
Return value	none
Remarks	none
Method ShowMetadata	
void ShowMetadata()	
Description	allows to show the metadata of the whole object (using Dublin Core information)
Input parameters	none
Return value	none
Remarks	none
Method ShowLicense	
void ShowLicense()	
Description	allows to show the license available on the object
Input parameters	none
Return value	none
Remarks	none
Property NeedLicense (read only)	
bool NeedLicense	
Description	Indicates if a license is needed to view the object
DefaultValue	false
Property HaveLicense (read only)	
bool HaveLicense	
Description	Indicates if is available a license to view the object
DefaultValue	false
Method AcquireLicense	
void AcquireLicense(string licenseID)	
Description	allows to acquire locally a license given the license ID (if it is allowed)
Input parameters	• *string licenseID* – contains the license identification code
Return value	none
Remarks	The method is used to preload in the local cache the license allowing to use it offline even using the AXMEDIS Player. The license could be even for another AXMEDIS Object not for the one shown in the ActiveX (which can be a preview version)

5.5.2 Visual Control Interface

Property SupportVisualControl (read only)	
bool SupportVisualControl	
Description	indicates if for the currently opened content the visual control methods can be used
DefaultValue	false
Property Zoom (read write)	
double Zoom	
Description	is the zoom factor to be applied (1 = 100%)
DefaultValue	false
Remark	if the zoom factor is changed the AutoFit property is set to false
Property AutoFit (read write)	

bool AutoFit	
Description	when true indicates to resize the visual content to be fitted inside the window
DefaultValue	true
Remark	none
Method Fit	
void Fit(double width, double height)	
Description	fits the zoom factor to display the image at the desired size (keeping aspect ratio)
Input parameters	• *double width, height* - the image size in pixels
Return value	none
Remarks	the AutoFit property is set to false
Method FitToWindow	
void FitToWindow()	
Description	fits the visual content to the size of the image
Input parameters	none
Return value	none
Remarks	the AutoFit property is manteined
Method ZoomIn	
void ZoomIn(double perc)	
Description	Increments the zoom factor of the given percentage
Input parameters	• *double perc* – the percentage used to increment the zoom factor (e.g. 10 for 10%)
Return value	none
Remarks	the AutoFit property is set to false
Method ZoomOut	
void ZoomOut(double perc)	
Description	Decrements the zoom factor of the given percentage
Input parameters	• *double perc* – the percentage used to decrement the zoom factor (e.g. 10 for 10%)
Return value	none
Remarks	the AutoFit property is set to false
Property FullScreen (read write)	
bool FullScreen	
Description	when true switch to full screen mode
DefaultValue	false
Remark	none
Method Print	
void Print()	
Description	Prints the visual content
Input parameters	none
Return value	none
Remarks	none

5.5.3 Time Control Interface

Property SupportTimeControl (read only)	
bool SupportTimeControl	
Description	indicates if for the currently opened content the time control methods can be used
DefaultValue	false

Method Play	
void Play()	
Description	Starts playing the content or continue in case of pause
Input parameters	none
Return value	none
Remarks	has no effect if it is already playing
Method Pause	
void Pause()	
Description	Pause the execution or continue execution
Input parameters	none
Return value	none
Remarks	none
Method Stop	
void Stop()	
Description	Stops execution
Input parameters	none
Return value	none
Remarks	none
Method JumpToTime	
void JumpToTime(ulong time)	
Description	Set the execution time in milliseconds
Input parameters	• *ulong time* – the time at which jump the execution
Return value	none
Remarks	none
Property IsPlaying (read only)	
bool IsPlaying	
Description	indicates if the content is being played
DefaultValue	false
Property Duration (read only)	
ulong Duration	
Description	indicates duration in milliseconds
DefaultValue	0
Property CurrentTime (read only)	
ulong CurrentTime	
Description	indicates current execution time in milliseconds
DefaultValue	0
Property StartTime (read write)	
ulong StartTime	
Description	indicates the time in milliseconds where to start
DefaultValue	0
Property EndTime (read write)	
ulong EndTime	
Description	indicates the time in milliseconds where to end execution
DefaultValue	Duration

5.5.4 Generic Command Interface

This interface allows to execute commands giving a commad identifier. The command identifier are:

Command ID	Value
"Load"	string with the URL
"Src"	string
"OpenContent"	content position
"ContentCount"	
"BackgroundColor"	string
"ShowMetadata"	none
"ShowLicense"	none
"NeedLicense"	bool
"HaveLicense"	bool
"Zoom"	double
"ZoomIn"	double
"ZoomOut"	double
"AutoFit"	
"Fit"	string "w,h" (e.g. "100, 400")
"FullScreen"	bool
"Print"	none
"Play"	none
"Stop"	none
"Pause"	none
"IsPlaying"	bool
"CurrentTime"	ulong (read only)
"Duration"	ulong (read only)
"StartTime"	ulong
"EndTime"	ulong

Method execCommand	
HRESULT execCommand(string cmdID, bool showUI, variant value, bool ret)*	
Description	executes the command described in cmdID, with an optional argument contained in value
Input parameters	• *string cmdID* – command to be done • *bool showUI* – indicates if UI update is needed • *variant value* – the argument for the command • *bool* ret* – contains if the command has been executed or not
Return value	S_OK if ok
Remarks	none
Method queryCommandEnabled	
HRESULT queryCommandEnabled(string cmdID, bool ret)*	
Description	looks if the specified command can be successfully executed in the current context
Input parameters	• *string cmdID* – command to be checked • *bool* ret* – contains the result (true if the command is enabled)
Return value	S_OK if ok
Remarks	none
Method queryCommandIndeterm	
HRESULT queryCommandIndeterm(string cmdID, bool ret)*	
Description	looks if the specified command is in the indeterminate state
Input parameters	• *string cmdID* – command to be checked • *bool* ret* – contains the result (true if the command is in the indeterminate state)
Return value	S_OK if ok
Remarks	none
Method queryCommandStatus	

HRESULT queryCommandStatus(string cmdID, bool ret)*	
Description	indicates if the specified command has been executed on the object or not
Input parameters	• *string cmdID* – command to be checked • *bool* ret* – contains the result
Return value	S_OK if ok
Remarks	none
Method queryCommandSupported	
HRESULT queryCommandSupported(string cmdID, bool ret)*	
Description	looks if the specified command is supported
Input parameters	• *string cmdID* – command to be checked • *bool* ret* – contains the result (true if the command is supported)
Return value	S_OK if ok
Remarks	none
Method queryCommandValue	
HRESULT queryCommandValue(string cmdID, variant ret)*	
Description	returns the current value for a command
Input parameters	• *string cmdID* – command • *variant* value* – contains value
Return value	S_OK if ok
Remarks	none

5.6 Examples of usage

The following is an example of use of the activex to make a preview of an AXMEDIS object:

```
axactivex=new AXMEDISViewerCtrl();
axactivex.BackgroudColor="#ffffff";

//open the object downloading it from an URL
axactivex.Load(http://AXMEDIS.org/demo/metropolis.axm);

//show the metadata of the whole object
axactivex.ShowMetadata();

//look for a license and if not present acquire it
if(axactivex.NeedsLicense && !axactivex.HaveLicense)
    axactivex.AcquireLicense(licenseID);
else
    axactivex.ShowLicense();

// open all content elements inside the object and make a preview for 10s each
for(int i=1; i<=axactivex.ContentCount; i++)
{
    axactivex.OpenContent(i);

    //if currently opened content is image/video/document fit it to 100x100 preview
    if(axactivex.SupportVisualControl)
        axactivex.fit(100,100);

    //if currently opened content is audio/video preview it for 10s other wise wait for 10s
    if(axactivex.SupportTimeControl)
        axactivex.EndTime=10000; //10s preview
    else
        wait(10000); //wait 10s
}
```

5.7 Integration and compilation issues

None

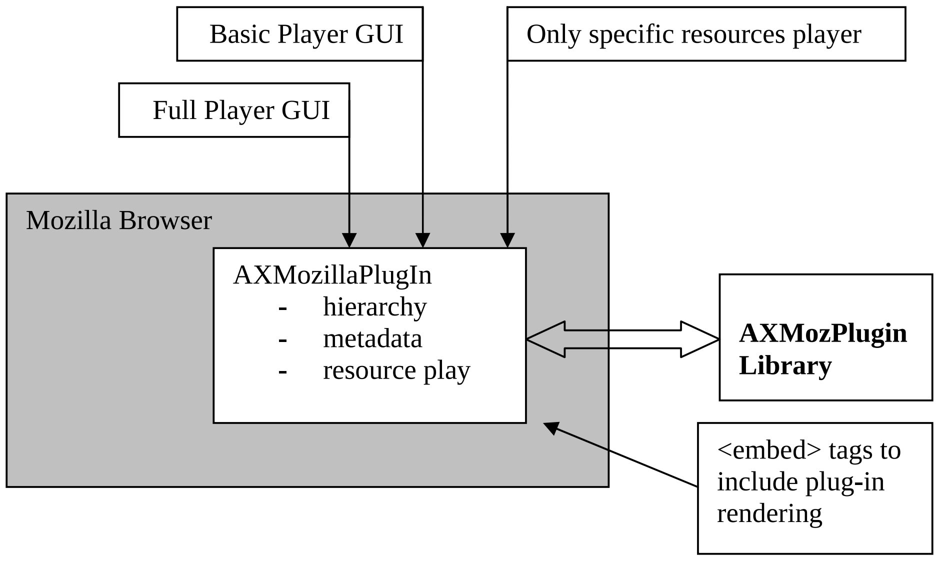
6 Module - AXMEDIS plug-in into Mozilla (SEJER)

Module/Tool Profile		
AXMEDIS plug-in into Mozilla		
Responsible Name	Vincent JACQUET	
Responsible Partner	SEJER	
Status (proposed/approved)		
Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)		
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Win32	
Reference to the AXFW location of the source code demonstrator	https://cvs.AXMEDIS.org/repos/Applications/axmozillaplugin	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs. AXMEDIS.org/repos/Applications/axmozillaplugin/doc/configuration-deployment	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any	N/A	
Test cases (present/absent)		
Test cases location	http://////////////////	
Usage of the AXMEDIS configuration manager (yes/no)	Yes	
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved	-- --	
Major pending requirements	Specification of the interface the plugin should expose to the javascript.This interface must be identical to the interface the ActiveX exposes.	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section

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

6.1 General Description of the Module

main purpose and description, major functionalities, etc.
The purpose of this module is to enable to rendering of AXMEDIS content inside Mozilla's web browser and let the user interact with it.
Generally speaking, a plug-in for mozilla is a external library exposing specific methods that is dynamically loaded by Mozilla. When embedded inside an HTML page, the web browser delegates the rendering and the event handling to this library.
Thus the AXMEDIS player is considered as being able to render the AXMEDIS content into its own window, which, in case of the Mozilla plug-in, .is owned by the web browser.



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;

6.1.1 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 run 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:

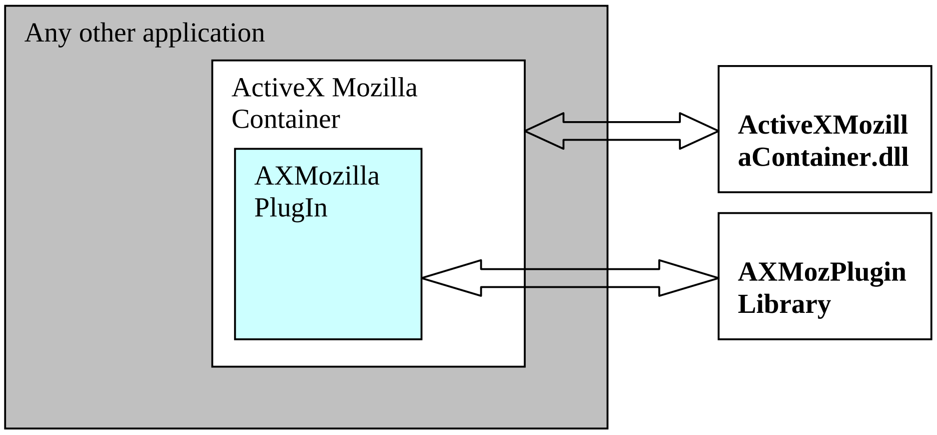
- allowing the hosting browser to call some methods the plug-in exposes to him, that is making the plug-in scriptable,
- allowing the plug-in to access the DOM of the hosting Window to manipulate it;

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 uniqueness, 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.
A possible identifier could be "@AXMEDIS.org/axmozilla,version=0.1"

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.

