



Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE3.1.2.2.4

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

Version: 2.1 derived from 2.0 plus review of Visual and Behavioral aspects

Date: 23-06-2006

Responsible: DSI (revised and approved by coordinator)

Project Number: IST-2-511299

Project Title: AXMEDIS

Deliverable Type: report

Visible to User Groups: yes

Visible to Affiliated: yes

Visible to the Public: yes

Deliverable Number: DE3.1.2.2.4

Contractual Date of Delivery: M18

Actual Date of Delivery: 21/08/2006

Title of Deliverable: Specification of AXMEDIS Editors and Viewers, first update of DE3.1.2 partB

Work-Package contributing to the Deliverable: WP3.1

Task contributing to the Deliverable: WP3, WP2

Nature of the Deliverable: report

Author(s): DSI, FUPF, EPFL, UNIVLEEDS, FHGIGD, SEJER

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

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

AXMEDIS Copyright Notice

The following terms (including future possible amendments) set out the rights and obligations licensee will be requested to accept on entering into possession of any official AXMEDIS document either by downloading it from the web site or by any other means.

Any relevant AXMEDIS document includes this license. PLEASE READ THE FOLLOWING TERMS CAREFULLY AS THEY HAVE TO BE ACCEPTED PRIOR TO READING/USE OF THE DOCUMENT.

1. DEFINITIONS

- i. **"Acceptance Date"** is the date on which these terms and conditions for entering into possession of the document have been accepted.
- ii. **"Copyright"** stands for any content, document or portion of it that is covered by the copyright disclaimer in a Document.
- iii. **"Licensor"** is AXMEDIS Consortium as a de-facto consortium of the EC project and any of its derivations in terms of companies and/or associations, see www.axmedis.org
- iv. **"Document"** means the information contained in any electronic file, which has been published by the Licensor's as AXMEDIS official document and listed in the web site mentioned above or available by any other means.
- v. **"Works"** means any works created by the licensee, which reproduce a Document or any of its part.

2. LICENCE

1. The Licensor grants a non-exclusive royalty free licence to reproduce and use the Documents subject to present terms and conditions (the **Licence**) for the parts that are own and proprietary property the of AXMEDIS consortium or its members.
2. In consideration of the Licensor granting the Licence, licensee agrees to adhere to the following terms and conditions.

3. TERM AND TERMINATION

1. Granted Licence shall commence on Acceptance Date.
2. Granted Licence will terminate automatically if licensee fails to comply with any of the terms and conditions of this Licence.
3. Termination of this Licence does not affect either party's accrued rights and obligations as at the date of termination.
4. Upon termination of this Licence for whatever reason, licensee shall cease to make any use of the accessed Copyright.
5. All provisions of this Licence, which are necessary for the interpretation or enforcement of a party's rights or obligations, shall survive termination of this Licence and shall continue in full force and effect.
6. Notwithstanding License termination, confidentiality clauses related to any content, document or part of it as stated in the document itself will remain in force for a period of 5 years after license issue date or the period stated in the document whichever is the longer.

4. USE

1. Licensee shall not breach or denigrate the integrity of the Copyright Notice and in particular shall not:
 - i. remove this Copyright Notice on a Document or any of its reproduction in any form in which those may be achieved;
 - ii. change or remove the title of a Document;
 - iii. use all or any part of a Document as part of a specification or standard not emanating from the Licensor without the prior written consent of the Licensor; or
 - iv. do or permit others to do any act or omission in relation to a Document which is contrary to the rights and obligations as stated in the present license and agreed with the Licensor

5. COPYRIGHT NOTICES

1. All Works shall bear a clear notice asserting the Licensor's Copyright. The notice shall use the wording employed by the Licensor in its own copyright notice unless the Licensor otherwise instructs licensees.

6. WARRANTY

1. The Licensor warrants the licensee that the present licence is issued on the basis of full Copyright ownership or re-licensing agreements granting the Licensor full licensing and enforcement power.
2. For the avoidance of doubt the licensee should be aware that although the Copyright in the documents is given under warranty this warranty does not extend to the content of any document which may contain references or specifications or technologies that are covered by patents (also of third parties) or that refer to other standards. AXMEDIS is not responsible and does not guarantee that the information contained in the document is fully proprietary of AXMEDIS consortium and/or partners.
3. Licensee hereby undertakes to the Licensor that he will, without prejudice to any other right of action which the Licensor may have, at all times keep the Licensor fully and effectively indemnified against all and any liability (which liability shall include, without limitation, all losses, costs, claims, expenses, demands, actions, damages, legal and other professional fees and expenses on a full indemnity basis) which the Licensor may suffer or incur as a result of, or by reason of, any breach or non-fulfillment of any of his obligations in respect of this License.

7. INFRINGEMENT

1. Licensee undertakes to notify promptly the Licensor of any threatened or actual infringement of the Copyright which comes to licensee notice and shall, at the Licensor's request and expense, do all such things as are reasonably necessary to defend and enforce the Licensor's rights in the Copyright.

8. GOVERNING LAW AND JURISDICTION

1. This Licence shall be subject to, and construed and interpreted in accordance with Italian law.
2. The parties irrevocably submit to the exclusive jurisdiction of the Italian Courts.

Please note that:

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

Table of Content

1	EXECUTIVE SUMMARY AND REPORT SCOPE	9
1.1	THIS DOCUMENT CONCERNS	10
1.2	LIST OF MODULES OR EXECUTABLE TOOLS SPECIFIED IN THIS DOCUMENT	10
1.3	LIST OF FORMATS SPECIFIED IN THIS DOCUMENT	10
2	GENERAL USE CASES AND SCENARIOS.....	12
2.1	USE CASE – CREATION OF A NEW AXMEDIS OBJECT	12
2.2	USE CASE - CREATION OF COMPOSITE AXMEDIS OBJECT	12
3	GENERAL ARCHITECTURE AND RELATIONSHIPS AMONG THE MODULES PRODUCED (DSI) 13	
3.1	VIEW MODULES (DRM EDIT/VIEW, HIERARCHY EDIT/VIEW, METADATA EDIT/VIEW, ETC...).....	13
4	EXECUTABLE TOOL - AXMEDIS EDITOR (DSI).....	15
4.1	GENERAL DESCRIPTION OF THE MODULE.....	16
4.1.1	Software Architecture	19
4.2	MODULE DESIGN IN TERMS OF CLASSES	19
4.3	USER INTERFACE DESCRIPTION	22
4.4	TECHNICAL AND INSTALLATION INFORMATION	27
4.5	DRAFT USER MANUAL	27
4.5.1	Create a new AXMEDIS Object	27
4.5.2	Modifying an AXMEDIS Object stored on Database.....	30
4.6	EXAMPLES OF USAGE	33
4.7	INTEGRATION AND COMPILATION ISSUES.....	33
4.8	CONFIGURATION PARAMETERS.....	33
4.9	ERRORS REPORTED AND THAT MAY OCCUR	33
5	MODULE - HIERARCHY EDITOR AND VIEWER (DSI)	33
5.1	GENERAL DESCRIPTION OF THE MODULE.....	35
5.2	MODULE DESIGN IN TERMS OF CLASSES	37
5.3	USER INTERFACE DESCRIPTION	37
5.4	TECHNICAL AND INSTALLATION INFORMATION	38
5.5	DRAFT USER MANUAL	38
5.6	EXAMPLES OF USAGE	38
5.7	INTEGRATION AND COMPILATION ISSUES.....	38
5.8	CONFIGURATION PARAMETERS.....	38
5.9	ERRORS REPORTED AND THAT MAY OCCUR	38
6	MODULE DRM EDITOR AND VIEWER (FUPF).....	40
6.1	GENERAL DESCRIPTION OF THE MODULE.....	41
6.1.1	Form of the DRM Editor & Viewer	41
6.1.2	Architecture.....	42
6.1.3	Functionalities	42
6.1.4	DRM Editor Business Logic.....	42
6.2	MODULE DESIGN IN TERMS OF CLASSES	42
6.3	USER INTERFACE DESCRIPTION	43
6.3.1	DRM Viewer.....	44
6.4	TECHNICAL AND INSTALLATION INFORMATION	45
7	MODULE - PROTECTION EDITOR AND VIEWER (FHGIGD)	45
7.1	GENERAL DESCRIPTION OF THE MODULE.....	47
7.2	MODULE DESIGN IN TERMS OF CLASSES	48
7.3	USER INTERFACE DESCRIPTION	48
7.4	TECHNICAL AND INSTALLATION INFORMATION	49
7.5	DRAFT USER MANUAL AND EXAMPLE OF USAGE	50
7.6	INTEGRATION AND COMPILATION ISSUES.....	50

7.7	CONFIGURATION PARAMETERS.....	50
7.8	ERRORS REPORTED AND THAT MAY OCCUR	50
8	MODULE VISUAL EDITOR AND VIEWER (EPFL).....	50
8.1	GENERAL DESCRIPTION OF THE MODULE.....	52
8.1.1.1	Hierarchy Authoring of SMIL with Hierarchy Editor.....	52
8.1.1.2	Load Media Resources from AXMEDIS Object with SMIL Hierarchy Editor.	53
8.1.1.3	Spatial Visual Authoring of SMIL with Visual Editor.....	53
8.1.1.4	Loading Media Resources from SMIL Hierarchy Structure with Visual Editor	55
8.2	MODULE DESIGN IN TERMS OF CLASSES	56
8.3	USER INTERFACE DESCRIPTION	56
8.4	TECHNICAL AND INSTALLATION INFORMATION	58
8.5	DRAFT USER MANUAL.....	59
8.5.1	Editing the visual scene for SMIL (create/delete, resize/move).....	59
8.5.2	Association of media resources within an element of SMIL	59
8.6	EXAMPLES OF USAGE	60
8.7	INTEGRATION AND COMPILATION ISSUES.....	60
8.8	CONFIGURATION PARAMETERS.....	60
8.9	ERRORS REPORTED AND THAT MAY OCCUR	60
9	MODULE BEHAVIOUR AND FUNCTIONAL EDITOR AND VIEWER (EPFL).....	60
9.1	GENERAL DESCRIPTION OF THE MODULE.....	62
9.1.1	Behaviour Business Logic	65
9.2	MODULE DESIGN IN TERMS OF CLASSES	68
9.3	USER INTERFACE DESCRIPTION	68
9.4	TECHNICAL AND INSTALLATION INFORMATION	70
9.5	DRAFT USER MANUAL.....	70
9.5.1	Editing the temporal information of media resources.....	70
9.6	EXAMPLES OF USAGE	71
9.7	INTEGRATION AND COMPILATION ISSUES.....	71
9.8	CONFIGURATION PARAMETERS.....	71
9.9	ERRORS REPORTED AND THAT MAY OCCUR	71
10	MODULE AXMEDIS OBJECT EDITOR AND VIEWER (DESCRIPTIONS AND COMMENTS) (EPFL, DSI).....	72
10.1	GENERAL DESCRIPTION OF THE MODULE.....	73
10.1.1	Business Logic.....	74
10.2	MODULE DESIGN IN TERMS OF CLASSES	75
10.3	USER INTERFACE DESCRIPTION	75
10.3.1	Main GUI.....	75
10.3.2	Configuration GUI.....	75
10.3.3	Renderer GUIs.....	76
10.4	DRAFT USER MANUAL.....	76
10.5	EXAMPLES OF USAGE	76
10.6	INTEGRATION AND COMPILATION ISSUES.....	76
10.7	CONFIGURATION PARAMETERS.....	76
10.8	ERRORS REPORTED AND THAT MAY OCCUR	76
11	MODULE METADATA EDITOR AND VIEWER (UNIVLEEDS).....	76
11.1	GENERAL DESCRIPTION OF THE MODULE.....	78
11.1.1	General Metadata Business Logic.....	81
11.1.2	Metadata Manager	81
11.1.3	Metadata Schemas	81
11.1.4	Metadata Viewer and Renderer.....	82
11.2	MODULE DESIGN IN TERMS OF CLASSES	82
11.3	USER INTERFACE DESCRIPTION	82
11.4	TECHNICAL AND INSTALLATION INFORMATION	84
11.5	DRAFT USER MANUAL.....	84
11.6	EXAMPLES OF USAGE	84
11.7	INTEGRATION AND COMPILATION ISSUES.....	84
11.8	CONFIGURATION PARAMETERS.....	84

11.9	ERRORS REPORTED AND THAT MAY OCCUR	84
12	MODULE - METADATA MAPPER EDITOR AND VIEWER (UNIVLEEDS).....	84
12.1.1	General Description of the Module.....	86
12.1.2	Module Design in terms of Classes.....	87
12.1.3	User interface description.....	87
12.1.4	Technical and Installation information	87
12.1.5	Draft User Manual.....	88
12.1.6	Examples of usage.....	88
12.1.7	Integration and compilation issues.....	88
12.1.8	Configuration Parameters.....	88
12.1.9	Errors reported and that may occur.....	88
13	MODULE WORKFLOW EDITOR AND VIEWER (DSI).....	88
13.1	GENERAL DESCRIPTION OF THE MODULE.....	90
13.2	MODULE DESIGN IN TERMS OF CLASSES	90
13.3	USER INTERFACE DESCRIPTION	90
13.4	TECHNICAL AND INSTALLATION INFORMATION	91
13.5	DRAFT USER MANUAL.....	91
13.6	EXAMPLES OF USAGE	91
13.7	INTEGRATION AND COMPILATION ISSUES.....	91
13.8	CONFIGURATION PARAMETERS.....	91
13.9	ERRORS REPORTED AND THAT MAY OCCUR	92
14	MODULE - AXMEDIS CONTENT TOOL ERROR MANAGER (DSI).....	92
14.1	GENERAL DESCRIPTION OF THE MODULE.....	94
14.2	MODULE DESIGN IN TERMS OF CLASSES	94
14.3	USER INTERFACE DESCRIPTION	94
14.4	DRAFT USER MANUAL.....	95
14.5	EXAMPLES OF USAGE	95
14.6	CONFIGURATION PARAMETERS.....	95
14.7	ERRORS REPORTED AND THAT MAY OCCUR	95
15	MODULE - AXMEDIS EDITOR CONFIGURATION MANAGER (DSI, EPFL)	96
15.1	GENERAL DESCRIPTION OF THE MODULE.....	97
15.2	MODULE DESIGN IN TERMS OF CLASSES	97
15.3	USER INTERFACE DESCRIPTION	100
15.4	TECHNICAL AND INSTALLATION INFORMATION	101
15.5	EXAMPLES OF USAGE	101
15.6	INTEGRATION AND COMPILATION ISSUES.....	101
15.7	CONFIGURATION PARAMETERS.....	101
15.8	ERRORS REPORTED AND THAT MAY OCCUR	102
16	MODULE - AXMEDIS EDITOR PLUG-IN MANAGER (DSI, EPFL).....	103
16.1	GENERAL DESCRIPTION OF THE MODULE.....	104
16.2	MODULE DESIGN IN TERMS OF CLASSES	105
16.2.1	Fundamental classes overview	106
16.3	INTEGRATION AND COMPILATION ISSUES.....	106
16.4	CONFIGURATION PARAMETERS.....	106
16.5	ERRORS REPORTED AND THAT MAY OCCUR	107
17	MODULE - AXOM CONTENT PROCESSING (DSI, EPFL)	108
17.1	GENERAL DESCRIPTION OF THE MODULE.....	109
17.2	MODULE DESIGN IN TERMS OF CLASSES	110
17.2.1	Plug-in function parameters class hierarchy	111
17.3	ERRORS REPORTED AND THAT MAY OCCUR	112
18	MODULE - AXOM COMMANDS AND REPORTING (DSI, EPFL)	112
18.1	GENERAL DESCRIPTION OF THE MODULE.....	114
18.2	MODULE DESIGN IN TERMS OF CLASSES	114

18.2.1	Different application dependent workflows interfaces.....	116
18.3	TECHNICAL AND INSTALLATION INFORMATION	117
18.4	DRAFT USER MANUAL.....	117
18.5	EXAMPLES OF USAGE	117
18.6	INTEGRATION AND COMPILATION ISSUES.....	117
18.7	CONFIGURATION PARAMETERS.....	117
18.8	ERRORS REPORTED AND THAT MAY OCCUR	118
19	MODULE - INTERNAL AUDIO PLAYER (DSI)	119
19.1	GENERAL DESCRIPTION OF THE MODULE.....	120
19.2	MODULE DESIGN IN TERMS OF CLASSES	121
19.3	USER INTERFACE DESCRIPTION	121
19.4	TECHNICAL AND INSTALLATION INFORMATION	122
19.5	DRAFT USER MANUAL.....	122
19.6	EXAMPLES OF USAGE	122
19.7	INTEGRATION AND COMPILATION ISSUES.....	122
19.8	CONFIGURATION PARAMETERS.....	122
19.9	ERRORS REPORTED AND THAT MAY OCCUR	123
20	MODULE - INTERNAL IMAGE VIEWER (DSI)	123
20.1	GENERAL DESCRIPTION OF THE MODULE.....	124
20.2	MODULE DESIGN IN TERMS OF CLASSES	125
20.3	USER INTERFACE DESCRIPTION	126
20.4	TECHNICAL AND INSTALLATION INFORMATION	127
20.5	DRAFT USER MANUAL.....	127
20.6	EXAMPLES OF USAGE	127
20.7	INTEGRATION AND COMPILATION ISSUES.....	127
20.8	CONFIGURATION PARAMETERS.....	127
20.9	ERRORS REPORTED AND THAT MAY OCCUR	127
21	MODULE - INTERNAL VIDEO PLAYER (DSI)	128
21.1	GENERAL DESCRIPTION OF THE MODULE.....	129
21.2	MODULE DESIGN IN TERMS OF CLASSES	130
21.3	USER INTERFACE DESCRIPTION	130
21.4	TECHNICAL AND INSTALLATION INFORMATION	131
21.5	DRAFT USER MANUAL.....	131
21.6	EXAMPLES OF USAGE	131
21.7	INTEGRATION AND COMPILATION ISSUES.....	131
21.8	CONFIGURATION PARAMETERS.....	131
21.9	ERRORS REPORTED AND THAT MAY OCCUR	131
22	MODULE - INTERNAL MPEG4 PLAYER (EPFL).....	133
22.1	GENERAL DESCRIPTION OF THE MODULE.....	134
22.2	INTEGRATION OF MPEG-4 IPMP eXTENSIONS INTO THE MPEG-4 PLAYER	136
22.2.1	Architectural elements of the IPMP-X framework.....	137
	The Message Router.....	137
	The Tool Manager.....	138
	IPMP Tools.....	138
	Normative messages.....	139
	THE AXMEDIS IPMP-X INTERFACE	140
22.2.2	Examples of scenarios to be demonstrated.	140
22.3	TRANSLATION OF MPEG-4 IPMPX BINARY DESCRIPTORS TO XML BASED MPEG-21 IPMP COMPONENTS	141
22.3.1	DMP Content Information overview	141
22.3.2	MPEG-4 IPMPX descriptors.....	141
22.3.3	IPMPX translation to XML	142
22.4	MODULE DESIGN IN TERMS OF CLASSES	148
23	MODULE - INTERNAL SMIL PLAYER (EPFL).....	151
23.1	GENERAL DESCRIPTION OF THE MODULE.....	152
23.2	MODULE DESIGN IN TERMS OF CLASSES	155

23.3	USER INTERFACE DESCRIPTION	157
23.4	TECHNICAL AND INSTALLATION INFORMATION	157
23.5	DRAFT USER MANUAL	157
23.6	EXAMPLES OF USAGE	157
23.7	INTEGRATION AND COMPILATION ISSUES.....	157
23.8	CONFIGURATION PARAMETERS.....	157
23.9	ERRORS REPORTED AND THAT MAY OCCUR	157
24	MODULE - INTERNAL DOCUMENT VIEWER (DSI)	159
24.1	GENERAL DESCRIPTION OF THE MODULE.....	160
24.1.1	HTML	160
24.1.2	MSWord Documents	162
24.1.3	PDF	162
24.1.4	Postscript.....	162
24.2	MODULE DESIGN IN TERMS OF CLASSES	163
24.2.1	Protocol handlers	163
24.2.2	AxIEDocumentViewer and AxMozillaDocumentViewer.....	163
24.3	USER INTERFACE DESCRIPTION	164
24.4	TECHNICAL AND INSTALLATION INFORMATION	164
24.5	DRAFT USER MANUAL	164
24.6	EXAMPLES OF USAGE	164
24.7	INTEGRATION AND COMPILATION ISSUES.....	164
24.8	CONFIGURATION PARAMETERS.....	164
24.9	ERRORS REPORTED AND THAT MAY OCCUR	165
25	TOOL – MPEG4 PLAYER (EPFL)	165
25.1	GENERAL DESCRIPTION OF THE MODULE.....	166
25.2	USER INTERFACE DESCRIPTION	167
25.3	TECHNICAL AND INSTALLATION INFORMATION	169
25.4	DRAFT USER MANUAL	170
25.5	EXAMPLES OF USAGE	170
25.6	INTEGRATION AND COMPILATION ISSUES.....	170
25.7	CONFIGURATION PARAMETERS.....	170
25.8	ERRORS REPORTED AND THAT MAY OCCUR	170
26	FORMAL DESCRIPTION OF FORMAT – ERROR CODING (DSI)	170
27	FORMAL DESCRIPTION OF FORMAT – ERROR LOG (DSI).....	171
28	FORMAL DESCRIPTION OF FORMAT – CONFIGURATION (DSI)	174
29	FORMAL DESCRIPTION OF FORMAT – PLUG-INS DESCRIPTION (DSI).....	175
30	FORMAL DESCRIPTION OF FORMAT – CONTENT PROCESSING PLUG-INS SPECIFIC DESCRIPTION (DSI).....	178
31	FORMAL DESCRIPTION OF FORMAT – PARAMETER DESCRIPTION	183

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	DSI (in place of 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	EXITECH
DE3.1.2.2.10	Specification of AXMEDIS P2P tools, AXEPTTool and AXMEDIS, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-10-Spec-of-AXEPTTool-and-AXMEDIA-tools-v1-0.doc	DSI (in place of CRS4)
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	EXITECH

1.1 This document concerns

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

1.2 List of Modules or Executable Tools Specified in this document

A module is a component that can be or it is reused in other cases or points of the AXMEDIS framework or of other AXMEDIS based solutions.

The modules/tools have to include effective components and/or tools and also testing components and tools.

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

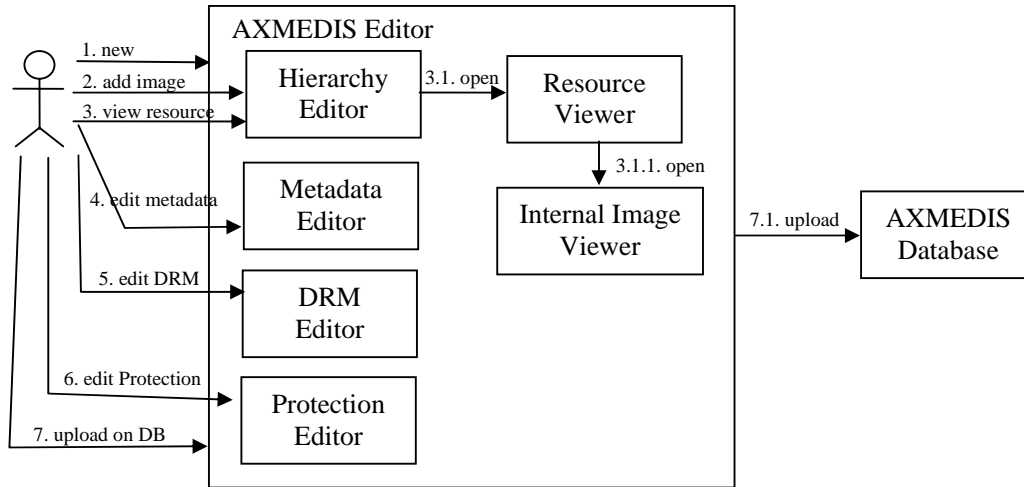
1.3 List of Formats Specified in this document

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

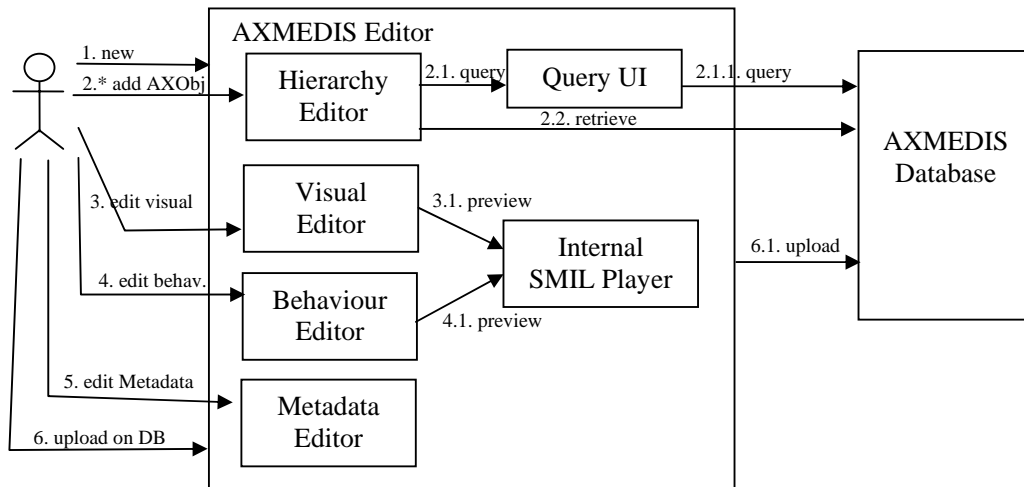
Format Name	Format Description and purpose, state also in which other modules is used	Standards exploited if any
Error Coding	XML based format to code errors information	
Error Log	XML based format to represent the errors happened	
Configuration	XML based format to represent the configuration information of AXMEDIS Tools	
Plug-in description	XML based format to represent functionalities provided by plug-ins	
Content Processing Plug-in specific description	XML based format for content-processing plug-ins functionalities	
Parameter description	XML based format for parameters of plug-ins	

2 General Use Cases and scenarios

2.1 Use Case – Creation of a New AXMEDIS Object



2.2 Use Case - Creation of Composite AXMEDIS Object

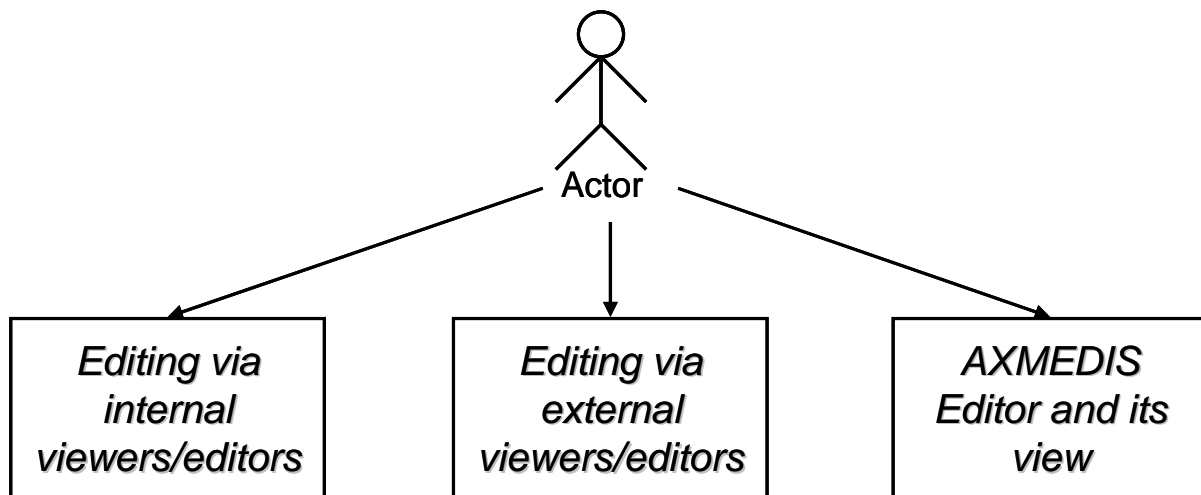


3 General architecture and relationships among the modules produced (DSI)

In this document everything concerning how to handle (play/edit) AXMEDIS content is detailed. Different scenarios of utilization are considered in the following sections. The beginning part put the basis to the construction of the AXMEDIS editor: all the editor/viewers that show different features of the structured content inside an AXMEDIS object are specified. These AXMEDIS Viewer can browse structure (hierarchy), multimedia feature (metadata, behaviour, visual, object), protection and DRM. Even workflow aspects are manageable in a suitable user interface integrated with the AXMEDIS Editor.

Errors and configuration of the AXMEDIS Editor are considered in the specification of the Content Tool Error Manager and Configuration Manager. The AXMEDIS Editor has to be extendable in order to add new content processing capabilities and to interoperate with different already established workflow management services. The AXMEDIS Editor Plug-in Manager has been specified in this document, providing methods to plug-in new functions (profile description and dynamic linking).

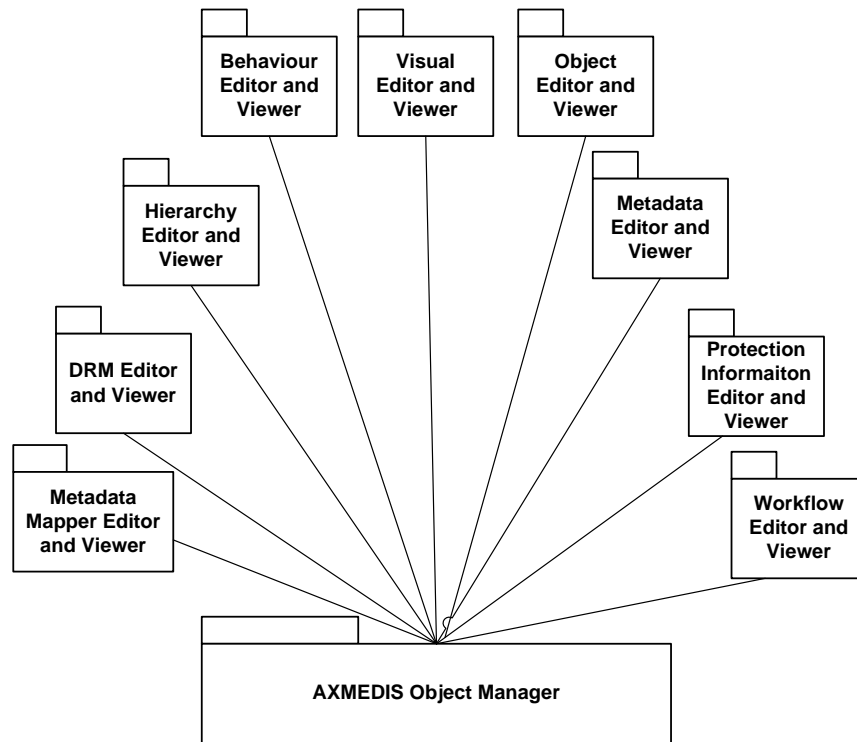
All the needed modules to render the multimedia resources included in the AXMEDIS objects (audio, video, images, documents, mpeg4, smil) are specified