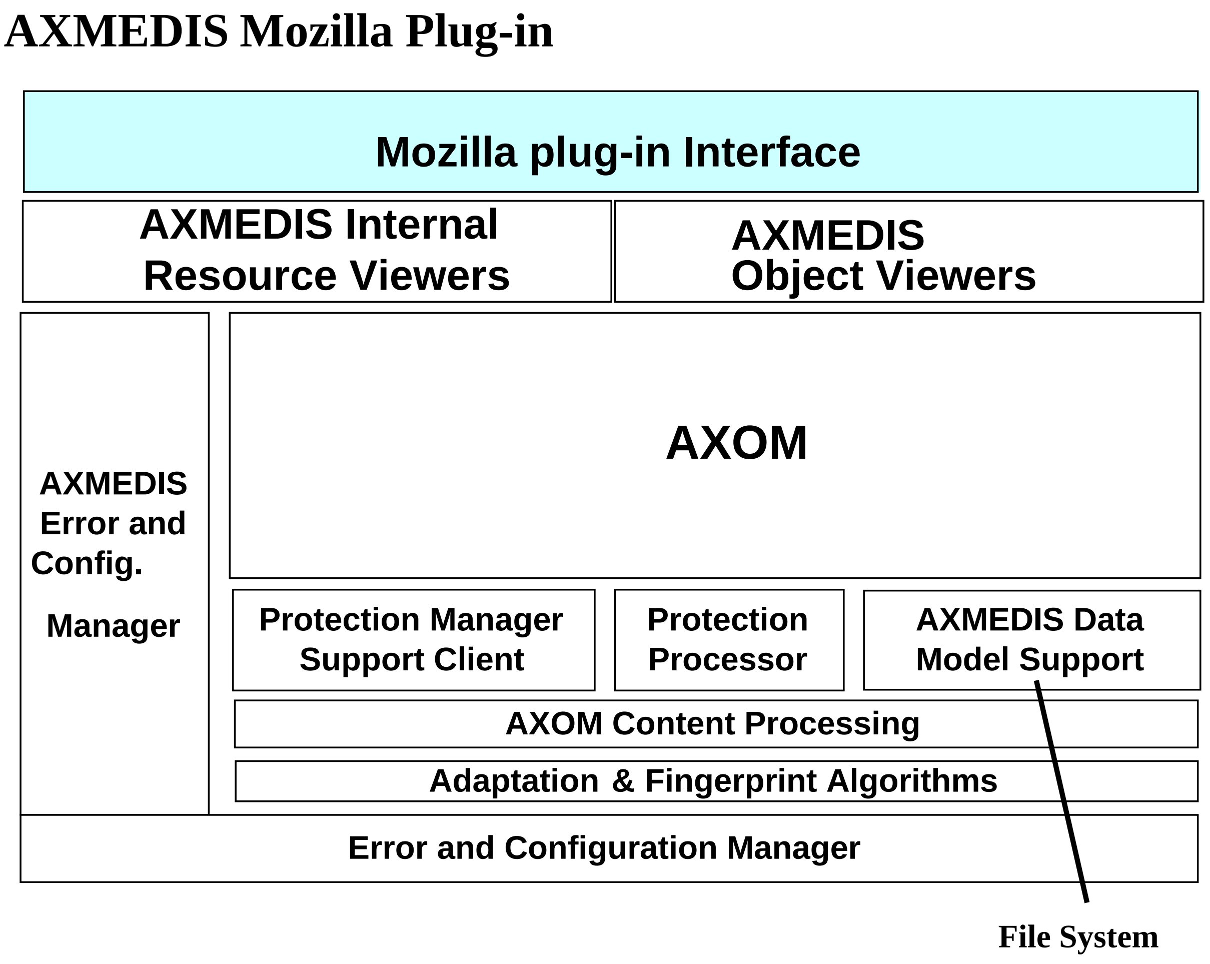


6.1.2 Authentication

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 license are at disposal to consume the requested content.

6.1.3 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

Mozilla documentation & API is available at http://www.mozilla.org/projects/plugins/

The Mozilla plug-in is constructed on top of the AXMEDIS framework and it exposes the interfaces required by Mozilla in order to embed the player functionnalities inside Mozilla.

The code of the plug-in is specific to the 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. As the AXMEDIS editor is cross-platform, having several build should not add a major overhead.

6.2 Module Design in terms of Classes

As most extensible architecture written in C/C++, the dynamic library export C methods to interface the application and the plugin: NP_Initialize to initialize the library, NP_Shutdown to free the resources and NP_GetEntryPoints to retrieve the address of functions needed for the management of the plugin.
The functions name are listed below:

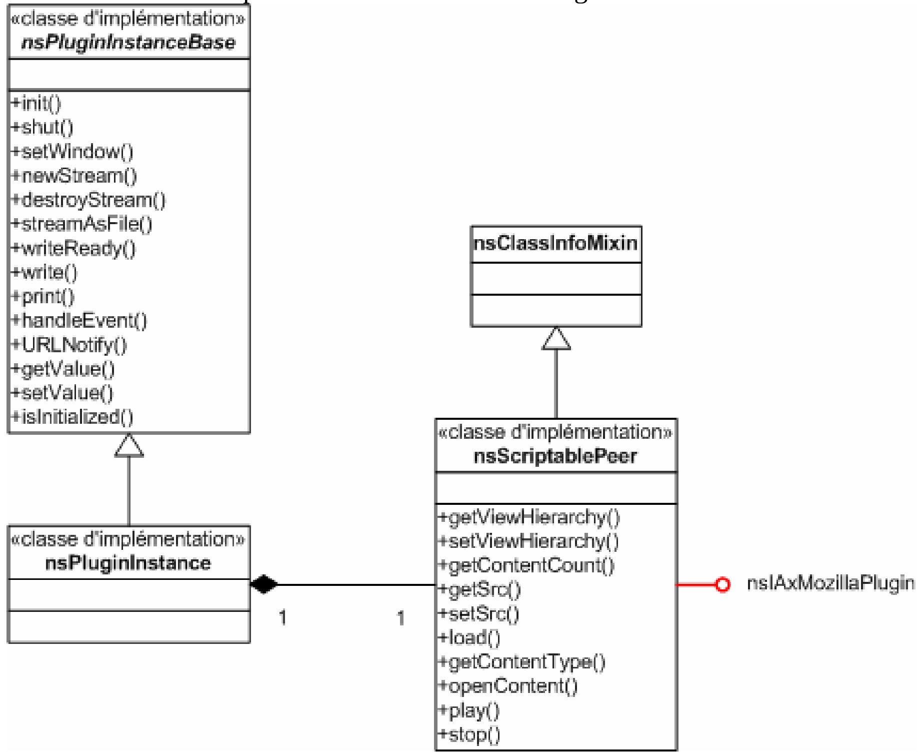
- `NPP_New;`
- `NPP_Destroy;`
- `NPP_SetWindow;`
- `NPP_NewStream;`
- `NPP_DestroyStream;`
- `NPP_StreamAsFile;`
- `NPP_WriteReady;`
- `NPP_Write;`
- `NPP_Print;`
- `NPP_HandleEvent;`

- `NPP_URLNotify;`
- `NPP_GetValue;`
- `NPP_SetValue;`

From an architectural point of view, these functions can be considered as methods and properties of an object, as each and every one of theme have “`NPP instance`” as first parameter.

The NPP_New method initialize a creation structure which is passed as parameter to the `NS_NewPluginInstance` method. This method is the factory of plug-in for the library, as the requested mime type is one of the parameter. Therefore the same library can be used to created different plugs-in. This method instantiates the plug-in and returns the abstract base class for it, so that the other functions will be able to delegate their calls to the plug-in object.

The Mozilla plug-in also expose the interface nsIAxMozillaPlugin to become scriptable through JavaScript in the browser. This interface is implemented in nsAxMozillaPlugin.



(note: nsScriptablePeer has been renamed as nsAxMozillaPlugin)

The initialization and cleanup of AXMEDIS component is done in the NP_ Initialize and NP_Shutdown methods.
The initialization of the plug-in window is done in the nsPluginInstanc::init method, which register the windows procedure for the player.
Then, all the methods of the scriptable peer, used for scriptability, invoke AXMEDIS internal methods.

The idl of the interface exposed to the browser is:

```
[scriptable, uuid(a59a052c-3ebe-48f2-8e96-a99896ba46cf)]
interface nsIAxMozillaPlugin : nsISupports {

    /* behaviour methods and attributes */
    attribute boolean viewHierarchy;
    attribute string backgroundColor;

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

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

    void play();

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

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

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

6.3 User interface description

The user interface of the player is the one of the basic AXMEDIS Player, as it is embedded in a rectangle in the display area of the web browser. Still, the scriptable capabilities allow the designer of the HTML page to add buttons and other commands that can invoke methods of the player itself.

6.4 Technical and Installation information

The installation can be automated using the XPI mechanism. It consist basically to define an zip archive containing the files to be installed in the Mozilla plug-in directory and a javascript –install.js -customizing the installation process. To be recognize by mozilla as n installation package, the extension must be .xpi.

The installation script declares, among other things:

- The component file describing the interface exposed by the plugin: nsIAxMozillaPlugin.xpt
- the plugin file: npaxmozilla.dll
- the required files: "xerces-c_2_6D.dll", "libcurl.dll", "msvcp71d.dll", "msvcr71d.dll"
- the extension of the file : axm
- the mime-type: application/AXMEDIS-object

- the version of the plugin.

The script copies the plugin's file, the required files and the component file in the Mozilla's plugins subdirectory.
It also creates the following keys in the registry

```
[HKEY_CURRENT_USER\Software\MozillaPlugins\@AXMEDIS.org/axmozilla,version=0.1]
"Path"="C:\\Program Files\\Mozilla Firefox\\plugins\\npaxmozilla.dll"
"XPTPath"="C:\\Program Files\\Mozilla Firefox\\plugins\\nsIAxMozillaPlugin.xpt"
"ProductName"="AXMEDIS Mozilla plugin"
"Vendor"="AXMEDIS"
"Description"="Plugin for playing AXMEDIS Objects"
"Version"="0.1.0.0"

[HKEY_CURRENT_USER\Software\MozillaPlugins\@AXMEDIS.org/axmozilla,version=0.1\MimeTypes]

[HKEY_CURRENT_USER\Software\MozillaPlugins\@AXMEDIS.org/axmozilla,version=0.1\MimeTypes\
application/AXMEDIS-object]
"Description"="AXMEDIS Objects"
"Suffixes"="axm"

[HKEY_CURRENT_USER\Software\MozillaPlugins\@AXMEDIS.org/axmozilla,version=0.1\Suffixes]
"ax"="axm"
"axm"=""
```

The script do not create the following keys, as they are part of the editor/activeX.