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

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

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



4 Executable Tool - AXMEDIS Editor (DSI)

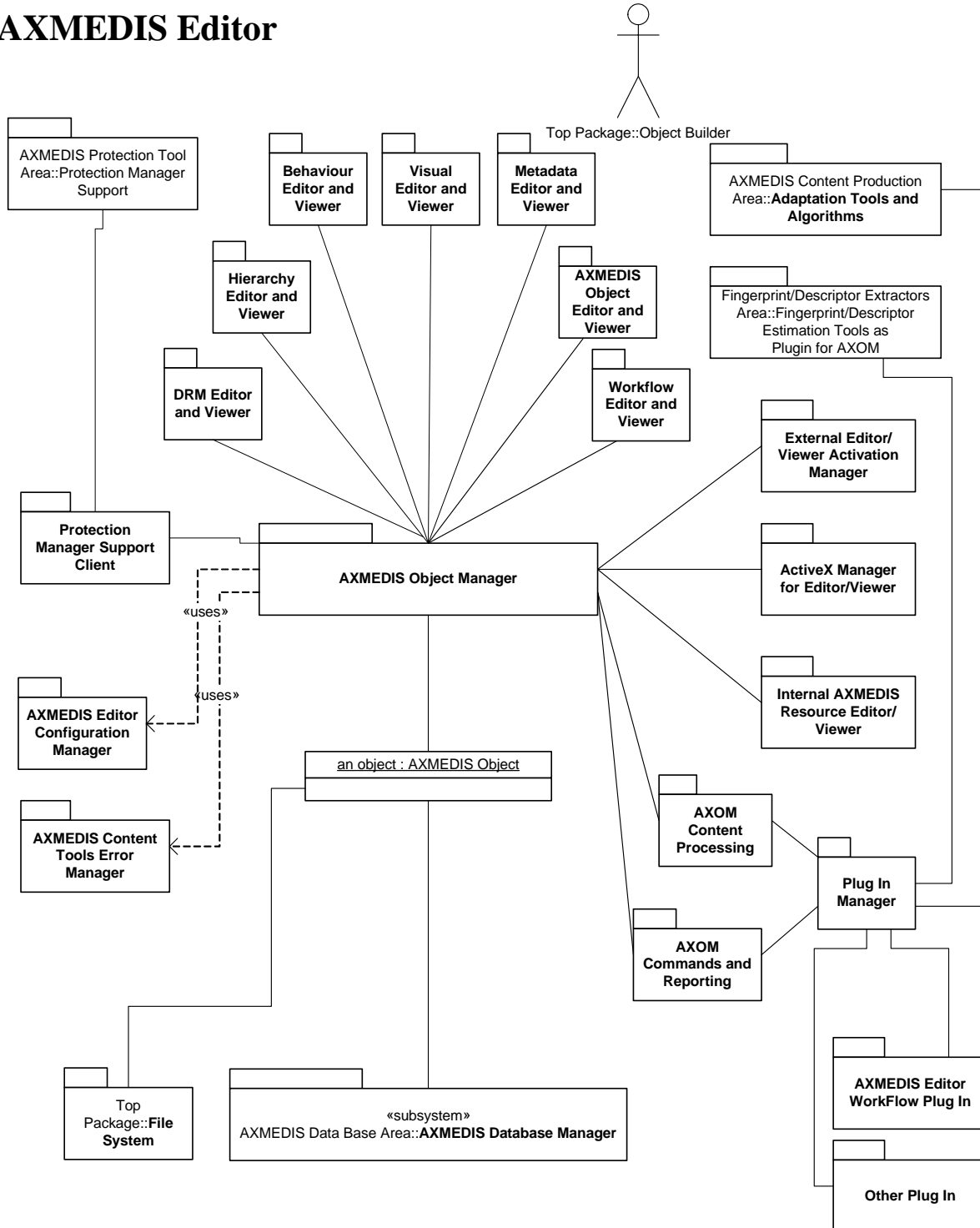
Module/Tool Profile		
AXMEDIS Editor		
Responsible Name	Bellini	
Responsible Partner	DSI	
Status (proposed/approved)		
Implemented/not implemented	Implemented	
Status of the implementation	80%	
Executable or Library/module (Support)	Executable	
Single Thread or Multithread	Multithread	
Language of Development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

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

4.1 General Description of the Module

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

AXMEDIS Editor



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

AXMEDIS Editor includes (or use) the following modules:

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

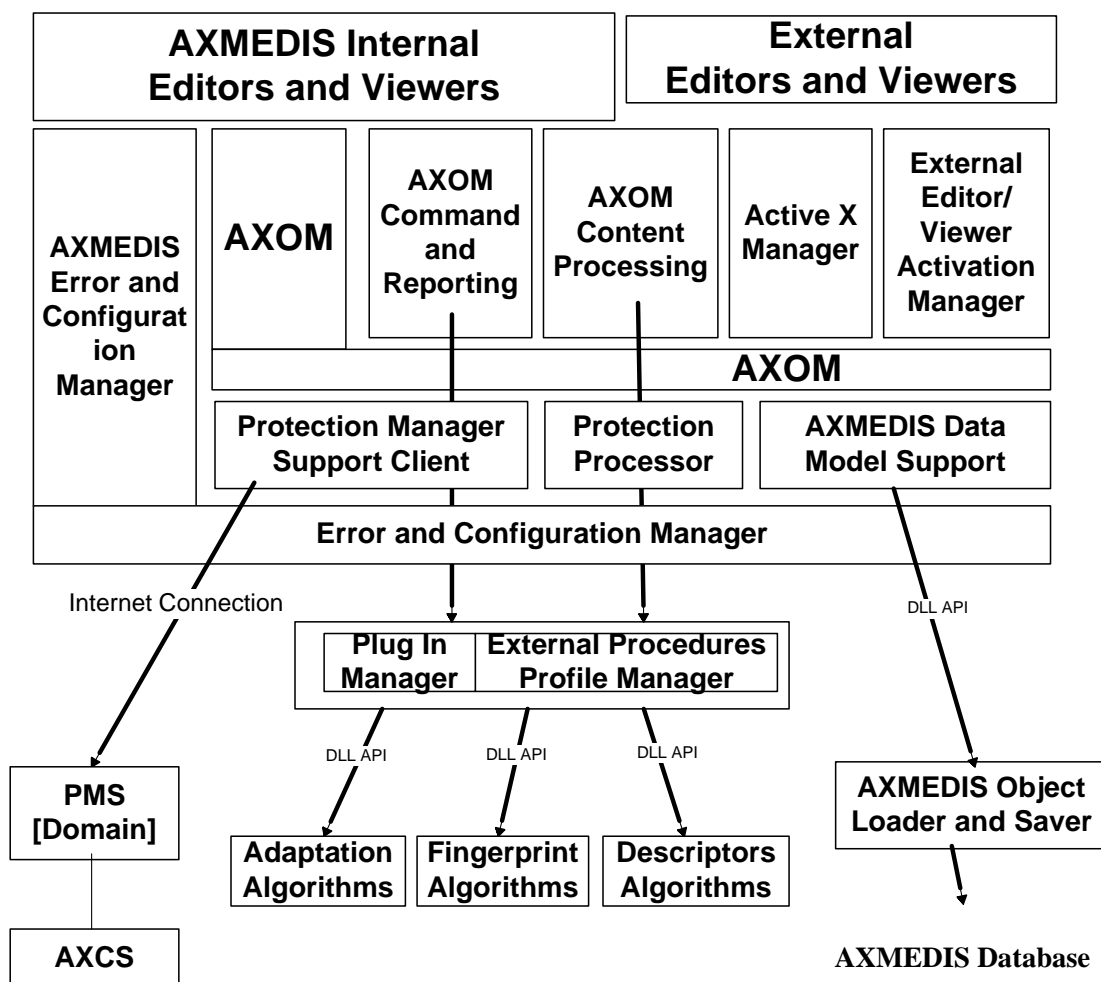
Further AXMEDIS Editor specifications are certainly:

- Each configurable AXMEDIS Editor modules shall conform to AXMEDIS Editor Configuration Manager requirements. Moreover, each configurable AXMEDIS Editor modules shall own a default settings set in order to work also on AXMEDIS Editor Configuration Manager fault or absence;
- AXMEDIS Editor and all its sub-modules shall support multi-language. Multi-language option will be managed through AXMEDIS Editor Configuration Manager User Interface;

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

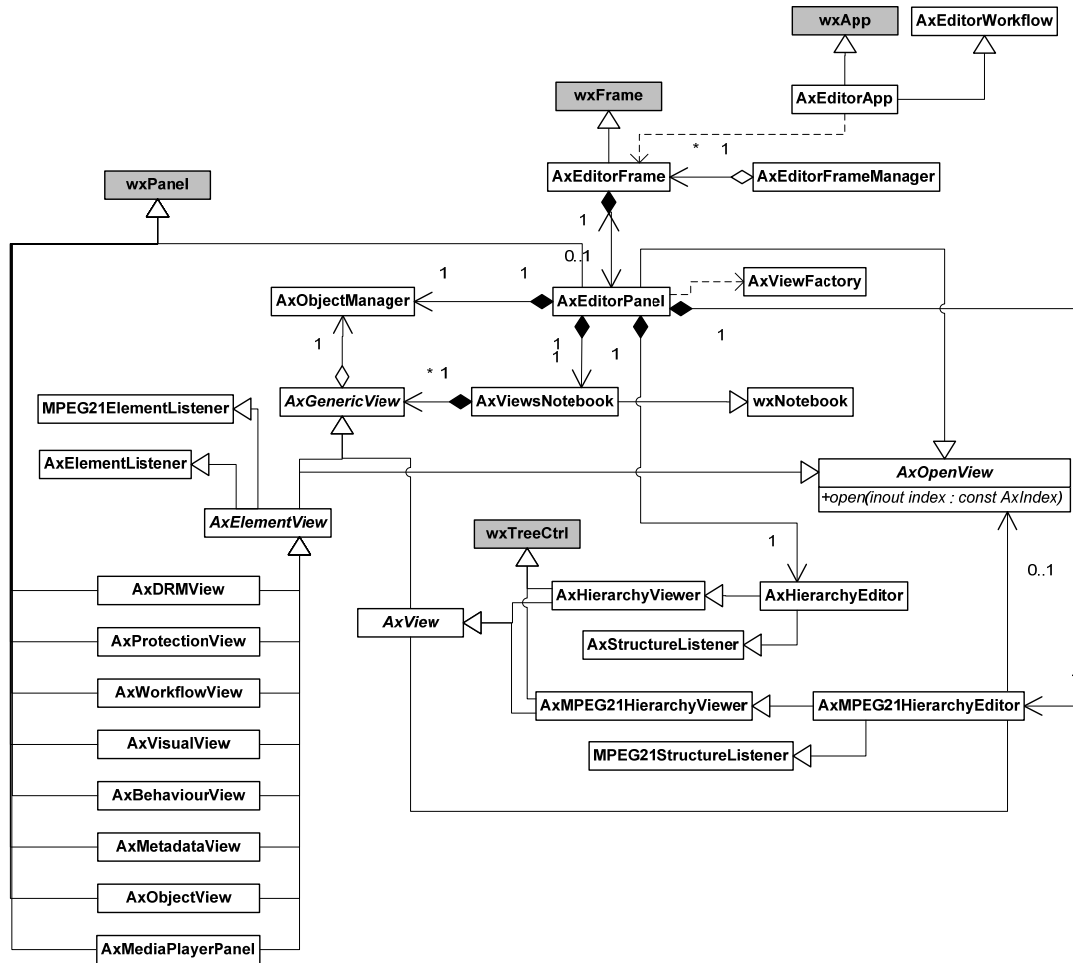
4.1.1 Software Architecture

AXMEDIS Editor Software Architecture



4.2 Module Design in terms of Classes

The following is the class diagram for the AXMEDIS Editor:

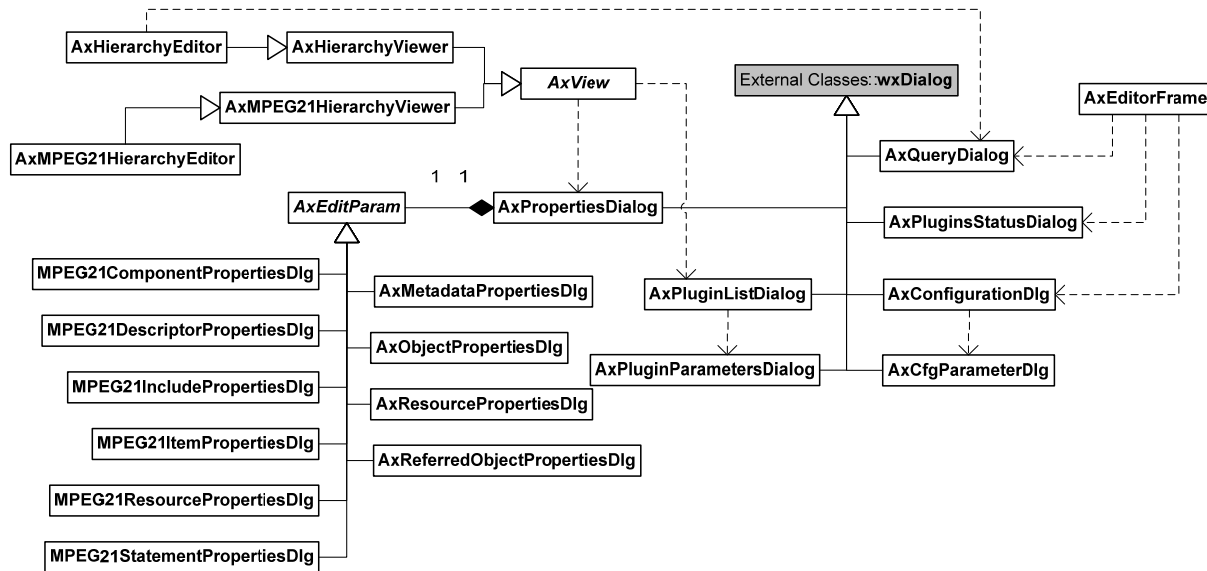


where:

- class *AxEditorApp* is the AXMEDIS Editor application, it creates the first *AxEditorFrame* and it receives the messages coming from the Workflow Manger using the workflow plugin.
- class *AxEditorFrame* is the main frame, it contains an *AxEditorPanel*
- class *AxEditorFrameManager* is a singleton class where each *AxEditorFrame* registers itself on creation. It allows to know which frames are currently open.
- class *AxOpenView* is an interface called when an element of an AXMEDIS Object have to be opened
- class *AxEditorPanel* contains:
 - an *AxObjectManager* used to manipulate the AXMEDIS object;
 - an *AxHierarchyEditor* and an *AxMPEG21HierarchyEditor*, tree controls used to manipulate the object structure
 - a set of views open for the object contained in *AxViewsNotebook*
 and it uses the *AxViewFactory* object to create the Views, it is derived from *AxOpenView* to open the view requested from the two hierarchy editors
- class *AxViewFactory* is a singleton class used to create the different views
- class *AxViewsNotebook* is used to contain different views of the object that can be hosted inside a Notebook, they can be: *AxDRMView*, *AxProtectionView*, *AxMetadataView*, *AxWorkflowView*, *AxVisualView*, *AxObjectView* or *AxBehaviourView*.
- Abstract class *AxGenericView* represents a view on an *AxObjectManager*, it is derived from *MPEG21ElementListener*, *MPEG21StructureListener*, *AxElementListener*, *AxStructureListener* to receive events when the object is manipulated.
- Abstract class *AxView* represents a view on the whole object (like a hierarchy view) and *AxElementView* is a view on a specific element of the object.

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

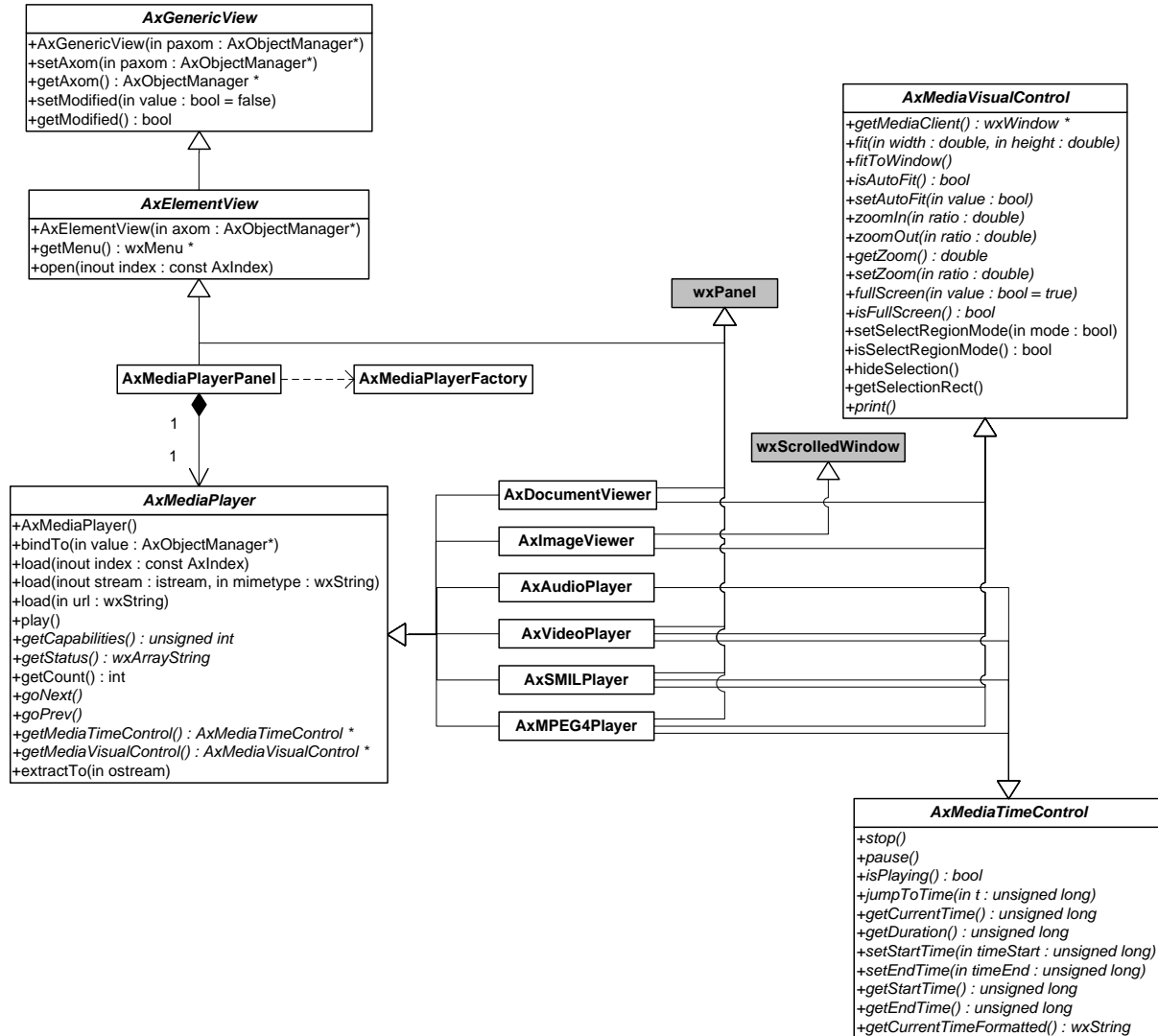
the following class diagram is on the Dialogs used:



where:

- class *AxQueryDialog* is used to make queries on the database, it is used from the *AxEditorFrame* to open an object from DB or from the *AxHierarchyEditor* to embed an AXMEDIS object coming from the DB.
- class *AxPluginStatusDialog* it is used from the *AxEditorFrame* to show the plugins installed
- class *AxPropertiesDialog* it is used by the *AxView* to show/edit the properties of the MPEG21/AXMEDIS elements
- class *AxPluginDialog* is used from the *AxView* to use a content processing plugin on a resource, the dialog shows the plugins that can be used and using the *AxParameterDialog* the plugin parameters can be set and the plugin called.
- class *AxConfigurationDlg* is used from the *AxEditorFrame* to edit the configuration parameters

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

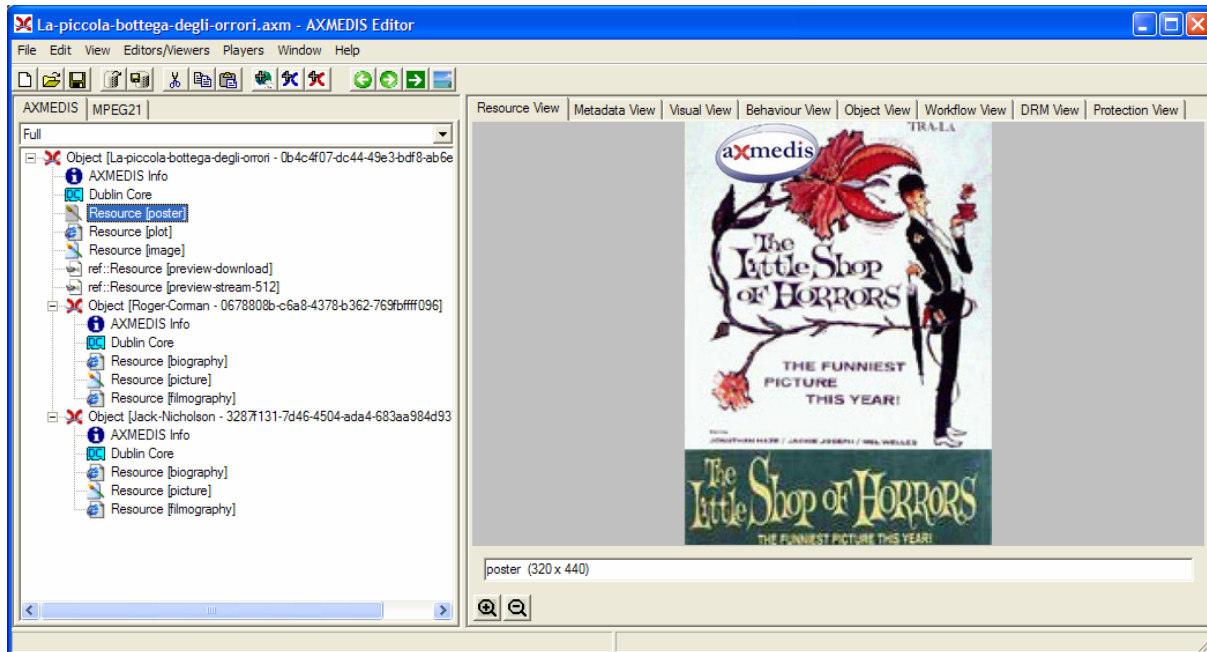


where:

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

4.3 User interface description

The following is a possible user interface for the AXMEDIS Editor, a Frame is used for each object opened. Inside the frame, on the left a tree view representing the object structure is presented (see part B for details on the Hierarchy View) and on the right the different views of the object are hosted (see part B for the different viewers/editors). The AXMEDIS Hierarchy can be shown also as it will be visible from the End User by selecting the specific Combo Box.



AXMEDIS Editor

The top-level menus are:

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

Edit	Description
Undo	to undo the last operation
Redo	to redo the last operation undone
Cut	cuts the element
Copy	copy an element
Paste	paste an element
Delete	removes an element without putting it in the clipboard
Content processing Plugin...	Shows the plugins that can be used on the selected element

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

Editors/Viewers	Description
Resource Editor & Viewer	Opens the resource viewer
Metadata Editor & Viewer	Opens the metadata editor & viewer
Annotation Editor & Viewer	Opens the annotation editor & viewer
Visual Editor & Viewer	Opens the visual view on the whole object
Behaviour Editor & Viewer	Opens the behaviour editor & viewer on the whole object
DRM Editor & Viewer	Opens the DRM editor & viewer
Workflow Editor & Viewer	Opens the workflow editor & viewer
Show all	Opens all views

Play	Description
Start	Starts playing the object
Pause	Pause object palying
Stop	Stop playing

Window	Description
...	contains the list of windows currently opened

Help	Description
Guide to AXMEDIS Editor...	Opens the guide to the AXMEDIS Editor
About AXMEDIS Editor...	Opens a dialog showing information on the AXMEDIS Editor

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

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

On the MPEG21 Hierarchy specific contextual menus are used:

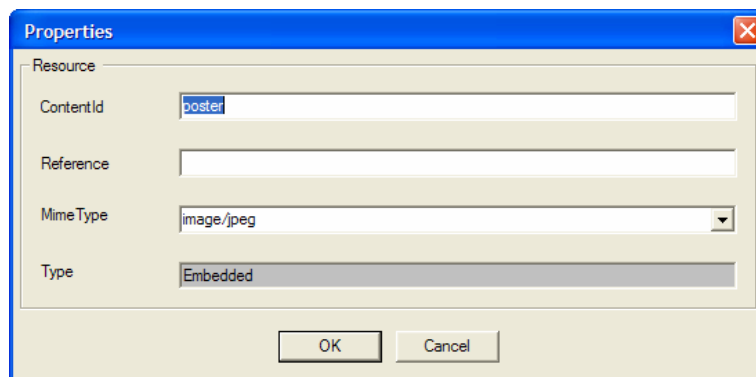
Open	open a default view for the element
Open with...	in case more choices are available let the user choose
Properties...	show properties of the element
Cut	cuts the element
Copy	copies the element
Paste	pastes an element after the selected one
Delete	removes the element without putting it in the clipboard

Move up	move the element up
Move down	move the element down
Insert	
Descriptor	inserts a new descriptor
Statement	inserts a new statement
Item	inserts a new item
Component	inserts a new component
Resource	inserts a new resource
Container	inserts a new container
Reference	inserts a new reference
...	
Plugin...	let the user select the operation to be performed on the element

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

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

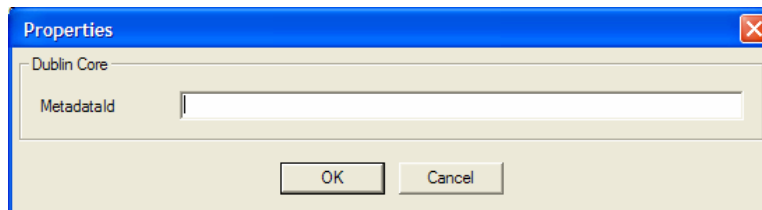
The following are the dialog to edit properties of a resource, a metadata and an AXMEDIS Object:



The 'Properties' dialog for a Resource has a title bar with a close button. It contains the following fields:

- ContentId:** A text field containing the value 'poster'.
- Reference:** An empty text field.
- MimeType:** A dropdown menu showing 'image/jpeg'.
- Type:** A dropdown menu showing 'Embedded'.

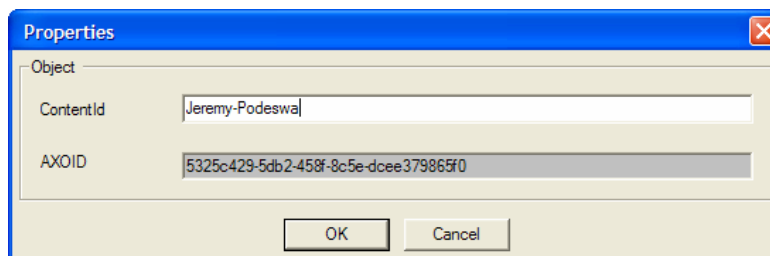
At the bottom are 'OK' and 'Cancel' buttons.



The 'Properties' dialog for Dublin Core Metadata has a title bar with a close button. It contains the following field:

- MetadataId:** An empty text field.

At the bottom are 'OK' and 'Cancel' buttons.

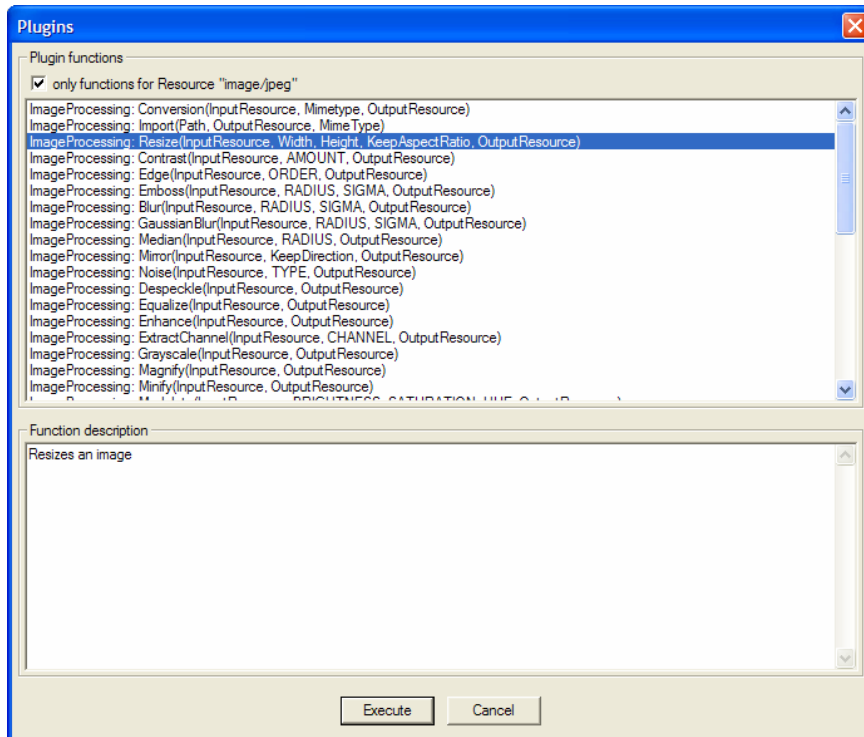


The 'Properties' dialog for an AXMEDIS Object has a title bar with a close button. It contains the following fields:

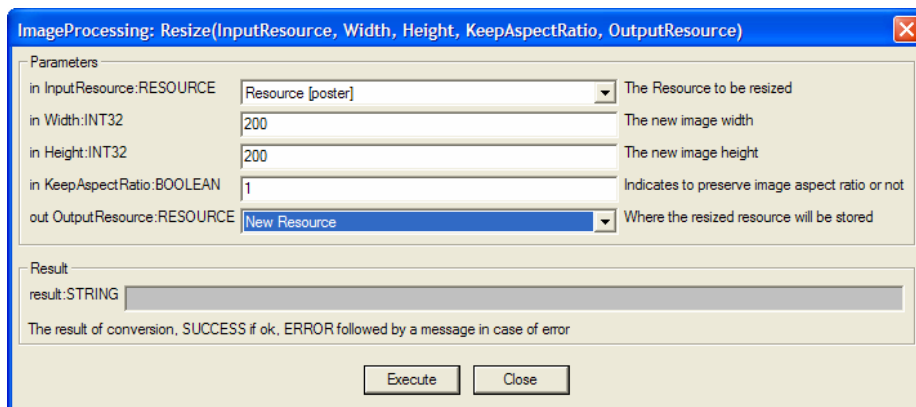
- ContentId:** A text field containing the value 'Jeremy-Podeswa'.
- AXOID:** A text field containing the value '5325c429-5db2-459f-8c5e-dcee379865f0'.

At the bottom are 'OK' and 'Cancel' buttons.

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



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



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

Title	Object Id	So...
The sin of Harold Diddlebock	00e02acf-21f2-4394-be9f-71dd3a801e53	AXDB
La mia brunetta preferita	028a06da-d56a-4a3b-9c09-d29b3d5fc133	AXDB
La palla numero 13	04ce32dd-5065-4653-99ef-d32d5c78536a	AXDB
Il Navigatore	06326ae3-b807-4105-9a27-ef9bcccc2f13e	AXDB
Il cameraman	06fb8775-1f3c-45a3-bf7b-ac66e43ea02a	AXDB
Follie di Jazz (Second Chorus)	09d29cd8-4cb5-4b9f-b67f-26cd45825138	AXDB
The perils of Pauline	16d7ce20-f45a-4a18-b217-167044983e86	AXDB
The perils of Pauline	1a38d5e6-b86a-4a78-abc8-a35072e1a9b5	AXDB
McLintock!	1dcca9f3-ab67-4a57-b730-05d24b0d53c2	AXDB
L'uomo in più	24a45cb9-1943-4816-93cc-12c5f05e472a	AXDB
McLintock!	25fe87f9-7764-42e7-a6b7-e3e27589ca75	AXDB
Il tesoro del santo	2b1ced87-a3bf-4178-9374-e6a529af2d8b	AXDB
Love Laughs at Andy Hardy	2b3a72f0-c37e-49a6-bcf1-a41fc32b7231	AXDB
La mia brunetta preferita	2db6f9a6-707e-4e4c-ad73-42952803a0eb	AXDB
Love Laughs at Andy Hardy	2dd476db-b480-40ea-881d-47121db51d92	AXDB
Il Navigatore	2df62823-ae4e-4767-b301-af6c3d04f4ae	AXDB
La palla numero 13	39d6c07d-3d6b-41b9-9392-8b49682f51b4	AXDB
La mia brunetta preferita	3c6e856b-5bb1-41f0-9f22-75d099c7f41c	AXDB
Il Navigatore	3f6de7fb-f5d7-48a4-8266-13b071383784	AXDB