```
[HKEY_CLASSES_ROOT\.axm]
@="AXMEDIS-object"
"Content Type"="application/AXMEDIS-object"

[HKEY_CLASSES_ROOT\MIME\Database\Content Type\application/AXMEDIS-object]
"Extension"=".axm"
```

These registry entries allow firefox top open .axm files direclty, without to download the file to the hard-drive or without embedding it in an HTML page.
Basically, the axm files will behave like pdf and swf files.

References to other major components needed	The same as the AXMEDIS player and ActiveX control.
Problems not solved	•
Configuration and execution context	This plug-in is executed inside of the Mozilla web browser. An XPI installation program can be defined to ease the installation process of the plug-in it-slef. The installation of all the part of AXMEDIS players is still to be defined.

6.5 Draft User Manual

The following sample of code shows how to declare the player in a cross-browser way. Mozilla will use the plug-in and Internet will use the ActiveX.

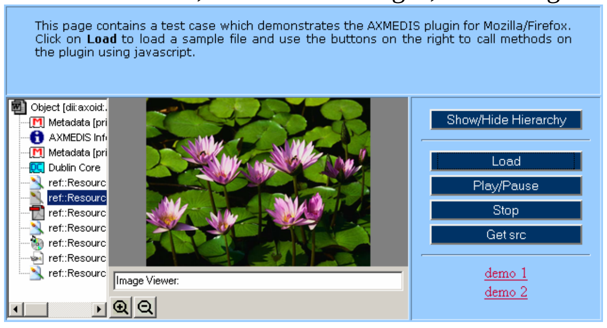
```
<object name="player" height="100%" width="100%" classid=clsid:B30B789D-2378-45BD-A1B5-AFC0B43919D3 VIEWASTEXT>
	<embed id="player"
		type="application/axmedis-object"
		width="100%"
		height="100%"
		viewHierarchy="yes">
	</embed>
</object>
```

The code below shows how to retrieve the plug-in defined in the previous example and how to invoke some of its methods.

```
<script>
var embed;
if (document.embeds != null && document.embeds.length > 0) {
    //mozilla
    embed = document.getElementById('player');
} else {
    // IE
    embed = document.all['player'];;
}
function toggleHierarchy() {
    embed.viewHierarchy = !embed.viewHierarchy;
}