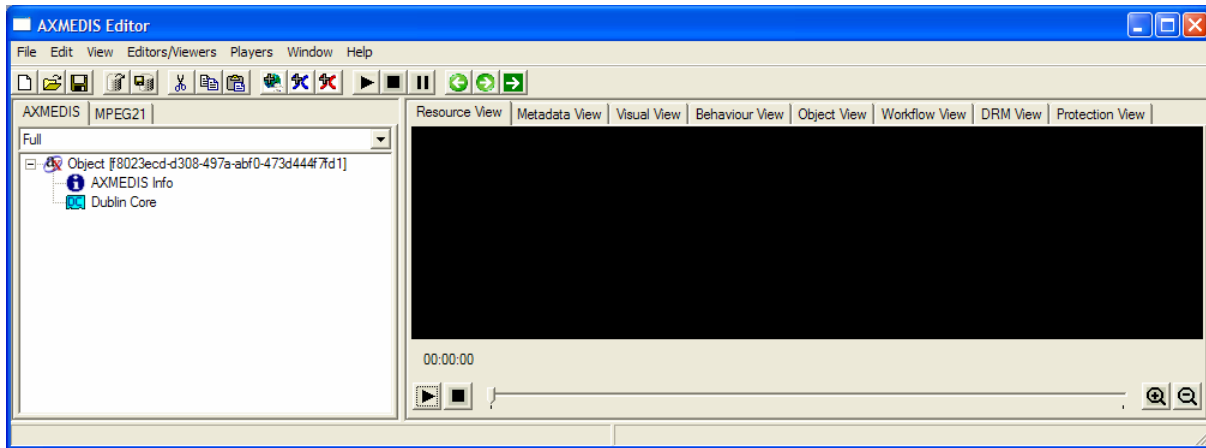
4.4 Technical and Installation information


References to other major components needed	<ul style="list-style-type: none"> AXMEDIS Object Manger AXMEDIS Plugin Manager AXMEDIS Content Processing Plugins
Problems not solved	
Configuration and execution context	

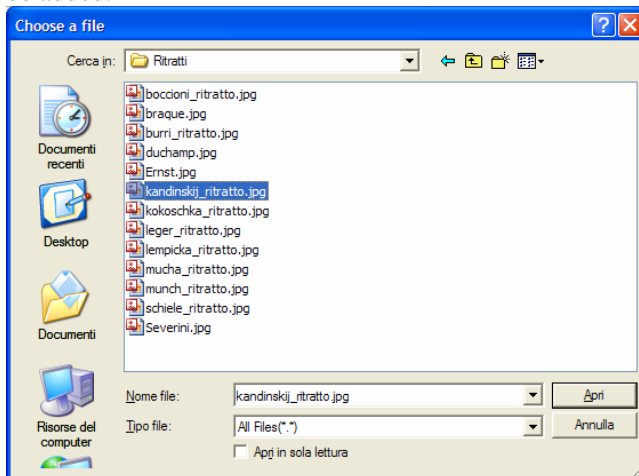
4.5 Draft User Manual

4.5.1 Create a new AXMEDIS Object

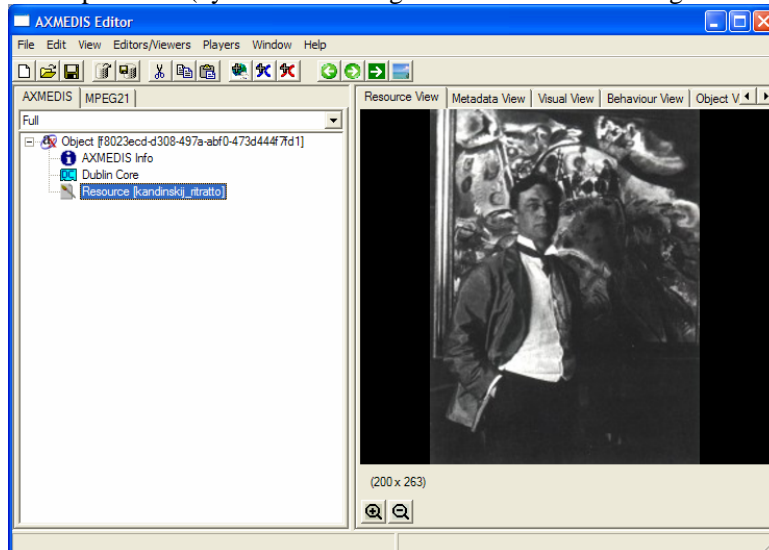
To create a new AXMEDIS object select **File/New** from the Menu or use the  button on the toolbar



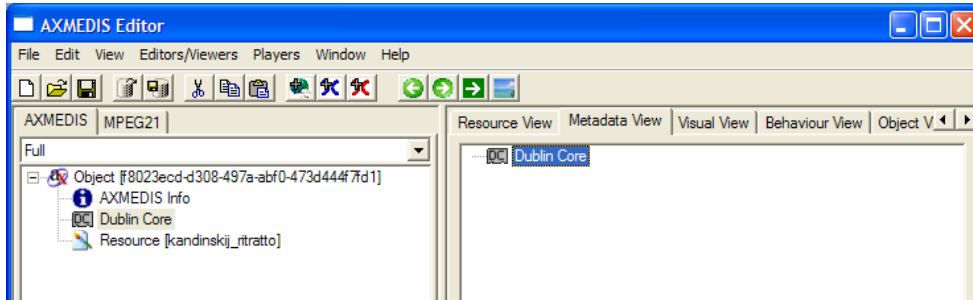
To add a resource from a file on the local hard disk select the  button from the toolbar and select the file to be added:



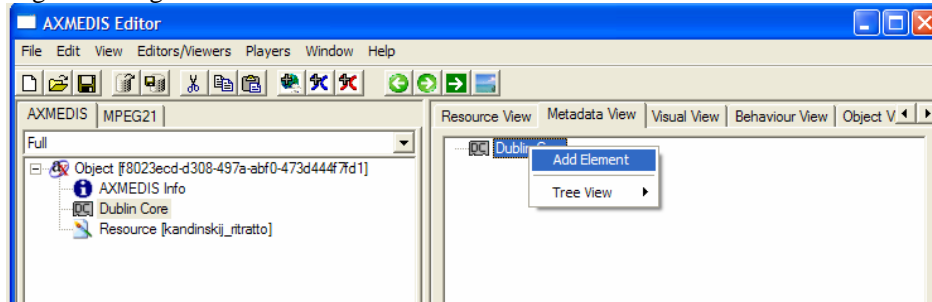
Which produces (by double clicking on the resource the ImageViewer is opened):



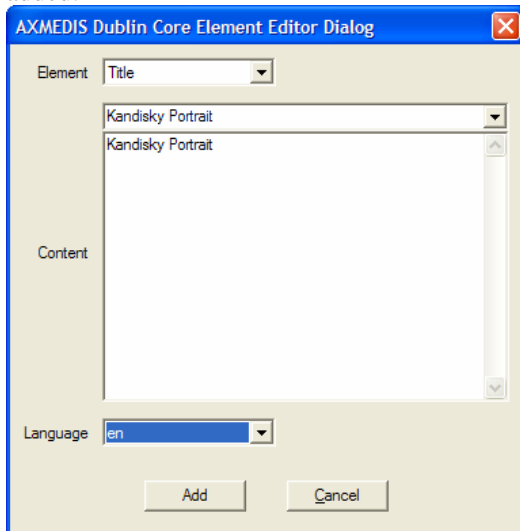
Double clicking on the Dublin Core metadata the metadata editor is opened:



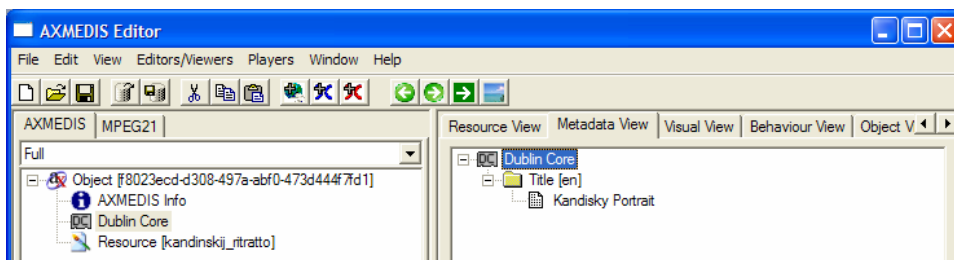
Right clicking on the Dublin core a new element can be added



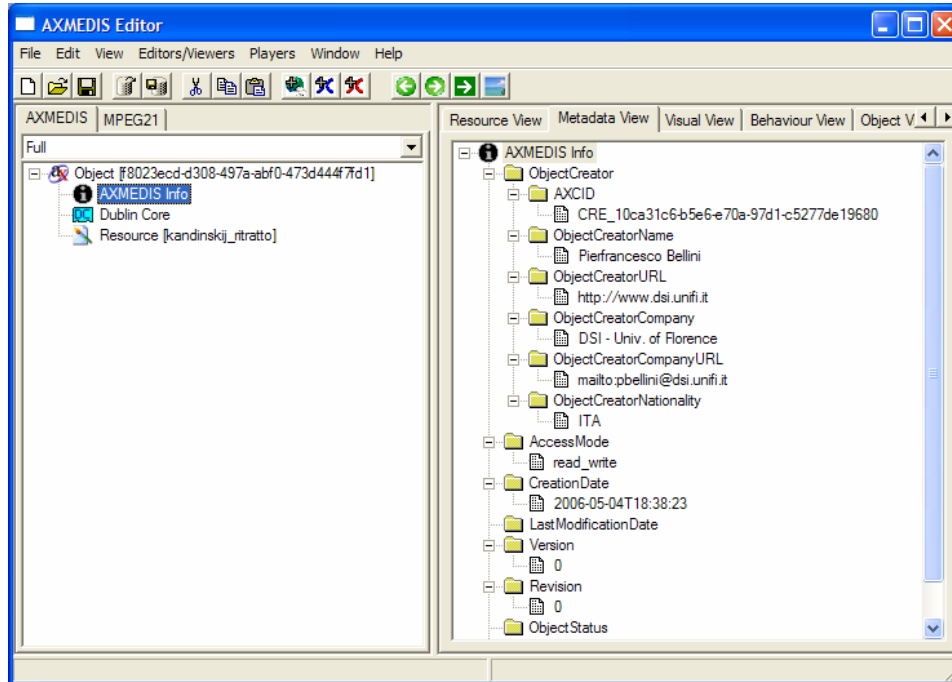
using the following dialog, selecting the Title element and entering the value and the language the title is added:




which produces:




Double clicking on the AXMEDIS Info the Metadata Viewer shows the AXMEDIS specific information associated with the object:

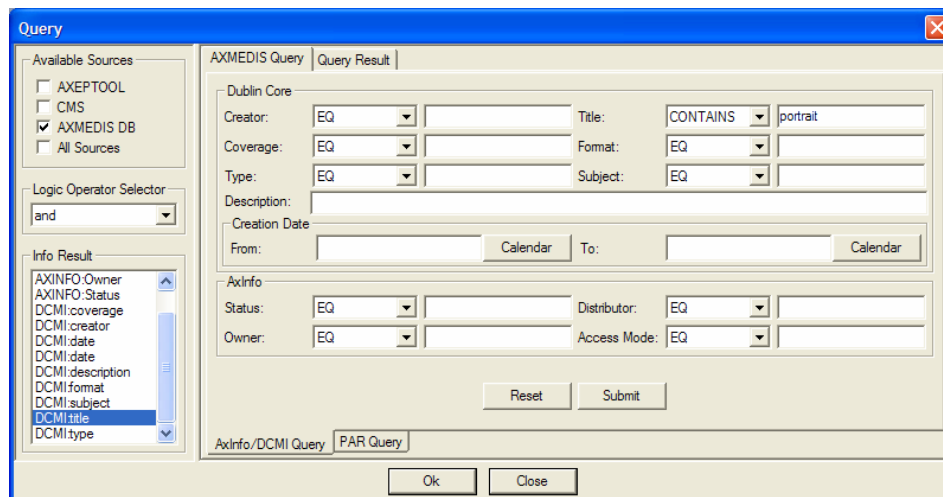


The Object Creator information is automatically added getting information from the configuration (**File/Configuration...** from the menu)

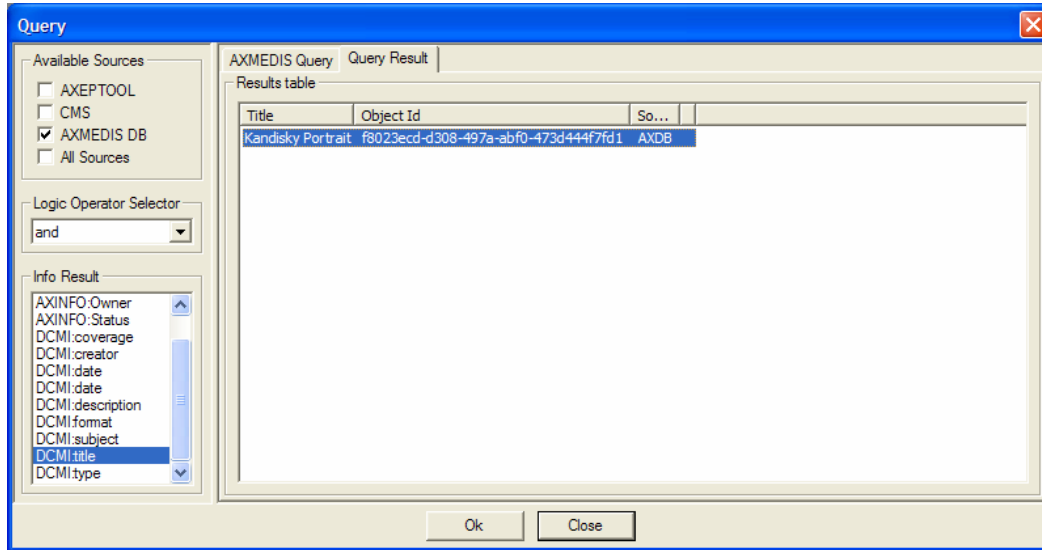
The object can be uploaded on the AXMEDIS Database using the  button on the toolbar or **File/Upload into Database...** from the menu or the object can be saved on the local hard disk using **File/Save** or **File/Save As...** from the menu.

4.5.2 Modifying an AXMEDIS Object stored on Database

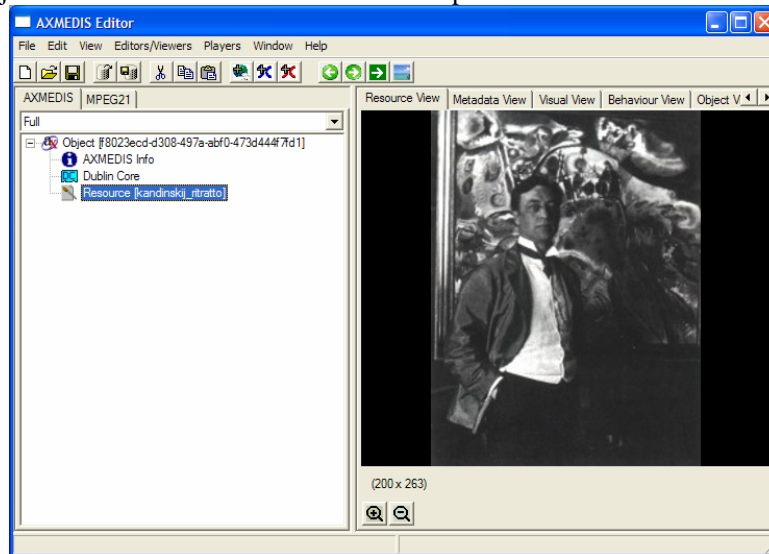
To search for the AXMEDIS object on the Database select the  button on the toolbar or using **File/Open from Database...** in the menu, the query dialog is opened and it is searched for an object with the title containing "portrait":



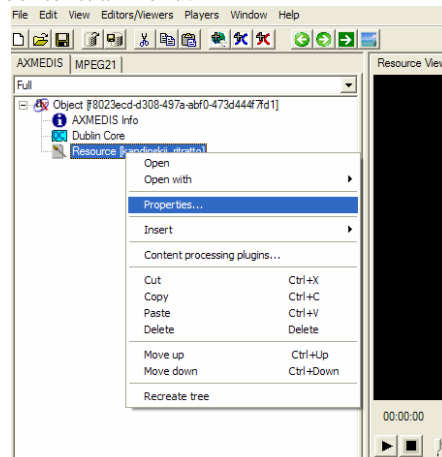
Pressing the Submit button the query is sent to the database and the results are shown:



Pressing Ok the object is recovered from the database and opened:

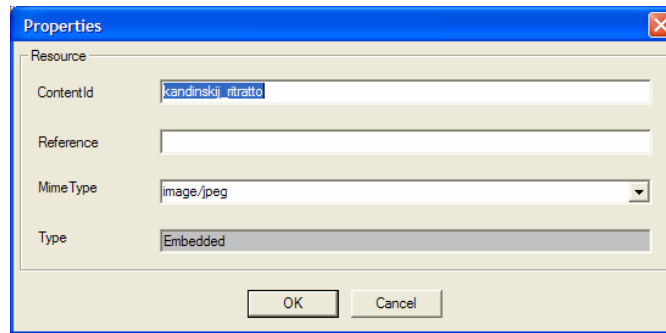


The Properties of the Resource (and for any other element) can be edited right clicking on the element and selecting **Properties...** from the contextual menu:

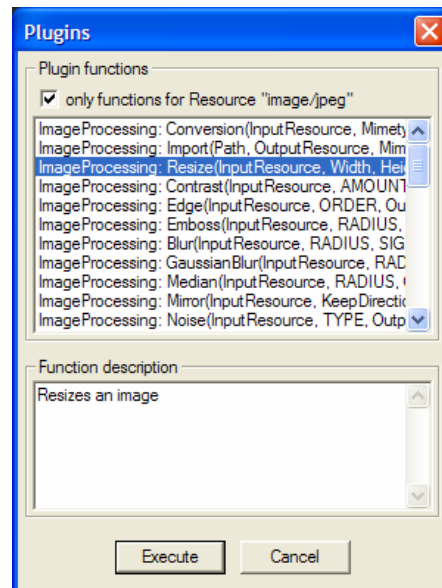


and the following dialog is opened and the properties can be modified:

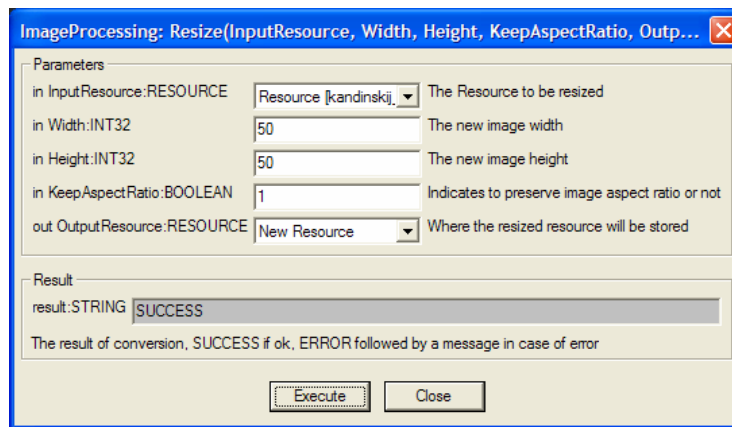
AXMEDIS Project



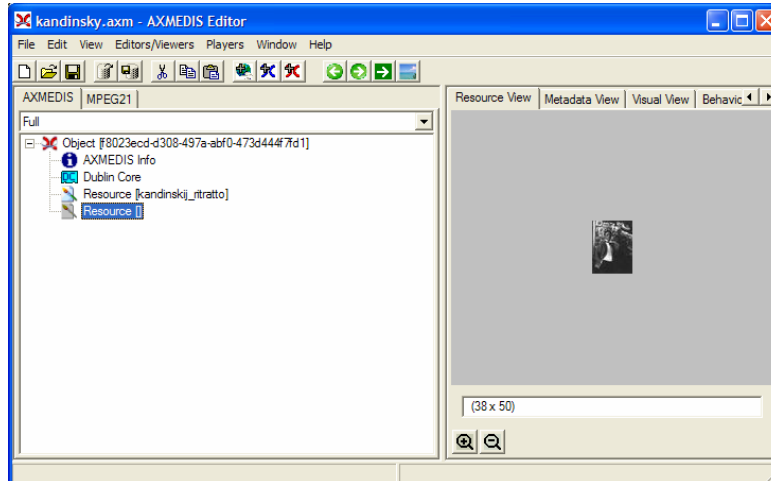
On a Resource the contextual menu enables to use content processing plugins, selecting the **Content processing plugins...** menu option the plugin function list is provided with the list of applicable plugins functions (based on resource mimetype):



Selecting a plugin function and pressing the **Execute** button a dialog is presented allowing to provide the arguments for the function, clicking on the **Execute** button the plugin function is executed:



The result is the following:



4.6 Examples of usage

see Draft User Manual

4.7 Integration and compilation issues

None

4.8 Configuration Parameters

Config parameter	Possible values

4.9 Errors reported and that may occur

Error code	Description and rationales

5 Module - Hierarchy Editor and Viewer (DSI)

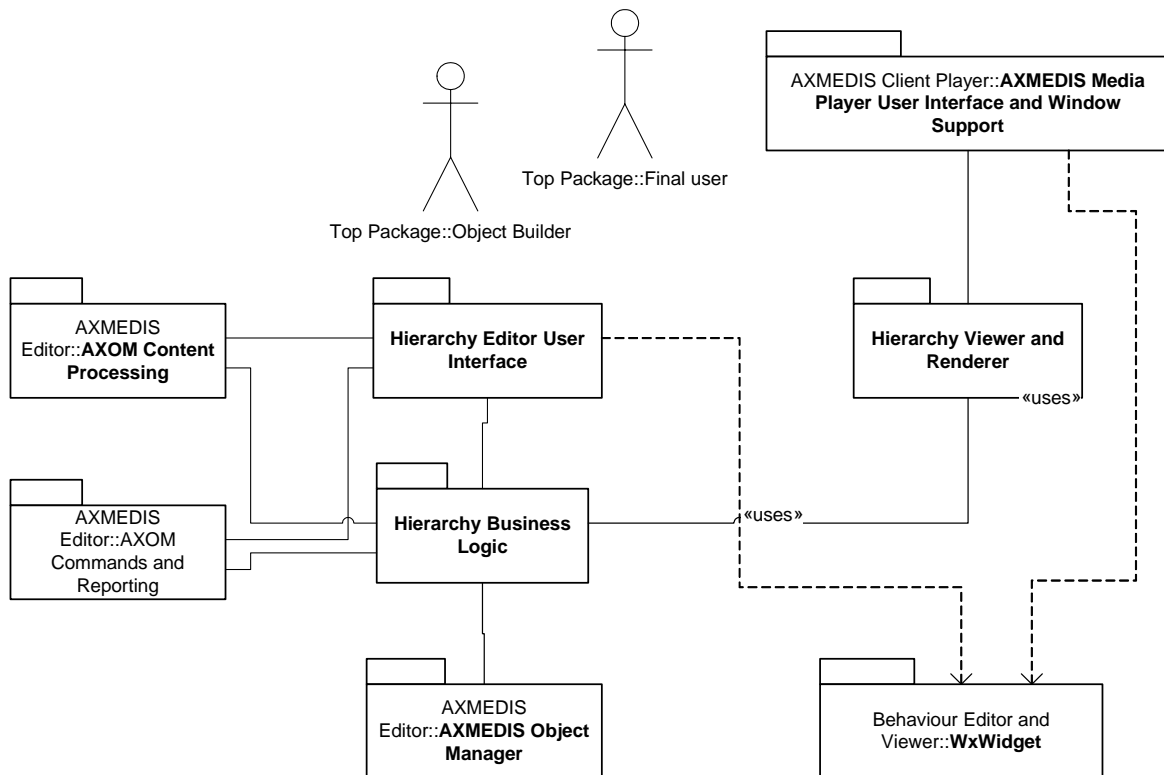
Module/Tool Profile	
Hierarchy Editor and Viewer	
Responsible Name	Bellini
Responsible Partner	DSI
Status (proposed/approved)	

Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)	Library	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported		
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/axeditor/hierarchy_view	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/repos/Applications/axeditor/bin	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	yes	
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved	-- --	
Major pending requirements	Synchronization of MPEG21 view and AXMEDIS view	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		

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

5.1 General Description of the Module

Hierarchy Editor and Viewer



Hierarchy View should show hierarchical relationship among object subparts. Because of linearity of MPEG-21/AXMEDIS relationship among components (i.e. each component has one, and only one, father component), tree-like view (similar to the one of explorer) is the most suitable solution. Such view should

AXMEDIS Project

permit cut and paste operation and, moreover, it should permit specific operations (through contextual-menu usage) on showed elements.

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

The AXMEDIS Hierarchy Editor and Viewer allows to:

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

The MPEG-21 Hierarchy Editor and Viewer allows to:

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

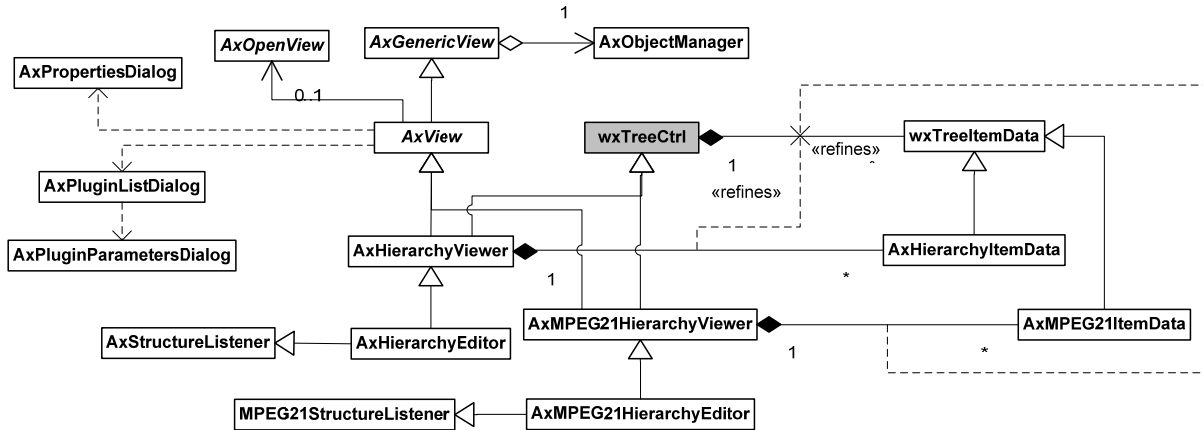
The business logic under the hierarchy interface will basically cover four action on the hierarchy element

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

All the commands will be mapped on several command instances:

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

5.2 Module Design in terms of Classes



5.3 User interface description

The following is an example of AXMEDIS Hierarchy:

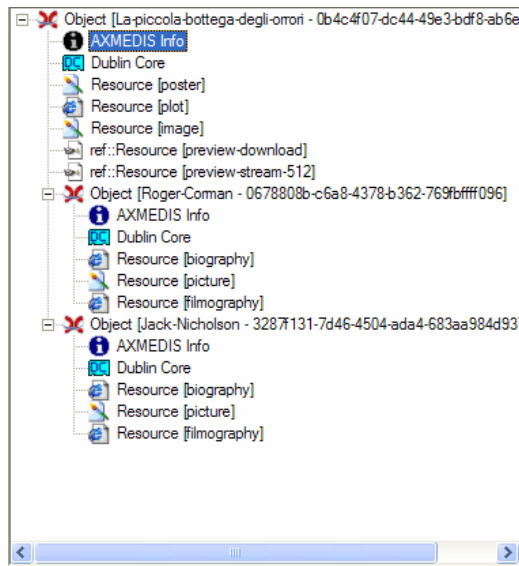


Figure 1 - Example of AXMEDIS hierarchy

The following is an example of MPEG-21 Hierarchy:

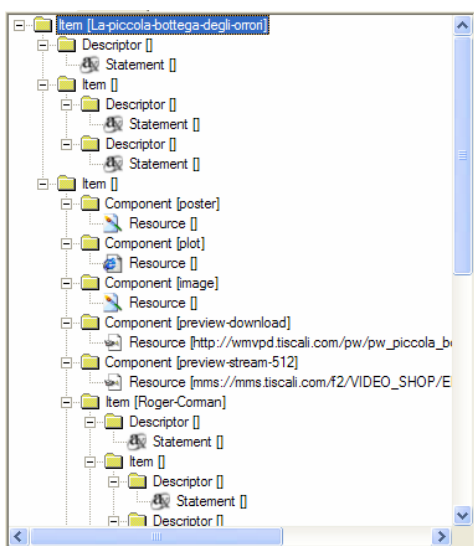


Figure 2 - Example of MPEG-21 hierarchy

5.4 Technical and Installation information

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

5.5 Draft User Manual

5.6 Examples of usage

see Draft User Manual

5.7 Integration and compilation issues

None

5.8 Configuration Parameters

Config parameter	Possible values

5.9 Errors reported and that may occur

Error code	Description and rationales

6 Module DRM Editor and Viewer (FUPF)

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

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

6.1 General Description of the Module

6.1.1 Form of the DRM Editor & Viewer

- **Libraries.** DRM Editor & Viewer are two different C++ libraries. They present a tree view where a License can be viewed and eventually edited (this last feature, only in the Editor). The Editor is a superset of the Viewer. Additionally, a panel on the right side of the window displays the information in a different form.
- **Executables.** They provide the stand-alone functionality to demonstrate their use.

6.1.2 Architecture

The DRM Editor and Viewer libraries are based in a important number of libraries, whose relationship are shown here.

6.1.3 Functionalities

- (E,V). It permits to visualize the license, in a browsable tree structure.
- (E,V). The tree structure can be easily embedded in a wxWindows application.
- (E,V). Elements of the license are represented in the leaves of the tree, in a user readable way.
- (E,V). The tree to be visualized can be expanded or collapsed at once or individually.
- (E,V). Licenses can be open from a XML file.
- (E,V) A panel is displayed in the right side of the tree, showing partially or totally the information from the tree in an alternative format
- (E,V) The right panel can be hidden, reverting the screen to the single tree.
- (E,V) An icon in the right bottom corner shows whether there is or there is not connection with the PMS Server available. The refresh rate of this icon has been set in 10 seconds, so possible reconections or disconnections take a few seconds in being shown.
- (E,V) A button bar contains the different possible actions for the license. This button bar is not available in the integrated version of the editor, but alternative access to the functionality is also offered.
- (E). A context menu can be displayed in order to show edit options.
- (E). Each of the leaves can be edited. (only valid values may be accepted).
- (E). Elements can be deleted. (with the restriction to those elements who cannot dissappear)
- (E). New elements can be added.
- (E). The license can be stored in a xml file and in the server.
- (E). The information in the right hand side panel can be changed. Upon the presh of a button, new introduced information is transferred into the current license.
- (E). Creation of PAR from internal PAR associated to an AXMEDIS object.
- (E). Creation of new license from PAR associated to an AXMEDIS object.