function Play() {
    embed.play();
}

function Stop() {
    embed.stop();
}
```

6.6 Examples of usage

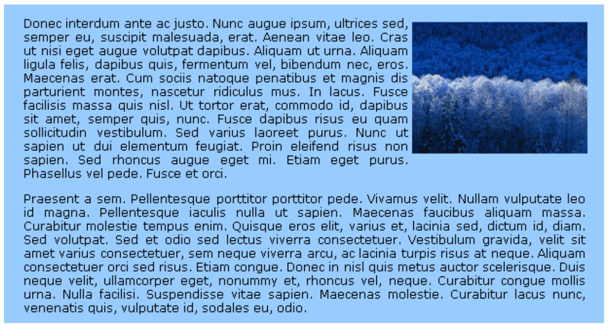
The two examples presented below demonstrate that the plug-in can be used as a component, with the tree view and tool bar, or as a “still image”, decorating the text.



This page contains a test case which demonstrates the AXMEDIS plugin for Mozilla/Firefox. Click on **Load** to load a sample file and use the buttons on the right to call methods on the plugin using javascript.

Object [dii:axoid:.
Metadata [pri
AXMEDIS Inf
Metadata [pri
Dublin Core
ref::Resourc
ref::Resourc
ref::Resourc
ref::Resourc
ref::Resourc
ref::Resourc
ref::Resourc

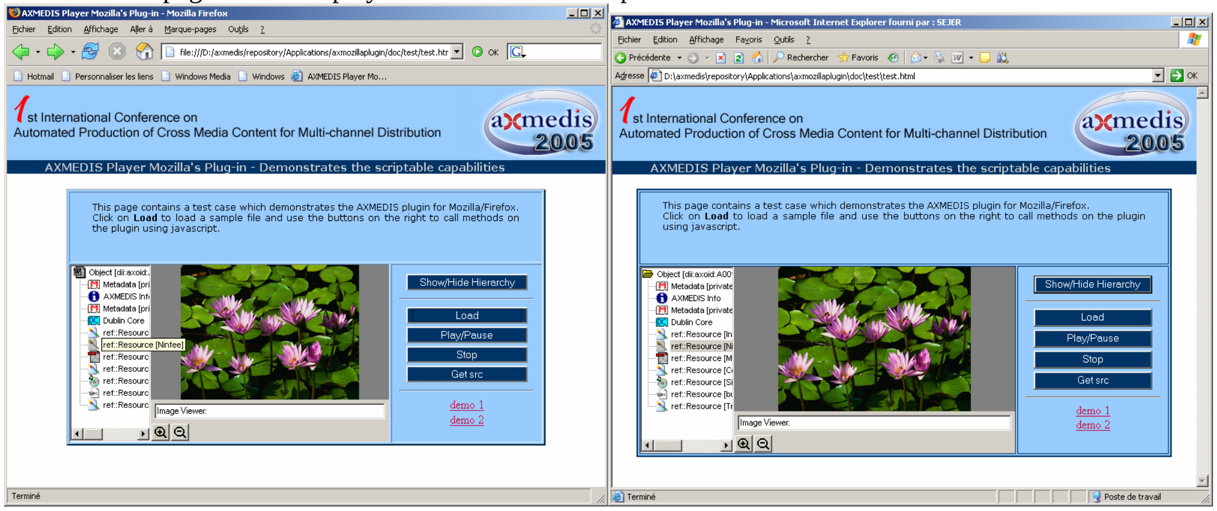
Image Viewer:



Donec interdum ante ac justo. Nunc augue ipsum, ultrices sed, semper eu, suscipit malesuada, erat. Aenean vitae leo. Cras ut nisi eget augue volutpat dapibus. Aliquam ut urna. Aliquam ligula felis, dapibus quis, fermentum vel, bibendum nec, eros. Maecenas erat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In lacus. Fusce facilisis massa quis nisl. Ut tortor erat, commodo id, dapibus sit amet, semper quis, nunc. Fusce dapibus risus eu quam sollicitudin vestibulum. Sed varius laoreet purus. Nunc ut sapien ut dui elementum feugiat. Proin eleifend risus non sapien. Sed rhoncus augue eget mi. Etiam eget purus. Phasellus vel pede. Fusce et orci.

Praesent a sem. Pellentesque porttitor porttitor pede. Vivamus velit. Nullam vulputate leo id magna. Pellentesque iaculis nulla ut sapien. Maecenas faucibus aliquam massa. Curabitur molestie tempus enim. Quisque eros elit, varius et, lacinia sed, dictum id, diam. Sed volutpat. Sed et odio sed lectus viverra consectetuer. Vestibulum gravida, velit sit amet varius consectetuer, sem neque viverra arcu, ac lacinia turpis risus at neque. Aliquam consectetuer orci sed risus. Etiam congue. Donec in nisl quis metus auctor scelerisque. Duis neque velit, ullamcorper eget, nonummy et, rhoncus vel, neque. Curabitur congue mollis urna. Nulla facilisi. Suspendisse vitae sapien. Maecenas molestie. Curabitur lacus nunc, venenatis quis, vulputate id, sodales eu, odio.

And the same page can be displayed in both Internet Explorer and Mozilla:


AXMEDIS Player Mozilla's Plug-in - Mozilla Firefox
AXMEDIS Player Mozilla's Plug-in - Microsoft Internet Explorer fourni par : SEJER
1st International Conference on
Automated Production of Cross Media Content for Multi-channel Distribution
axmedis 2005
AXMEDIS Player Mozilla's Plug-in - Demonstrates the scriptable capabilities
Show/Hide Hierarchy
Load
Play/Pause
Stop
Get src
demo 1
demo 2
Image Viewer:
Terminé

This page contains a test case which demonstrates the AXMEDIS plugin for Mozilla/Firefox. Click on **Load** to load a sample file and use the buttons on the right to call methods on the plugin using javascript.

6.7 Integration and compilation issues

The plug-in is at the end of the development chain. Therefore, it is very sensitive to changes in other projects.

7 Module - AXMEDIS plugin into Multimedia Players

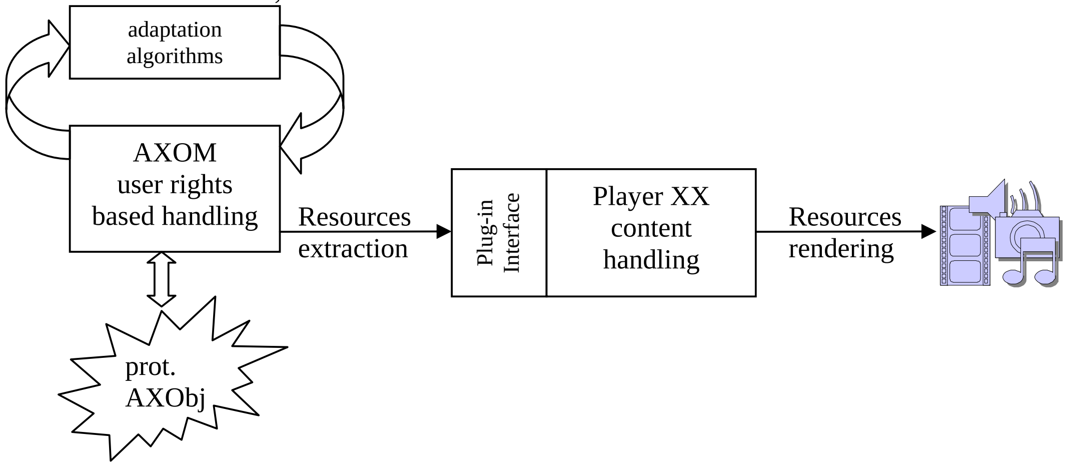
Module/Tool Profile	
AXMEDIS plugin into Multimedia Players	
Responsible Name	Rogai
Responsible Partner	DSI
Status (proposed/approved)	
Implemented/not implemented	Not implemented
Status of the implementation	
Executable or Library/module (Support)	
Single Thread or Multithread	
Language of Development	
Platforms supported	
Reference to the AXFW location of the source code demonstrator	https://cvs.AXMEDIS.org/repos/....................
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.
Reference to the AXFW location of the demonstrator executable tool for public download	
Address for accessing to WebServices if any, add	

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

7.1 General Description of the Module

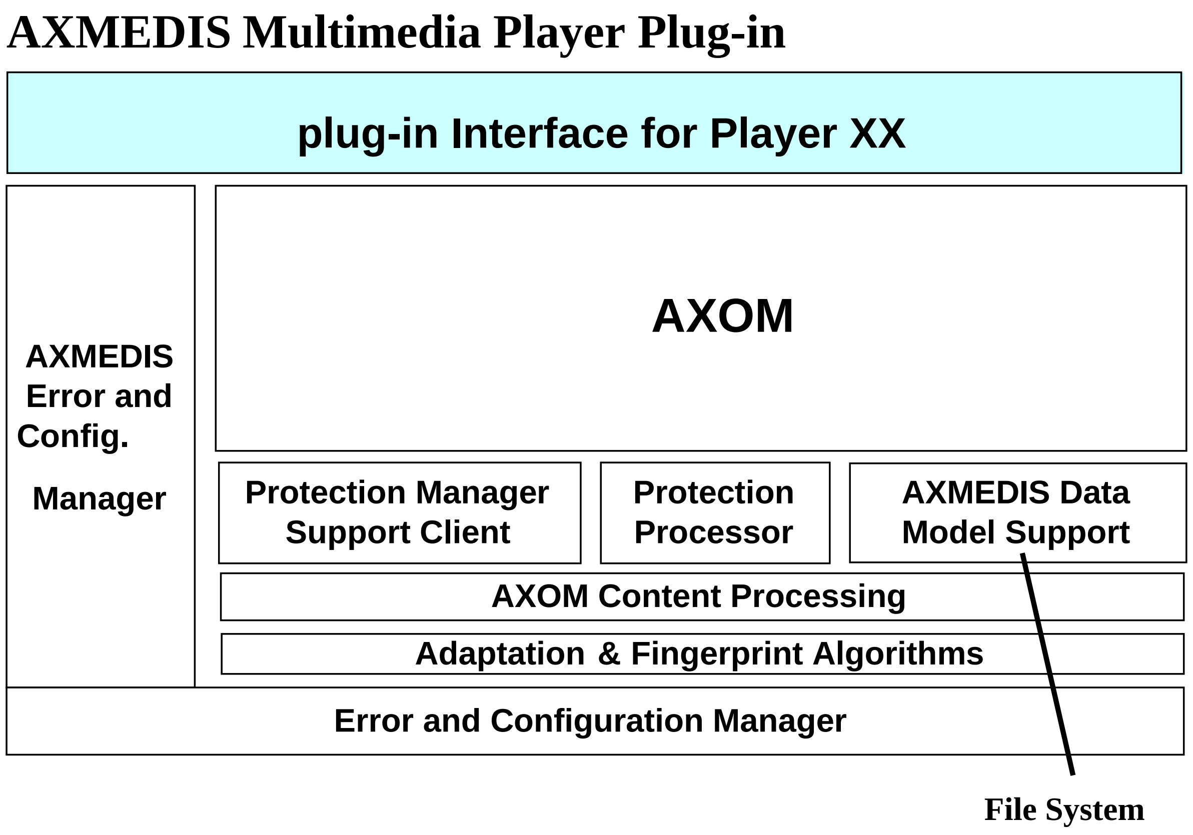
The main difference among AXMEDIS ActiveX/Mozilla Plug-in and AXMEDIS Plug-in for Multimedia players is the rendering capabilities. The first manages all the client area assigned to it, full of controls, scrollbars, options, tabs… etc. In the Plug-in for Multimedia the most important part is rendering 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. consumers' world; the DRM has to be considered with much attention.



Since the plug-in of AXMEDIS in other player can really increase the AXMEDIS utilization in the

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

7.1.1 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)

7.1.2 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.

8 Executable Tool – AXMEDIS PC Player (DSI)

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

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

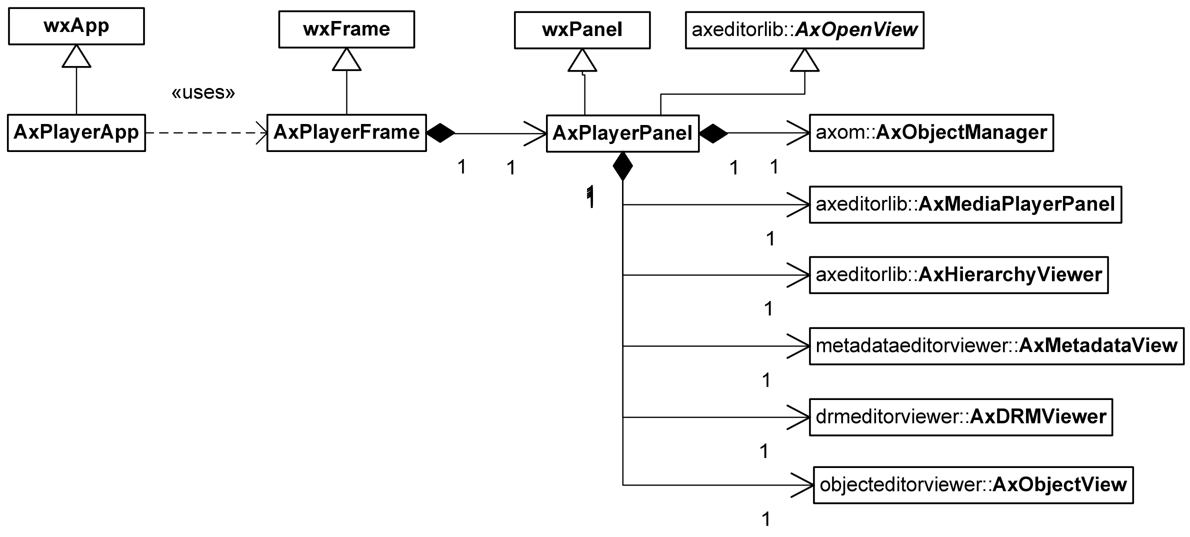
8.1 General Description of the Module

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 nevertheless allow, for his limited capabilities (in comparison to the Editor), a simpler porting to other platforms like e.g. Linux OS. At the same time, 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. 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.

8.2 Module Design in terms of Classes


wxApp
wxFrame
wxPanel
axeditorlib::AxOpenView
«uses»
AxPlayerApp
AxPlayerFrame
AxPlayerPanel
axom::AxObjectManager
axeditorlib::AxMediaPlayerPanel
axeditorlib::AxHierarchyViewer
metadataeditorviewer::AxMetadataView
drmeditorviewer::AxDRMViewer
objecteditorviewer::AxObjectView
1

8.3 User interface description

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 behavior of the Player according to the user preferences. 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 Hierarchy Viewer to select which content to reproduce.

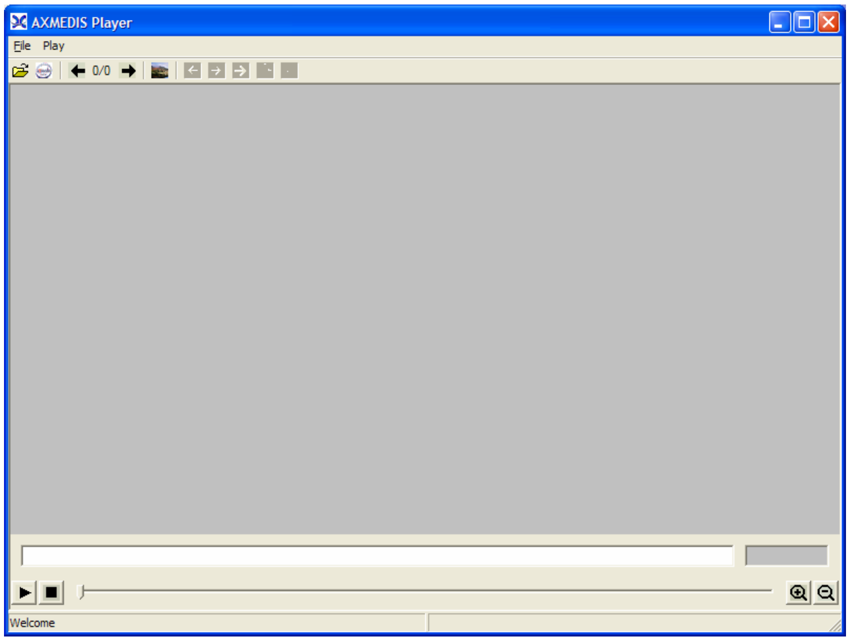
See the Draft User Manual for some examples.

8.4 Technical and Installation information

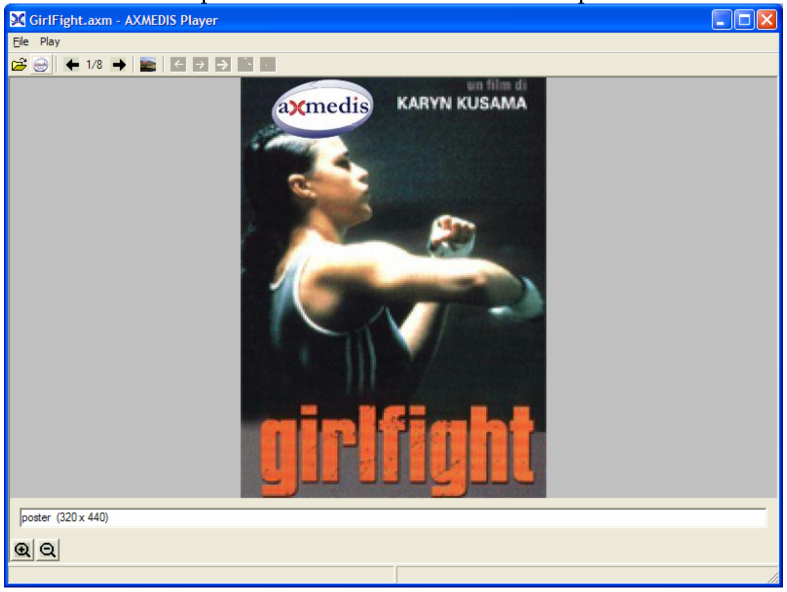
References to other major components needed	AXMEDIS Object Manager AXMEDIS Internal Players AXMEDIS Hierarchy Editor & Viewer AXMEDIS Metadata Editor & Viewer AXMEDIS DRM Editor & Viewer AXMEDIS Object Editor & Viewer
Problems not solved	•
Configuration and execution context	

8.5 Draft User Manual

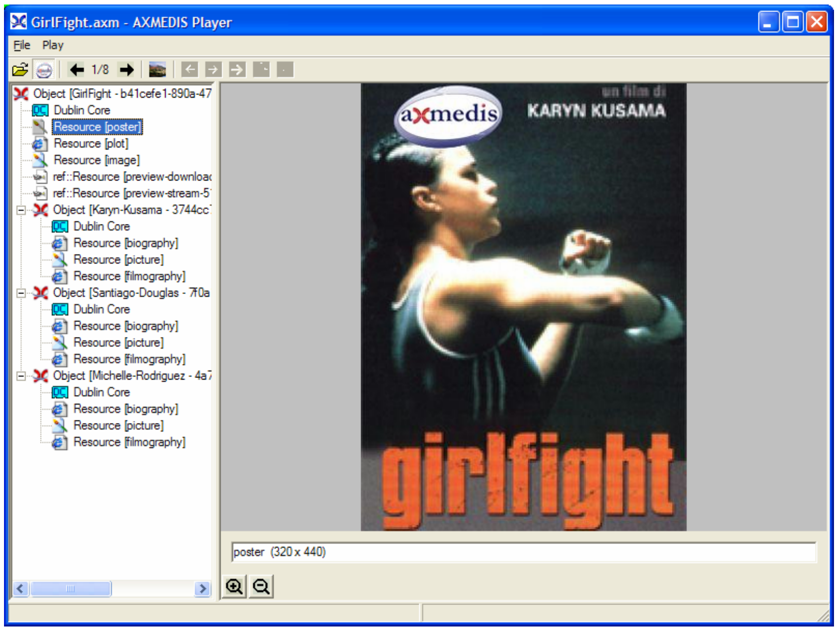
When the player is started an AXMEDIS object may be loaded using the **File/Open…** menu or using the button on the toolbar

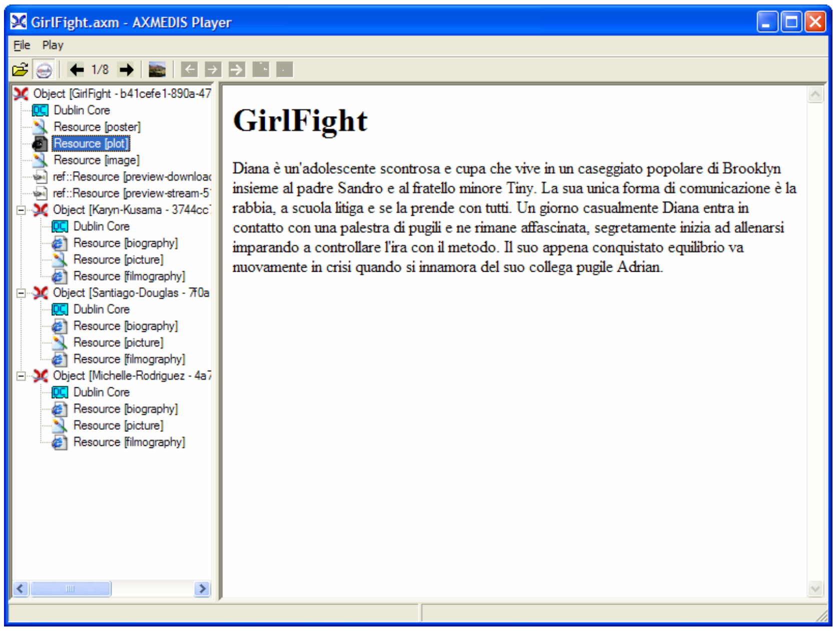


When the file is opened the first resource inside is opened:



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

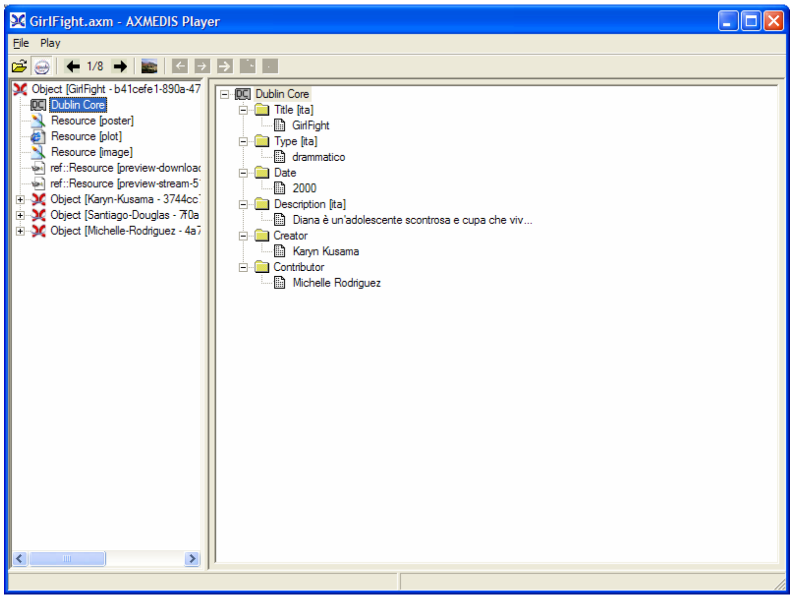




GirlFight

Diana è un'adolescente scontrosa e cupa che vive in un caseggiato popolare di Brooklyn insieme al padre Sandro e al fratello minore Tiny. La sua unica forma di comunicazione è la rabbia, a scuola litiga e se la prende con tutti. Un giorno casualmente Diana entra in contatto con una palestra di pugili e ne rimane affascinata, segretamente inizia ad allenarsi imparando a controllare l'ira con il metodo. Il suo appena conquistato equilibrio va nuovamente in crisi quando si innamora del suo collega pugile Adrian.

The metadata of the object may be view double clicking on the Dublin Core element in the hierarchy view:



- Object [GirlFight - b41cefe1-890a-47
  - Dublin Core
  - Resource [poster]
  - Resource [plot]
  - Resource [image]
  - ref::Resource [preview-downloac
  - ref::Resource [preview-stream-5
- Object [Karyn-Kusama - 3744cc
- Object [Santiago-Douglas - 7f0a
- Object [Michelle-Rodriguez - 4a7

- Dublin Core
  - Title [ita]
    - GirlFight
  - Type [ita]
    - drammatico
  - Date
    - 2000
  - Description [ita]
    - Diana è un'adolescente scontrosa e cupa che viv...
  - Creator
    - Karyn Kusama
  - Contributor
    - Michelle Rodriguez

8.6 Examples of usage

See Draft User Manual

8.7 Integration and compilation issues

None

8.8 Configuration Parameters

Config parameter	Possible values

8.9 Errors reported and that may occur

Error code	Description and rationales

9 Executable Tool – AXMEDIS PDA Player (TISCALI)

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

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

9.1 General Description of the Module

The AXMEDIS Player may be partially ported on at least one of the most widespread PDA device with well supported OS and platform. The functions of Playing SMIL and MP4 will be supported.

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. The players for SMIL and MP4 on WindowsCE are proposed.

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.

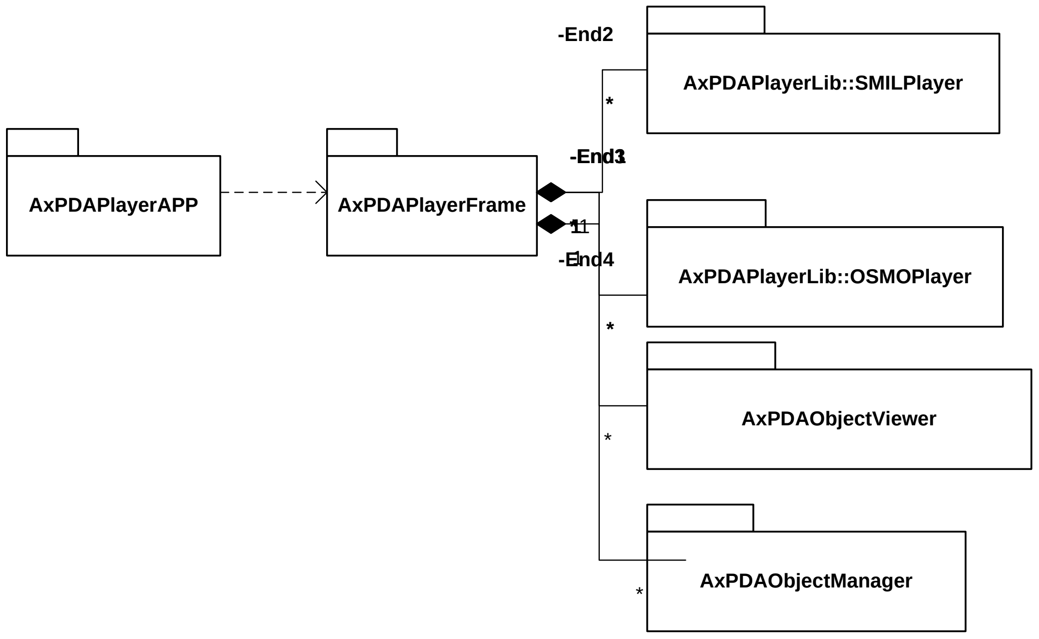
For development, Windows CE edition of the Visual Studio package is needed. The package allows compiling an application for Windows CE/PocketPC and testing it on PC in emulated environment (it is not an emulation of the device, it is instead another compiler for PC with APIs redirected).Portability of 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 internal SMIL player will support to play SMIL objects inside AXMEDIS object. A SMIL player has to be able to render different kinds of media objects (text, audio, images, video...). Currently the SMIL 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. The internal SMIL player will be controlled 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. We use AMBULANT SMIL player for the internal SMIL player. 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.

The SMIL Player should directly extract the SMIL resource from AXMEDIS object without need to saving it as a temporary file on the disk.

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

9.2 Module Design in terms of Classes



Opens an AXMEDIS file
```
function openFile()
```

Gets the player
```
function getInternalPlayer()
```
For each player it at least contains the following functions:

Calls the play, pause and stop commands on the player
```
function play()
function pause()
function stop()
```

9.2.1 Porting of the AXOM on PDA (DSI)

In order to achieve a working version of the AXOM on PDA platforms a reference architecture has to be chosen. Pocket PC 2003 with Windows CE version 4.20 (WCE420) has been chosen as main testing platform. It is not the last version of Microsoft mobile operative system but it is quite new therefore it is widespread and provides a good set of basic functionalities. Moreover, it is compatible with new version of Microsoft OS for mobiles.
For the first tests, Microsoft eMbedded Visual C++ 4.0 (eVC4) has been chosen as reference development environment since it provides a quite new C++ compiler and it can be enriched with a wide set of Standard Development Kit for different OSs and hardware platforms.

The porting of AXOM libraries over PDA platforms requires the porting and usage of the following well-known libraries:

- STLPort version 5.0.1 or greater. This is an open source implementation of the C++ Standard Template Library (STL). It has already been ported for eVC4 and WCE platforms. In that way, AXOM code, which relies on STL classes, should not be changed during the porting.
- Xerces-C++ version 2.7.0. Some effort has to be done to port this library on PDA platform since Xerces community does not provide any hint for doing that port. Only general guidelines are available.
- OpenSSL version 0.9.8a or greater. It already provides means to be ported on PDAs. The main issue in doing that is the introduction of a support library (WCECompat version 1.2) which provides to

OpenSSL some feature of the C Runtime Library which are missed in the eVC4 libraries. In order to avoid complicated integration among STLPort, WCECompat and standard eVC4 CRT it is suitable to use OpenSSL as dynamic library on PDA.

-

The porting of the above libraries is sufficient in order to obtain a working version of the AXOM library with basic functionalities, i.e. loading and managing of AXMEDIS and MPEG-21 objects. AXDB related features are excluded from this "mobile" version since they are not useful in this context. In order to enable net protocols such as HTTP, FTP etc, also libcurl has to be ported. However, this is not essential in order to have a first working release.

All AXOM related libraries, such as PMS Client etc, have to be re-compiled using the development environment and the libraries described above.