E: Available only in the Editor

V: Available only in the Viewer

6.1.4 DRM Editor Business Logic

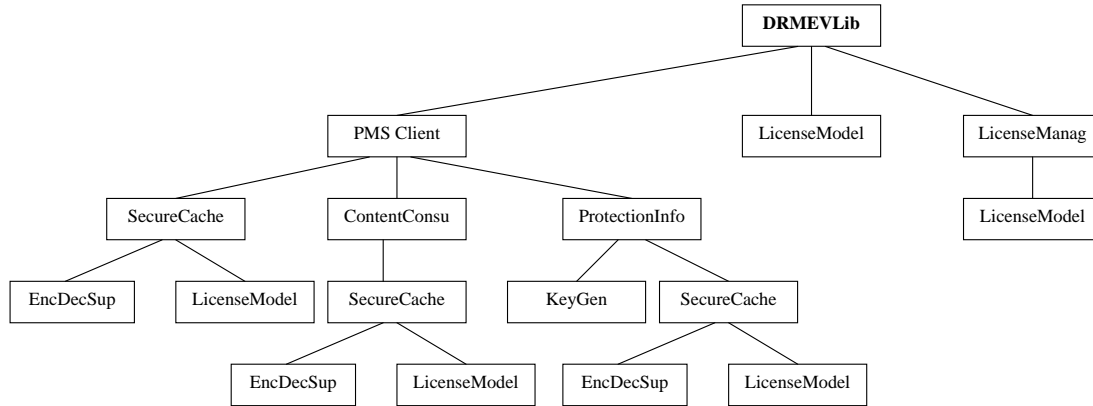
The business logic under the DRM viewer interface will cover the actions we can do over three kinds of DRM information:

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

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

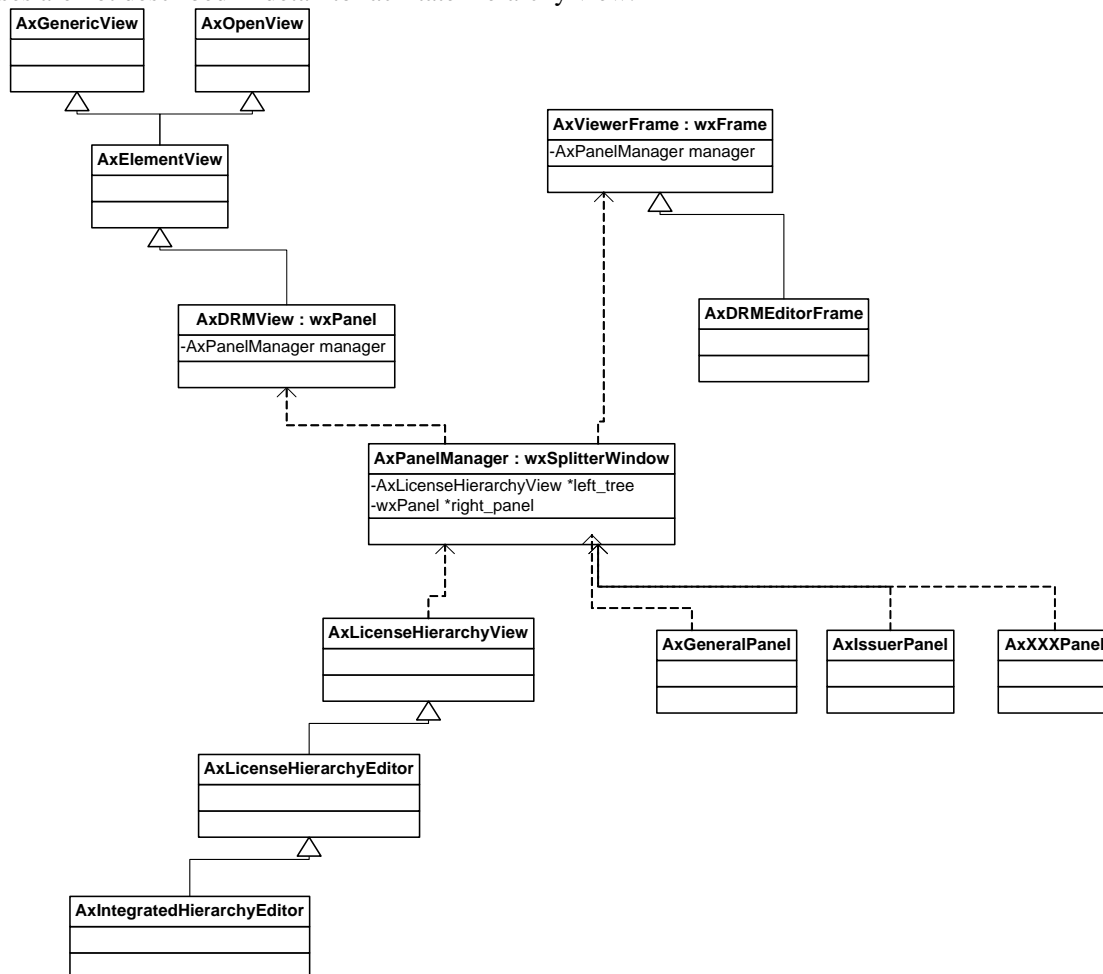
6.2 Module Design in terms of Classes

Dependencies of the DRM Editor and Viewer library.



Class diagram of DRM Editor and Viewer

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



6.3 User interface description

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

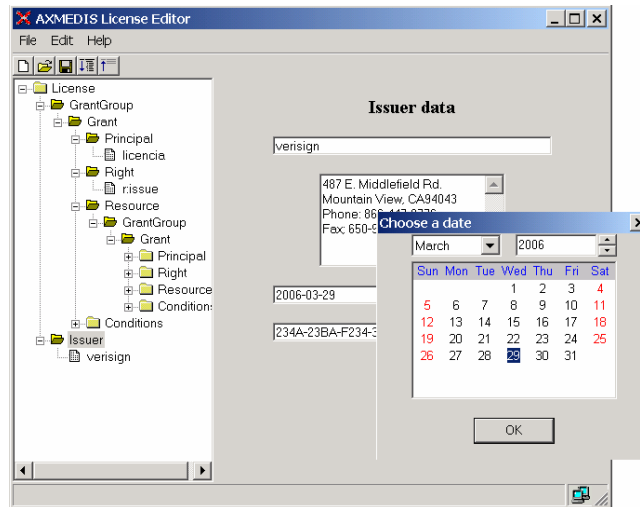


Figure. DRM Editor and Viewer user interface

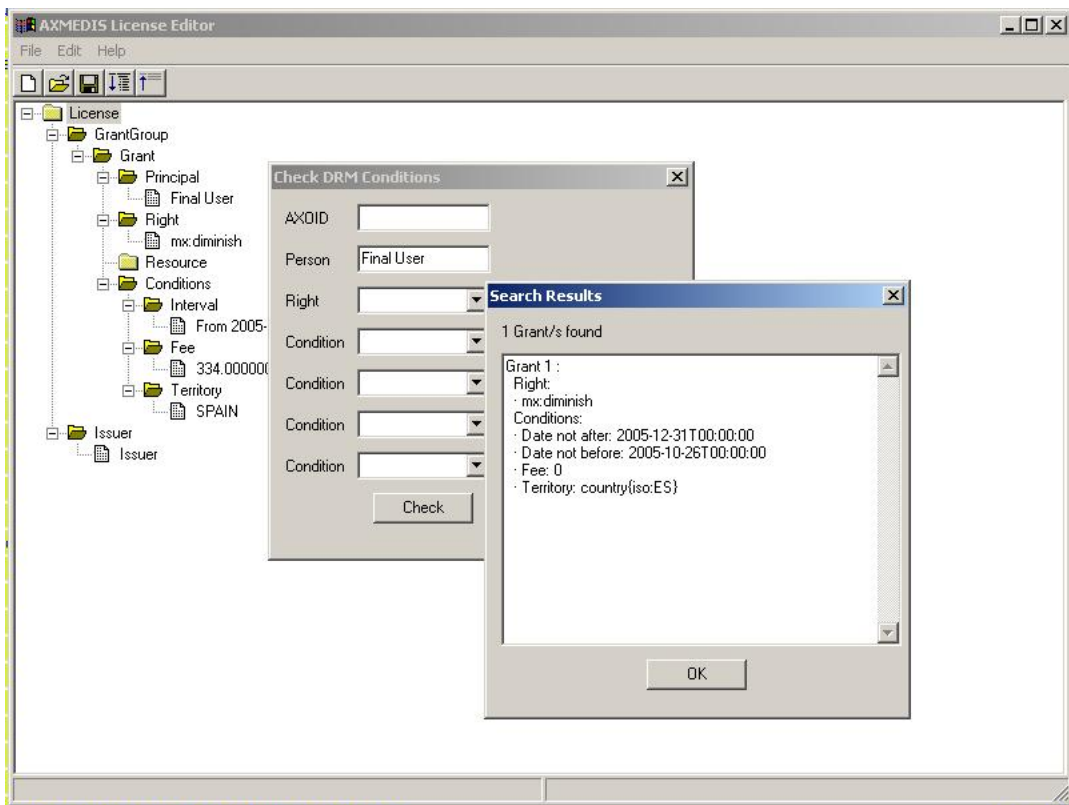


Figure. Check if license accomplishes conditions

6.3.1 DRM Viewer

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

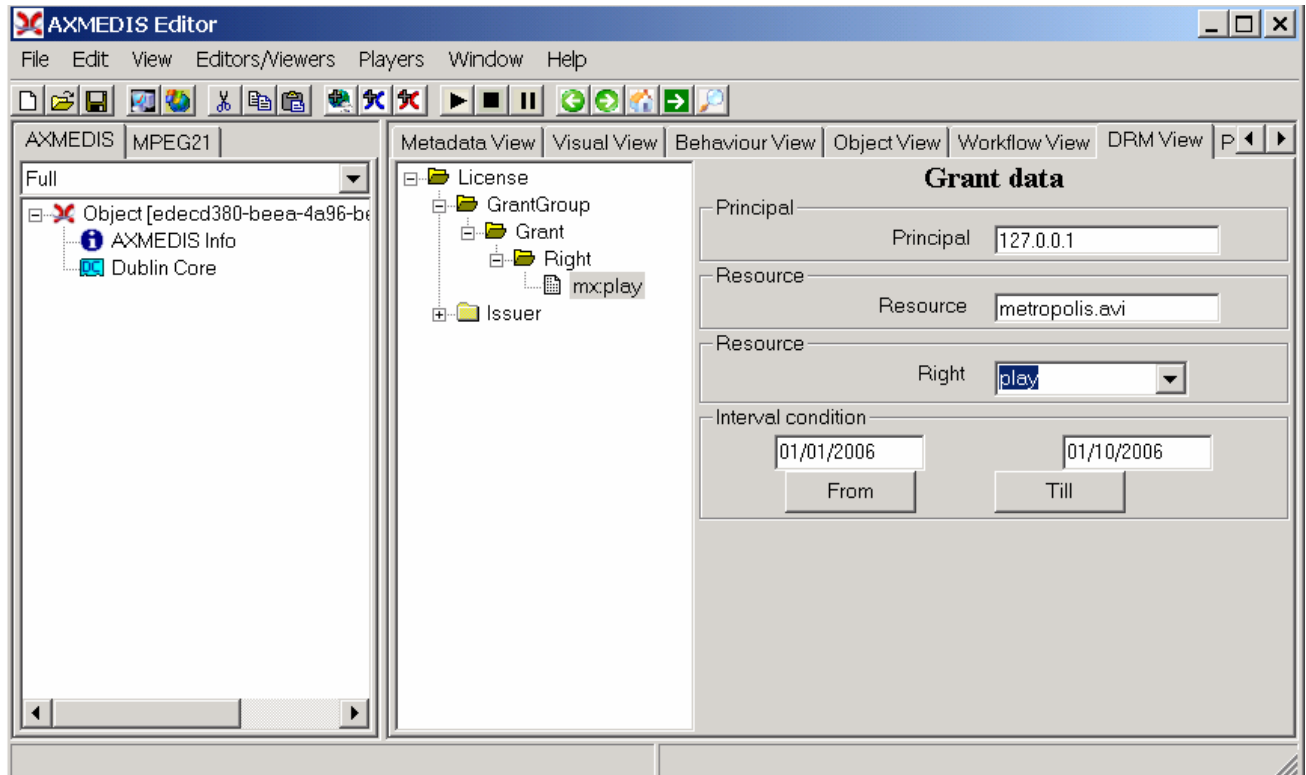


Figure: It shows the DRM Editor and Viewer integrated with the AXMEDIS Editor. In the viewer version, nothing changes excepting that fields are non-editable.

6.4 Technical and Installation information

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

7 Module - Protection Editor and Viewer (FHGIGD)

Module/Tool Profile	
Protection Editor and Viewer	
Responsible Name	Martin Schmucker
Responsible Partner	FHGIGD
Status (proposed/approved)	Approved
Implemented/not implemented	Implemented
Status of the implementation	First prototype
Executable or Library/module (Support)	Library
Single Thread or	Single thread

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

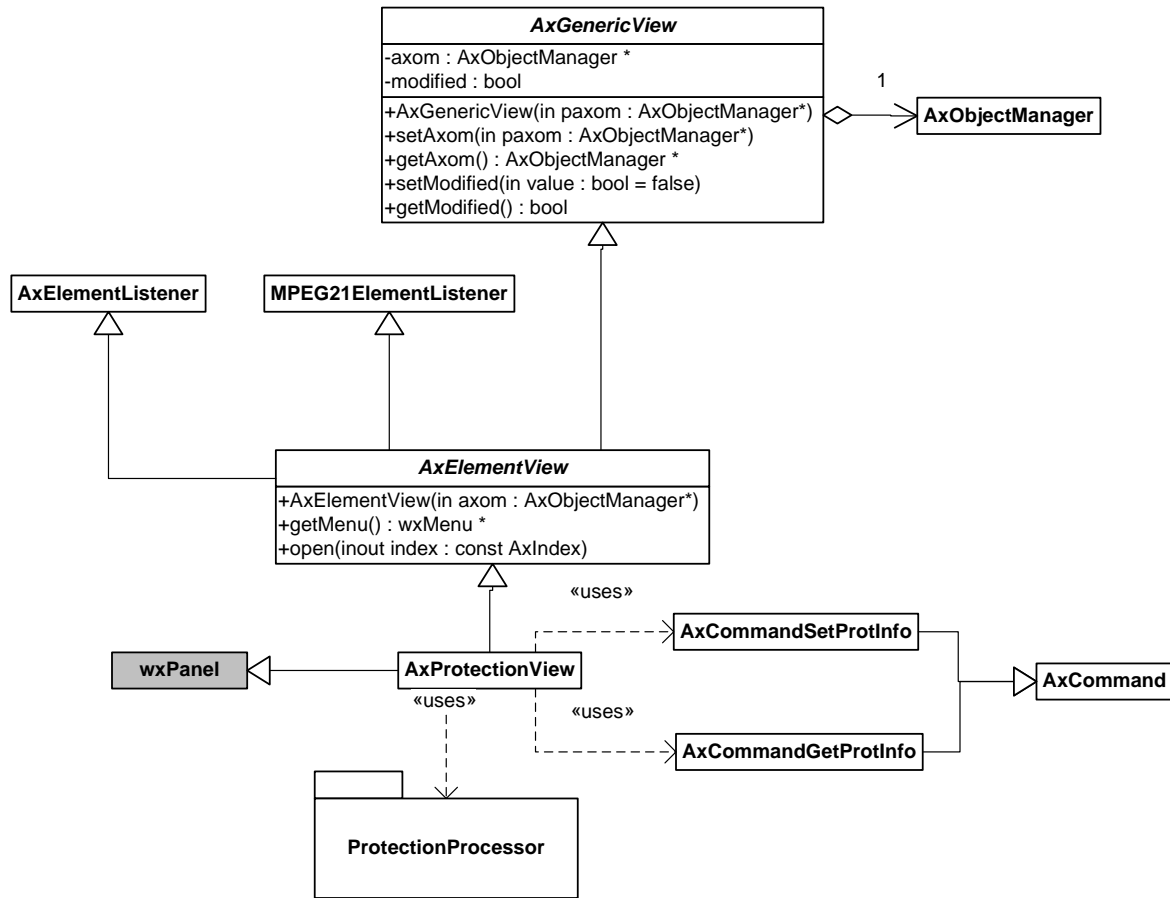
	etc.	platform, etc.
yes	C++	
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
yes	WxWidgets	wxWindows licence (http://www.wxwidgets.org/newlicen.htm)