9.2.2 Opening a SMIL component inside an AXMEDIS object

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

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

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

Note, since we will directly retrieve the resource through "AxIndex" instead of using original URL, we will modify the code here. i.e., to generate the data source or data stream from "AxIndex".

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

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

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

9.2.3 SMIL player for PDA (same as AMBULANT)

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

- A timer and event processor are created. The timer is the master clock for the presentation, and the event processor is a runqueue object that is used for low-level scheduling. Whenever the high-level scheduler wants some code to be executed it will add an event to the event processor, possibly with a timeout and a priority. The event processor runs in a separate thread, waits for events in it runqueue to become eligible and then runs them. This mechanism is the underlying engine that makes the whole player work, anything that needs to wait doesn't do so inline but uses the event processor to get a callback at a later stage: the scheduler, renderers needing input data, etc.
- A layout manager is created, which will be used to find where media items should be displayed. The SMIL layout manager reads the <layout> section of the DOM tree and builds a parallel layout tree (which also contains information on some of the body media nodes, the ones that have layout information themselves) of region node objects. Then this tree of region node objects is converted into a tree of surface template objects. To create top-level windows the new topsurface method of the window factory is used, and subregions are created using the new subsurface method of their parent surface template. The layout manager also contains mappings to be able to get from a node to the corresponding region node to the surface template, and this will be used during playback to play media items in the correct location.
- A timegraph is created. This is the internal representation of the <body> part of the DOM tree that will be used to play back the document. In addition a scheduler is created, which will interpret the data in the timegraph.

When the user selects Play we call the start method of the player object. This will invoke start on the scheduler. This will start playing the root node of the tree. The scheduler will now do all the SMIL 2 magic, whereby events such as the root node being played causes other nodes to become playable, etc. At some point a media item needs to be rendered. The scheduler calls the new playable method from the global playable factory. This will pass the DOM node to the various factories until one signals that it can create a playable for the object. In addition, if the playable has a renderer (which is true for most media objects, but not for things like SMIL animations) we also obtain the surface on which the media item should be rendered, through the layout manager. We then tell the renderer which surface to use. Soon afterwards the start method of the playable is called to start playback.

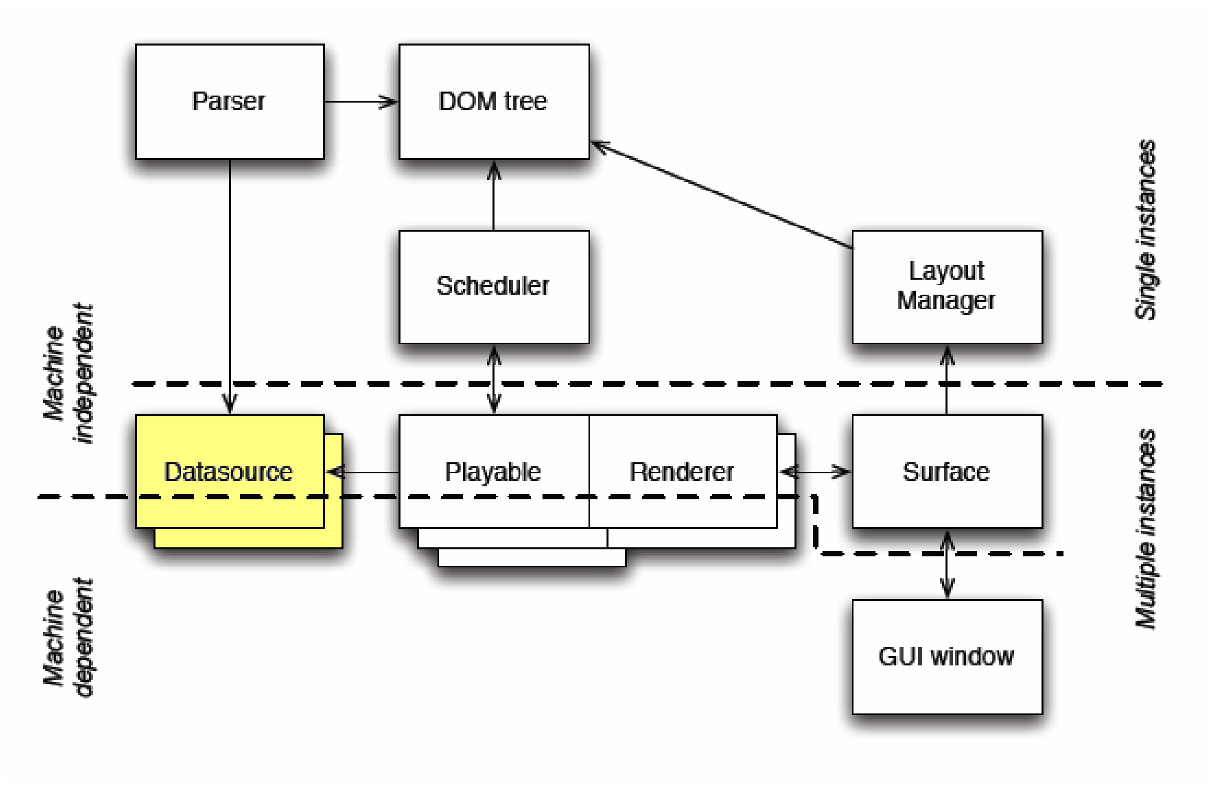
An average renderer will need to obtain data from AxIndex (note, originally, AMBULANT obtains from URL). It will do this by creating a datasource for the document through the datasource factory object.

Every time the renderer wants more data it calls the start method of the datasource passing a callback routine. Whenever data is available the datasource will schedule a call to the callback routine, through the event processor. When the renderer has enough information to start drawing it will not actually draw immediately, but it will send a need redraw call to its surface. This will percolate up the surface hierarchy, to the GUI code, and eventually come back down as a redraw call all the way to the renderer (assuming it is not obscured by other media items, etc). At this point the bits finally get drawn on the screen. Whenever anything “interesting” happens in the renderer (the media item stopped playing, the user clicked the mouse, etc) it invokes a corresponding method on its playable notification. This interface is implemented by the scheduler, and these notifications are how the scheduler gets informed that it can start scheduling new things, etc.

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

9.2.4 SMIL Player Ambulant and integration with AXOM (TISCALI)

Starting from the consideration done in the previous paragraph and analyzing the Ambulant player architecture (overall block diagram is shown in the figure below), components and interfaces, it's easy to identify in the Datasource interface one of the main points where performs changes to avoid integration with the AXMEDIS objects.

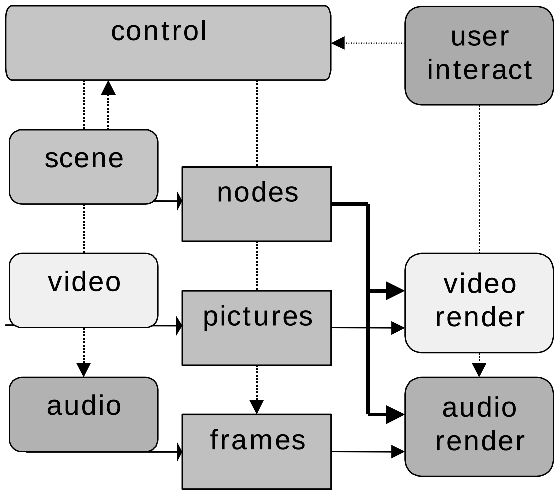


The Datasource interface is designed to implement URL retrieval schemes or file I/O and get external data to media handlers and other modules requiring data access.
The general interface is that a datasource is acquired through a datasource factory interface, which passes the URL to the various implementations until one is found that can handle it and returns a datasource object. The client then calls the start method on this object, passing a callback routine, and the datasource will arrange for the callback to be called as soon as data is available. No new callbacks will be done until a new call to start() is made, and the datasource has a buffer that can be limited, so this design allows for flow control over the net, if required.
There are specialised datasource interfaces for audio and video, that can handle extra things like converting audio from mp3 format to linear samples, or demultiplexing an audio/video stream.

The integration will be done extending the Datasource interface to manage AXMEDIS objects through the AXOM.
Main changes will regards the URL retrieval schemes and the file I/O that will be replaced with the rights AXOM function calls to retrieve the requested AXMEDIS object resource.

9.2.5 OSMO MPEG4 player on PDA (TISCALI)

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:

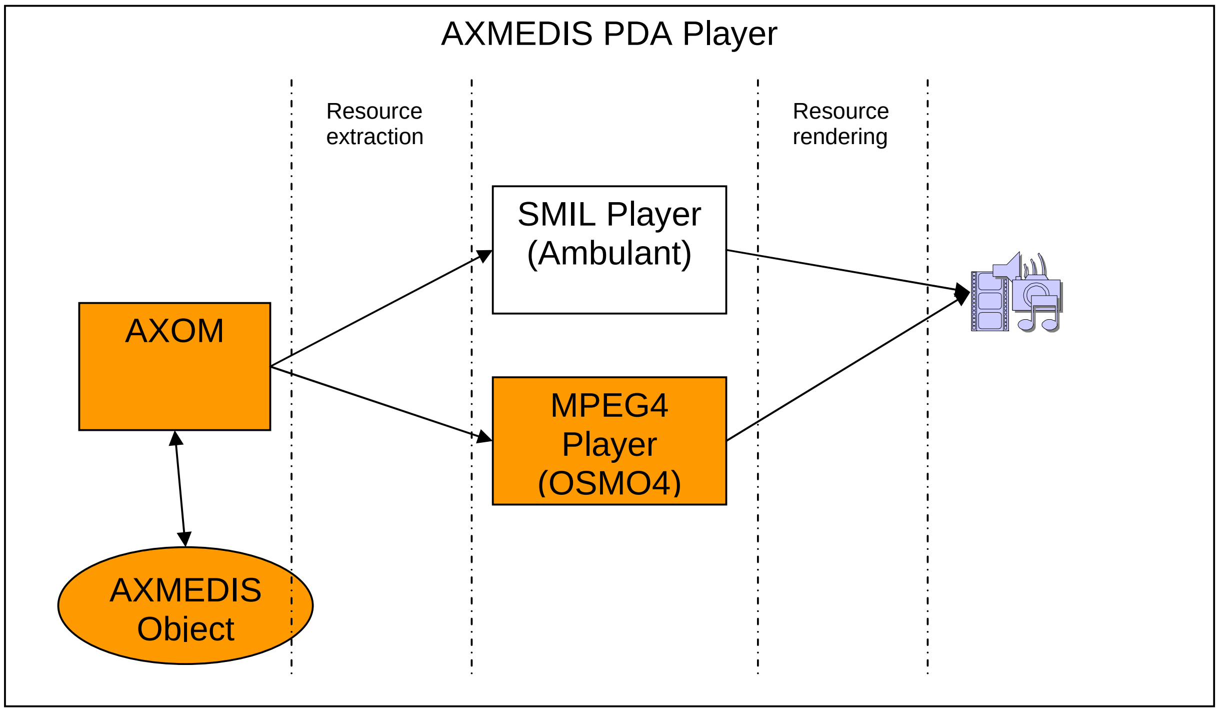
- Network Channel (DMIF): in this modality the only possible command is **open** of a network address After this is done by validation of the rights through the AXOM, all the streaming content is received and rendered including audiovisual objects and scene/interaction. Connection is closed when a new 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).

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

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.

9.2.6 OSMO and integration with AXOM AXMEDIS (TISCALI)

OSMO4 player integration will follow the same strategy adopted for the Ambulant player integration. So main changes need to be done in the input resource handler of OSMO4 player that needs to be adapted to interact with AXOM module.

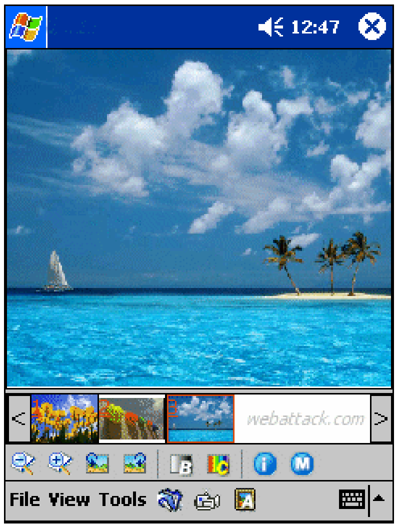


AXOM will be the access point to AXMEDIS objects, and it will provide to the axOSMO4 module, through the axCommand.Execute() function, an MPEG-4 bit stream to be played.

As for the Ambulant player integration, even the OSMO4 event manager need some changes to manage in the right way events like stop, pause, etc that need to be passed and managed in the AXOM too.

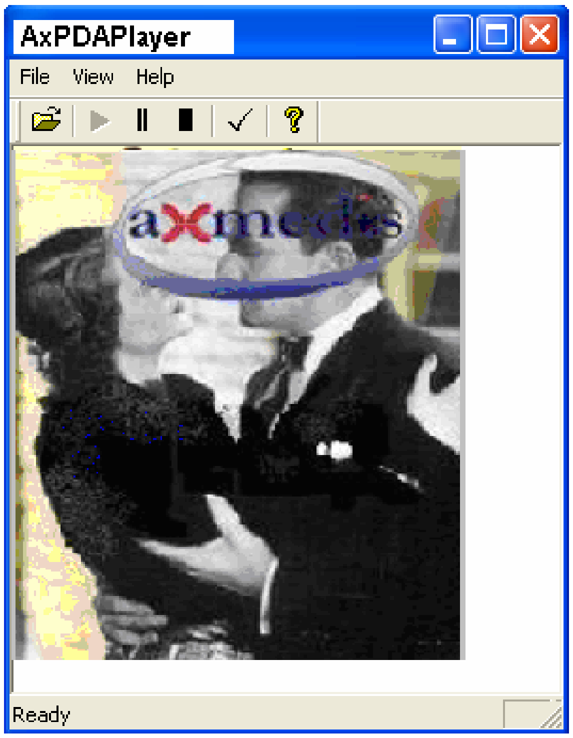
9.2.7 AXMEDIS PDA player user interface

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.

9.3 User interface description



Missing description of GUI functionalities

9.4 Technical and Installation information

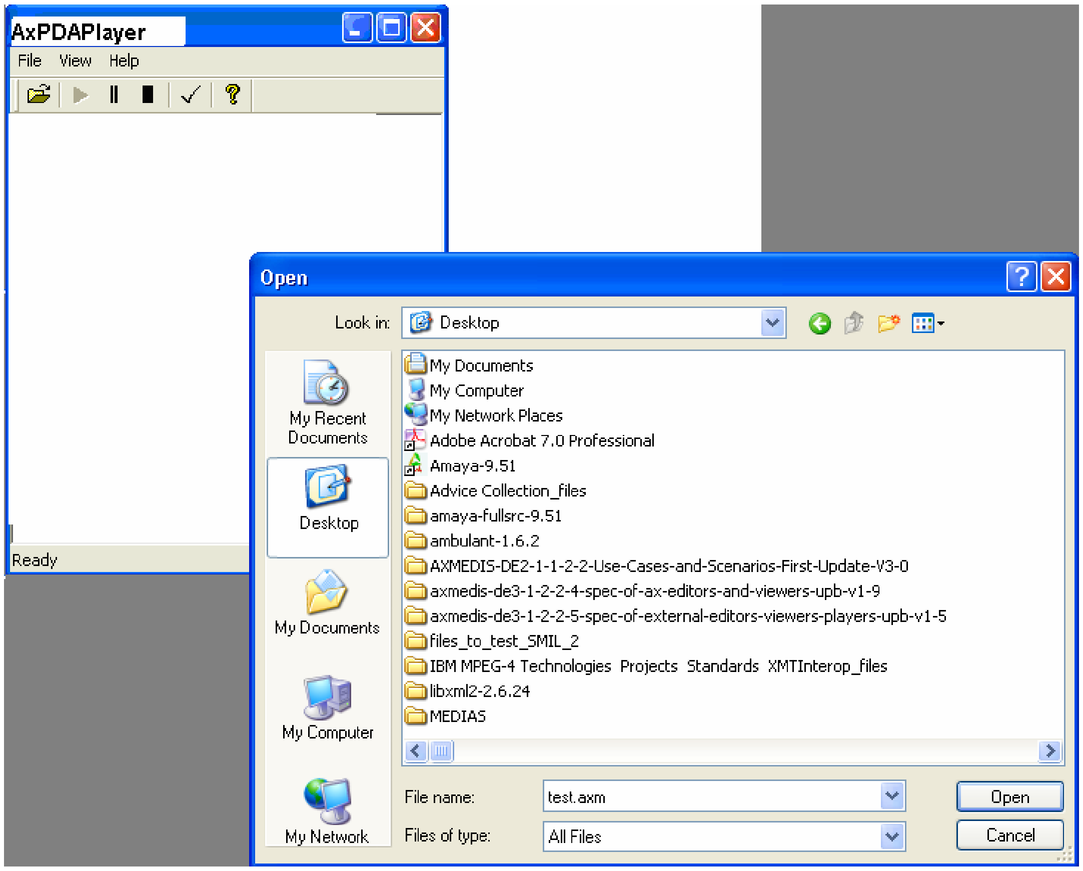
References to other major components needed	
Problems not solved	• Compilation of Ambulant and OSMO player on Windows CE
Configuration and execution context	

9.5 Draft User Manual

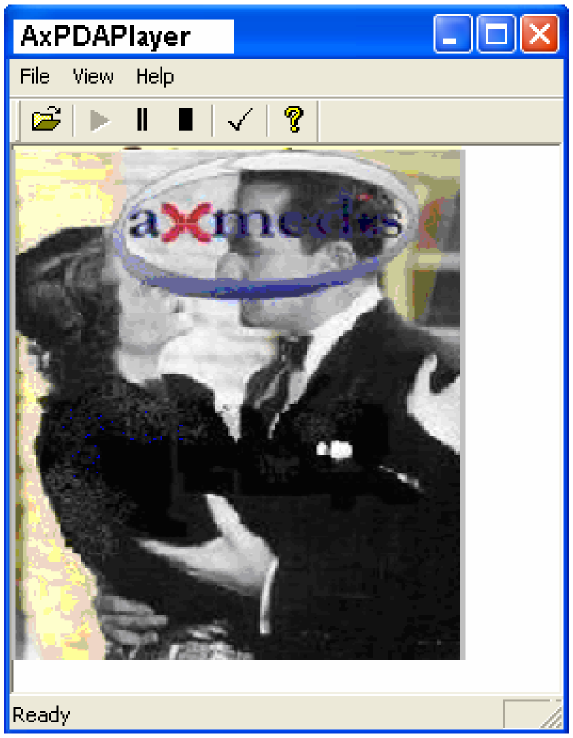
The AXMEDIS PDA player looks like a standard application, with a menu. The commands available so far are as follows:

File > Open	Opens a new AXMEDIS object
File > Play	Play an AXMEDIS object
File > Pause	Pause to play an AXMEDIS object
File > Stop	Stop an AXMEDIS object during the play
File > Exit	Closes the application
View > Toolbar	check to show the tool bar, unchecked to hide the tool bar
Help > About	Displays the about box

When the player is started an AXMEDIS object may be loaded using the **File/Open…** menu or using the button on the toolbar



When the player is started an AXMEDIS object can be played/paused/stopped using the **File/Play File/Pause, File/Stop** or using the button on the toolbar



9.6 Examples of usage

See Draft User Manual

9.7 Integration and compilation issues

- Description of the interoperability specification aspects related to the adoption of the software module in different operating systems and to be integrated in different contexts,
  - conditional compilations,
  - different behaviors in different context,
  - profiling,
  - configuration aspects,
  - etc.

10 Executable Tool – AXMEDIS Mobile Player (DSI, ILABS)

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

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

The specification of this part was in charge to EPFL until the July 2005. Not it has bee reallocated to other partners, see new DOW M25 M42.
A light version of the AXMEDIS player for Mobile will be realized in pure Java.
A simple mock up for symbim with capabilities of media streaming has been realized by DSI, demonstrating that the cost of developing a C++ AXOM based application could be too difficult and non portable on the many different mobile devices.

11 Executable Tool – AXMEDIS Tablet PC Player for School Bag on Mozilla (SEJER)

Module/Tool Profile		
AXMEDIS Tablet PC Player for School Bag on Mozilla		
Responsible Name	Vincent JACQUET	
Responsible Partner	SEJER	
Status (proposed/approved)		
Implemented/not implemented		
Status of the implementation		
Executable or Library/module (Support)		
Single Thread or Multithread		
Language of Development	C++, XUL, JavaScript	
Platforms supported		
Reference to the AXFW location of the source code demonstrator	https://cvs.AXMEDIS.org/repos/ Applications/axmozillaplayer	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs. AXMEDIS.org/repos/ Applications/axmozillaplayer	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location	http:///////////////////	
Usage of the AXMEDIS configuration manager (yes/no)		
Usage of the AXMEDIS Error Manager (yes/no)		
Major Problems not solved	-- --	
Major pending requirements	Specification of the interface the plugin should expose to the javascript.This interface must be identical to the interface the ActiveX exposes.	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a

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

11.1 General Description of the Module

The AXMEDIS player based on Mozilla technology can run on Tablet as well as on PC. The main idea is to build a XUL application, using Mozilla XULRunner, around the AXMEDIS Plug-in for Mozilla.

This architecture is based on:

- XUL files, for describing the user interface in an XML format, with tags such as <button>, <menuitem>, etc,
- Javascript, to implement the actions from the user interface
- The AXMEDIS Mozilla plug-in, to manage the rendering of AXMEDIS objects
- XPCOM to interface C++ and Javascript.

By design, the application is localizable and skin-able, and, if the C++ code and the plug-in are cross-platform, then the application is also cross-platform.

The player should provide user interface for the methods and properties exposed by the plug-in.

11.2 Module Design in terms of Classes

Each window of the module is defined by a XUL file, and, eventually a javascript file.

The main window, defined in axmozillaplayer.xul, also includes “content.html” which is the HTML page using the plug-in.

The “about box” is defined in “about.xul”.

The window.js file contains the window related commands:

```
function doClose()
function doMinimize()
function doMaximize()
function doRestore()
function toggleMaximizedNormal()
```

The axmozillaplayere.js file contains the functions related to the management of the player.
Displays the about box:

```
function about()
```

Handles the display of the toolbar and the status bar

```
function toggleViewToolbar()
function toggleViewStatusbar()
```

Opens an AXMEDIS file

```
function openFile()
```

Gets the player

```
function getInternalPlayer()
```

Calls the play, pause and stop commands on the player

```
function play()
function pause()
function stop()
```

Calls the goNext, goPrev commands on the player

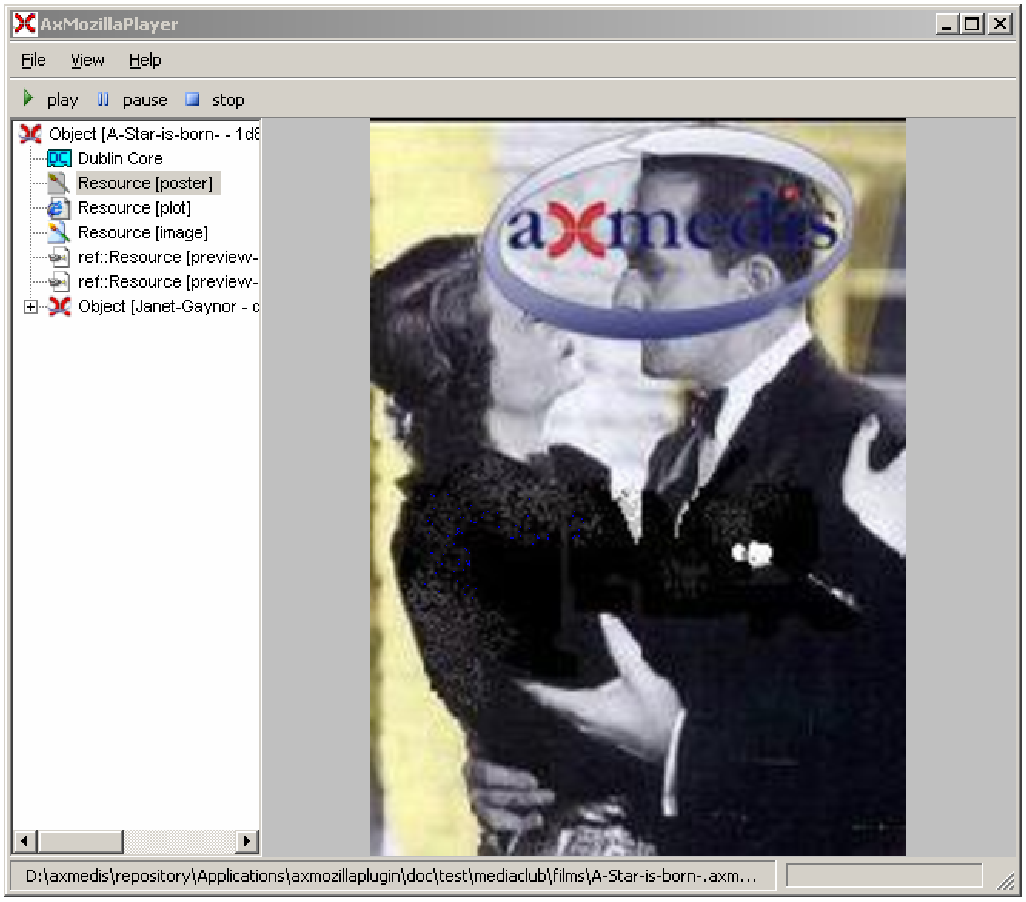
```
function goNext()
function goPrev()
```

Shows the hierarchy when hidden otherwise hides it

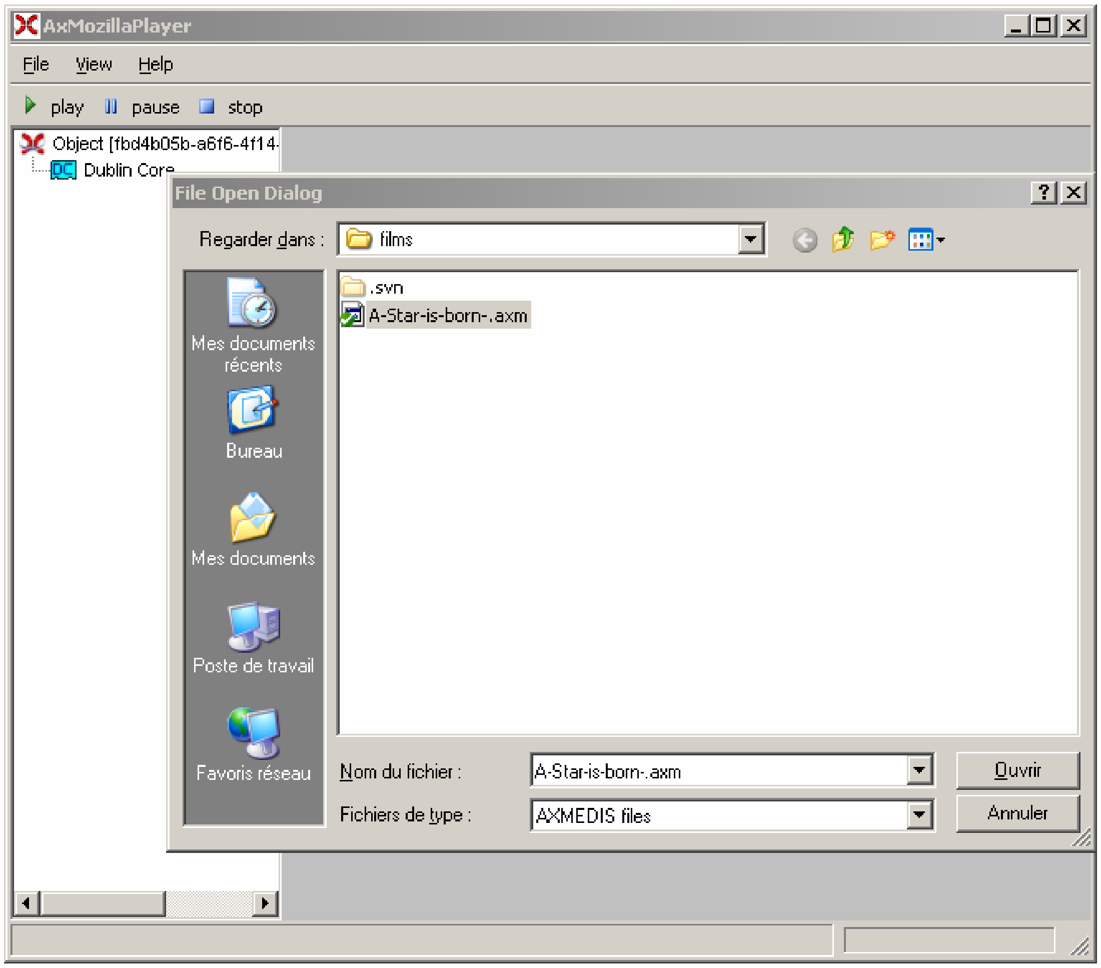
```
function toggleViewHierarchy()
```

11.3 User interface description

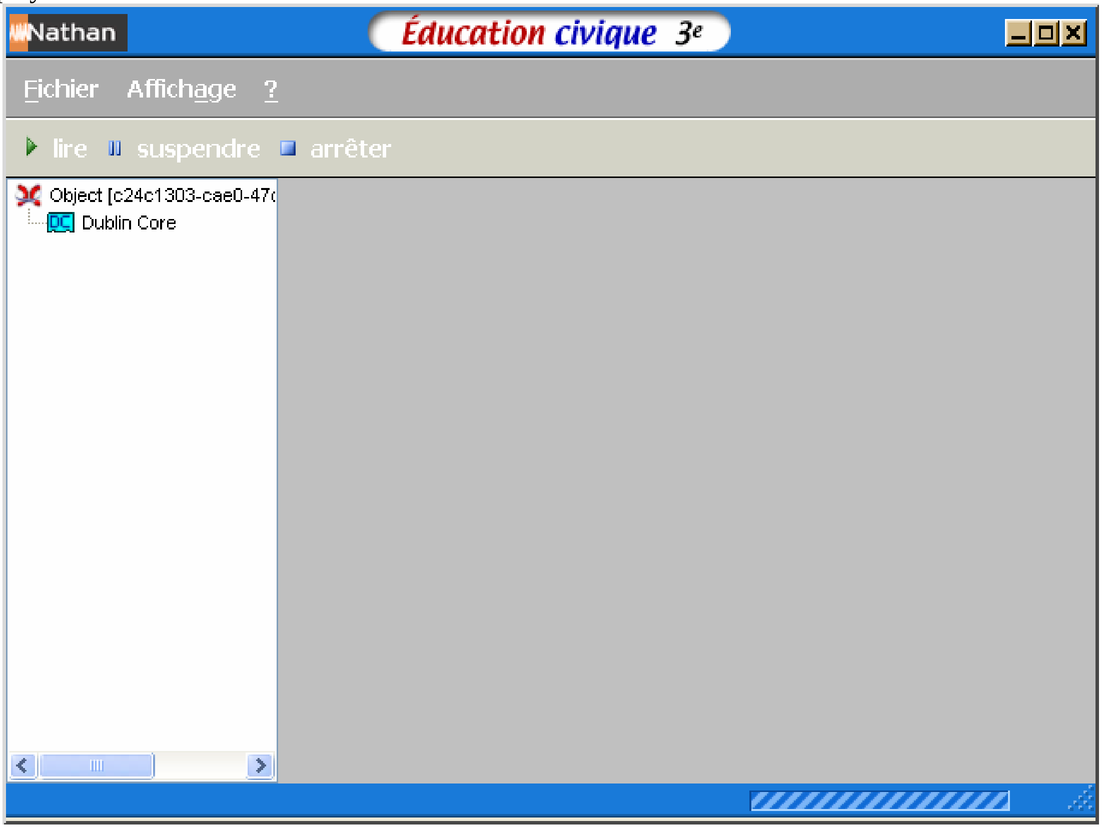
The player may look like the picture below, with standard menus, toolbars, and status, with the plug-in in the center of the application.



For instance, File > Open will open the standard “File Open” dialog box, filtering files using AXMEDIS objects extension.


AxMozillaPlayer
File View Help
play pause stop
Object [fbd4b05b-a6f6-4f14-
Dublin Core
File Open Dialog
Regarder dans : films
.svn
A-Star-is-born-.axm
Mes documents récents
Bureau
Mes documents
Poste de travail
Favoris réseau
Nom du fichier : A-Star-is-born-.axm
Fichiers de type : AXMEDIS files
Ouvrir
Annuler

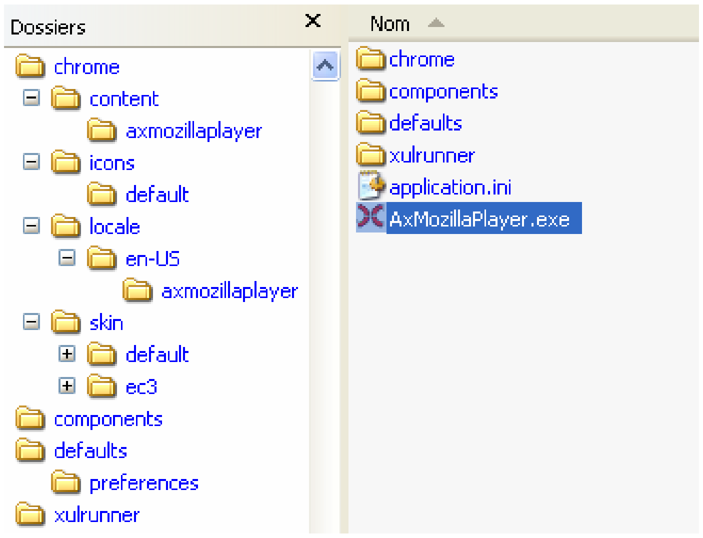
The player is skin-able and localizable:



11.4 Technical and Installation information

The installation is pretty straightforward as copying the whole AxMozillaPlayer directory is enough. Still, it is possible to create an XPI script for helping through the installation process.

The hierarchy of files must be as described below:



The application.ini file is required by XULrunner to define the name of the application, its vendor, its version number and the minimal and maximal version of gecko that is required for the application to run.

The defaults\preferences folder contains the axmozillaplayer-prefs.js file which specifies the starting URL of the application.

The Chrome directory contains the XUL files, the javascript files, the icons and the skins and localizable files.
The chrome.manifest file specifies where is located the content, which file to use for locale and which skin to use.

```
      content axmozillaplayer file:content/axmozillaplayer/
      locale axmozillaplayer en-US file:locale/en-US/axmozillaplayer/
      skin axmozillaplayer ec3 file:skin/ec3/axmozillaplayer/
```

The XUL files looks like:

```
<?xml version="1.0" ?>
<!DOCTYPE window SYSTEM "chrome://AxMozillaPlayer/locale/AxMozillaPlayer.dtd" [
<!ENTITY % skin SYSTEM "chrome://AxMozillaPlayer/skin/entities.ent">
   %skin;
]>
<?xml-stylesheet href="chrome://AxMozillaPlayer/skin/global.css"
type="text/css"?>
<?xul-overlay href="commands-overlay.xul" ?>
<?xul-overlay href="caption-overlay.xul" ?>
<window xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
      id="axmozillaplayer"
      title="&application.title;"
      width="640"
      height="480"
      hidechrome="&application.hidechrome;"
      orient="vertical"
      persist="width height screenX screenY"
      debug="true"
>
<script src="axmozillaplayer.js" />
<script src="io.js"/>
<script src="window.js"/>
<commandset id="windows-commands" />
<keyset>
  <key id="open-key" modifiers="accel" key="O" oncommand="openFile();"/>
  <key id="about-key" keycode="VK_F1" oncommand="about();"/>
  <key id="hierarchy-key" modifiers="accel" key="H"
oncommand="toggleViewHierarchy();"/>
</keyset>
<vbox id="application-caption" />
<toolbox>
  <menubar id="application-menubar">
    <!-- definition of the menu -->
  </menubar>
  <toolbar id="application-toolbar">
    <!-- definition of the toolbar -->
  </toolbar>
</toolbox>
<box flex="1">
  <iframe flex="1" name="player" src="content.html" />
</box>
<statusbar align="start" id="application-statusbar">
    <!-- definition of the statusbar -->
</statusbar>
</window>
```

The code of the command is defined in the associated javascript file.

The doctype specify the DTD in `chrome://AxMozillaPlayer/locale/AxMozillaPlayer.dtd` directory and therefore it uses the settings from the chrome.manifest.
All the elements localizable should be defined as entities in this DTD, like:

```
<!ENTITY win.minimize        "Minimize">
<!ENTITY win.maximize        "Maximize">
<!ENTITY win.restore         "Restore">
<!ENTITY win.close           "Close">

<!ENTITY application.title   "AxMozillaPlayer">
<!ENTITY application.back    "Back">
<!ENTITY application.forward "Forward">
<!ENTITY application.reload  "Reload">
<!ENTITY application.stop    "Stop">
<!ENTITY application.go      "Go">
<!ENTITY application.console "Console">
```

The skin is defined in the css files of the skin folder, along with the associated pictures. As choosing whether to display the standard frame of the window or not, the entity &application.hidechrome; is defined in `chrome://AxMozillaPlayer/skin/entities.ent.`

References to other major components needed	AXMEDIS Plug-in for Mozilla
Problems not solved	• When changing the skin, xulrunner do not detect that the entities.ent file is different, therefore the file must be touched to have the hidechrome value refreshed.
Configuration and execution context	

11.5 Draft User Manual

The player looks like a standard application, with a menu. The commands available so far are:

File > Open	Opens a new AXMEDIS object
File > Exit	Closes the application
View > Toolbar	Shows/hides the main toolbar
View > Status	Shows/hides the status bar
View > Hierarchy	Shows/hides the hierarchy tree
Help > About	Displays the about box

11.6 Integration and compilation issues

There is no compilation issues as:

- XUL is interpreted
- Javascript is interpreted
- The AXMEDIS Mozilla’s plug-in is compiled
- XUL runner is compiled