7.1 General Description of the Module

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

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

7.2 Module Design in terms of Classes



7.3 User interface description

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

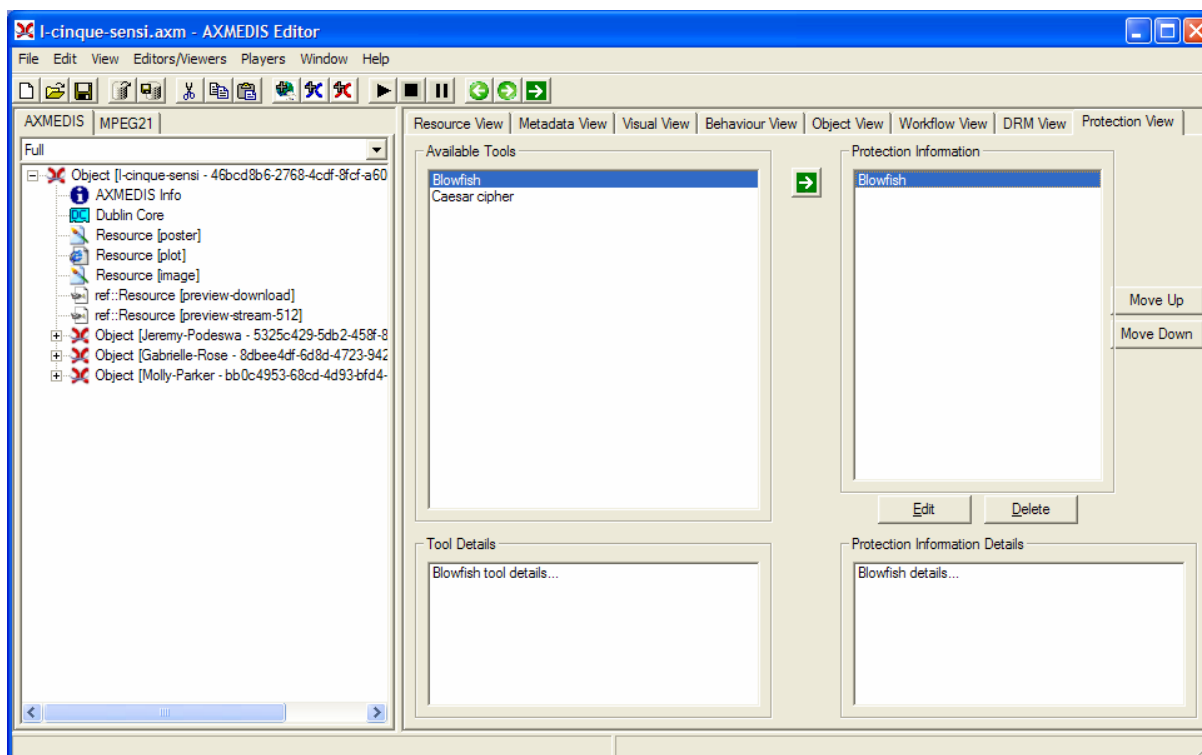


Figure. Protection Information Editor and Viewer user interface

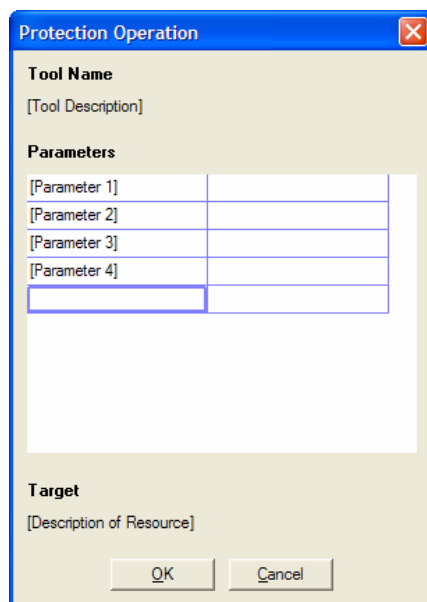


Figure: Parameter setting for a specific Protection Operation

7.4 Technical and Installation information

References to other major components needed	Protection Processor
Problems not solved	

Configuration and execution context	The AXMEDIS Protection Information Editor and Viewer is executed within the AXMEDIS Editor.
-------------------------------------	---

7.5 Draft User Manual and Example of Usage

The draft user manual and the examples of usage are currently under development.

7.6 Integration and compilation issues

None

7.7 Configuration Parameters

Config parameter	Possible values

7.8 Errors reported and that may occur

Error code	Description and rationales

8 Module Visual Editor and Viewer (EPFL)

Module/Tool Profile	
Visual Editor and Viewer	
Responsible Name	Beilu Shao
Responsible Partner	EPFL
Status (proposed/approved)	proposed
Implemented/not implemented	Implemented
Status of the implementation	80%
Executable or Library/module (Support)	Library
Single Thread or Multithread	Multithread
Language of Development	C++
Platforms supported	Windows and Linux
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/visual_editor_viewer
Reference to the AXFW	

location of the demonstrator executable tool for internal download		
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any	N/A	
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	Yes	
Usage of the AXMEDIS Error Manager (yes/no)		
Major Problems not solved	--Loading media resources from the panel instead of the disk --Navigation and hyperlinking with multiple SMIL Scenes	
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

8.1.1.1 Hierarchy Authoring of SMIL with Hierarchy Editor

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

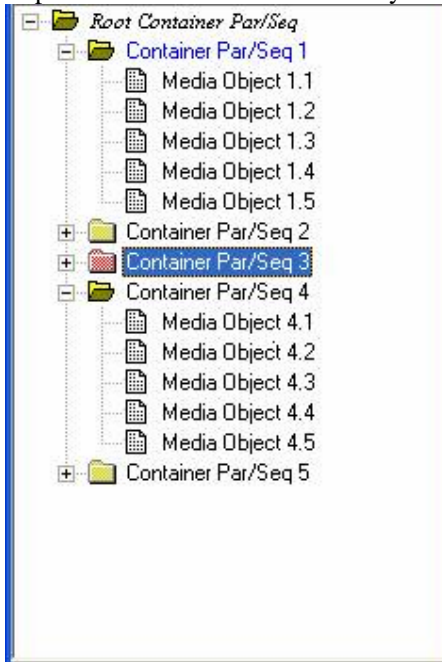
```
<body>
  <par>
    <seq>
      
      <audio id="Welcome-US" region="LogoRegion"
src="LITEdataCE/Welcome-US.mp3" dur="21.4s"/>
    </seq>
    <audio id="lamp-audio" region="content-audio"
src="LITEdataCE/lamp1.mp3"/>
  </par>
</body>
```

Or

```
<body>
  <par>
    <par dur="30s">
      
      <audio id="Welcome-US" region="LogoRegion"
src="LITEdataCE/Welcome-US.mp3" dur="21.4s"/>
    </par>
    <seq>
      
      <audio id="Welcome" region="LogoRegion" src="LITEdataCE/Welcome-
US.mp3" dur="2s"/>
    </seq>
  </par>
</body>
```

The hierarchy editing of SMIL components is the first step before the spatial visual editing and temporal synchronization. To create an element inside SMIL object, we have to determine what kind of structure of

these elements. To be more specific, we have to define it is container (par/seq) or object. The following is a snapshot of the GUI of the hierarchy structure.



8.1.1.2 Load Media Resources from AXMEDIS Object with SMIL Hierarchy Editor.

SMIL component is created by including media resources from the left panel of the AXMEDIS object. The working process is described as follows: with one internal media resource selected, the SMILHierarchy business logic will extract and return the reference or the ID from the AXMEDIS Object to the SMIL component.

8.1.1.3 Spatial Visual Authoring of SMIL with Visual Editor

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?><DIDL
xmlns:xi="http://www.w3.org/2001/XInclude" >
...
```

```

...
<Item>
    <Component id="Colline-azzurre">
        <Resource mimeType="image/jpeg"
encoding="base64">...</Resource>
    </Component>
    <Component id="smilcomponent">
        <Resource mimeType="application/smil"
encoding="base64">...</Resource>
    </Component>
</Item>
</Item>
</DIDL>

```

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

Below I will explain the list of SMIL XML elements and attributes that will allow the Visual Business Logic to organize and store the layout of the presentation. The following is the basic skeleton of the SMIL presentation that we will have to support:

```

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

```

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

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

8.1.1.4 Loading Media Resources from SMIL Hierarchy Structure with Visual Editor

After creating the SMIL hierarchy structure with the SMIL Hierarchy Editor in section 8.1.1 and 8.1.2 and the spatial layout with Visual editor in section 8.1.3, we have decided what is contained in each rectangular shape.

It proceeds by including media resources from the SMIL hierarchy structure to each visual rectangular. The working process is described as follows: with one internal media resource selected, the visual business logic will extract and return the reference or the ID to the SMIL component. The final result of the SMIL component is as follows:

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

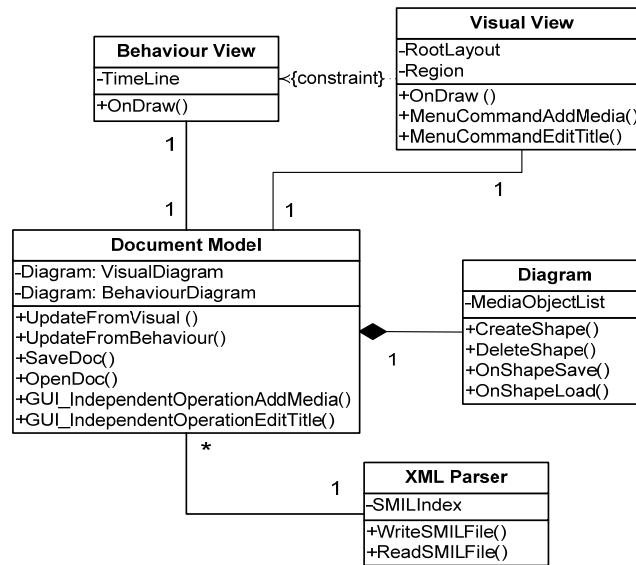
  <head>
    <layout>
      <root-layout background-color="red" height="600" id="rootlayout"
title="axmedis" width="800"/>
      <region height="143" id="RegionName1" left="105" top="124" width="104"/>
      <region height="145" id="RegionName2" left="295" top="125" width="235"/>
      <region height="147"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

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

  </smil>
```

Reference N is referred to the Nth media resource inside the AXMEDIS Object. After the creation this SMIL component can be compressed and saved as other typical media components inside the AXMEDIS Object.

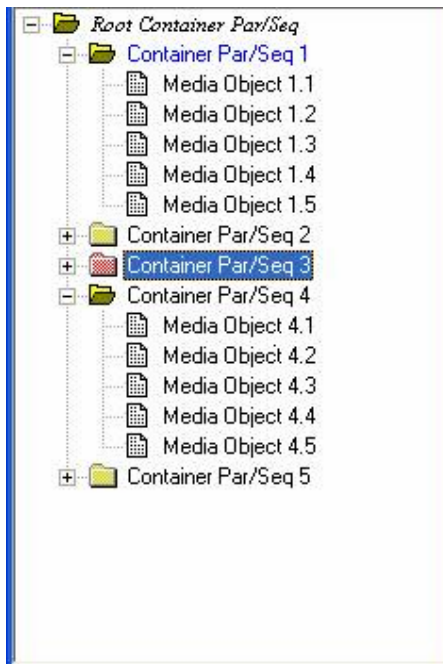
8.2 Module Design in terms of Classes



8.3 User interface description

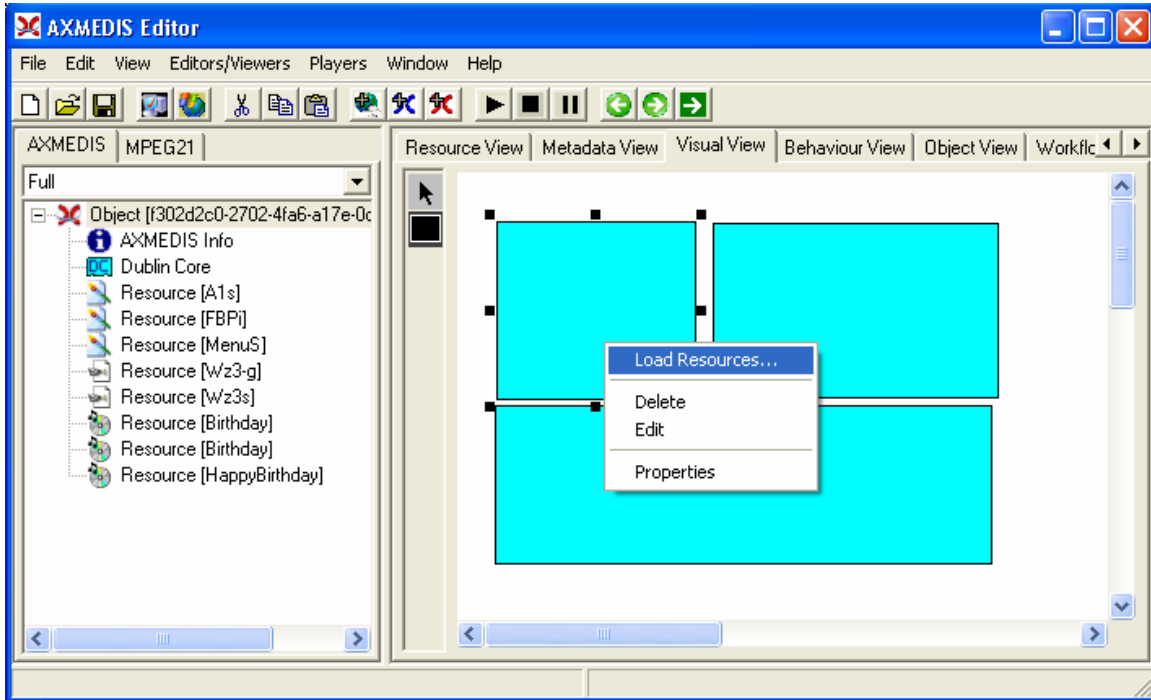
SMIL Hierarchy Editor and Viewer:

Inside the visual and behaviour editor, there is panel for the SMIL Hierarchy Editor and Viewer.



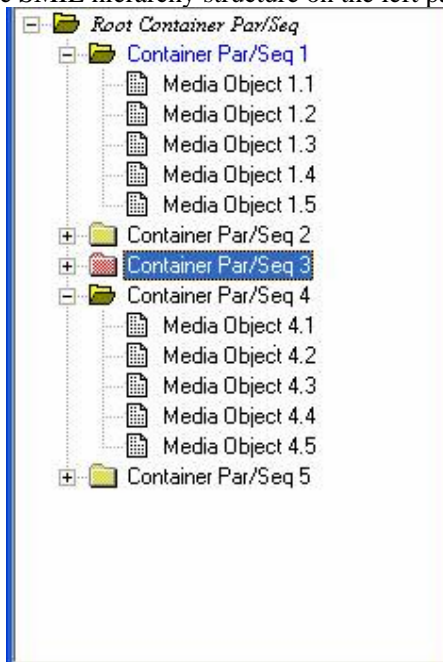
The Visual Editor GUI will allow the user to draw rectangles on the screen representing the regions where media objects will be rendered. To draw a new rectangle it will suffice to click-and-hold a mouse button, drag the mouse, and release the mouse button; this will define the upper left and lower right corners of the rectangle. Every rectangle will represent the region where a media object will be rendered. Every rectangle will be associated with a “region” element of the SMIL document managed by the Visual Business Logic. The user will be able to draw, resize, move, copy, paste, and delete rectangles. Every of this operations will be notified to the Visual Business Logic that in return will update the SMIL document. In this way, the SMIL document will be a faithful representation of what the rectangles the user will place on the screen. By right

clicking on a rectangle the user will access a context menu (see picture below) to add media resources from the list of on the left panel, change the rectangle properties. For instance if the user changes the “width” attribute, the rectangle will be resized to accommodate to the new width; this “width” attribute will also be updated in the SMIL document.



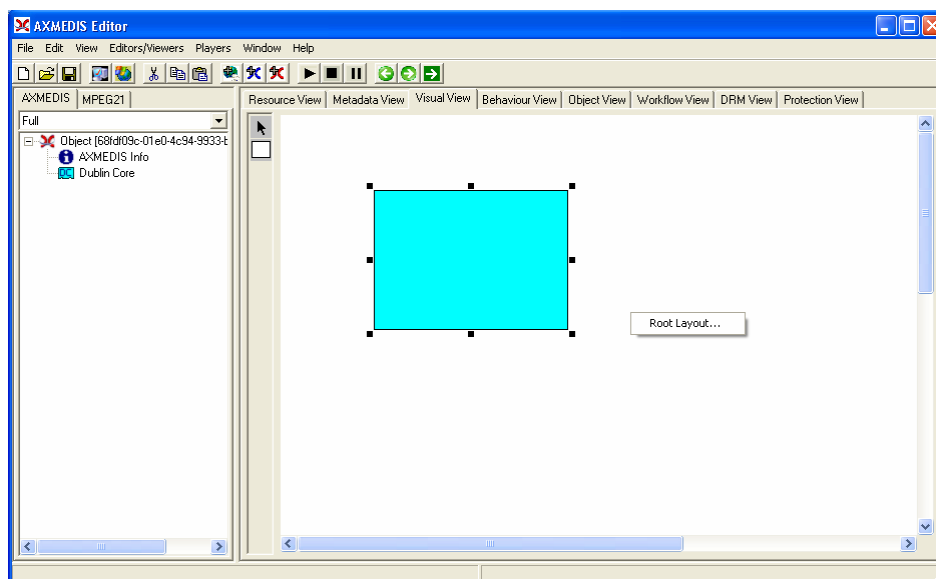
(The above visual editor is not integrated with the SMIL Hierarchy Editor and Viewer)

By right clicking the rectangular on the canvas, the user will have a context manual with the options “Loading Resources...”. With this option selected, the user will have a menu list of the media resources from the SMIL hierarchy structure on the left panel for selection as follows:

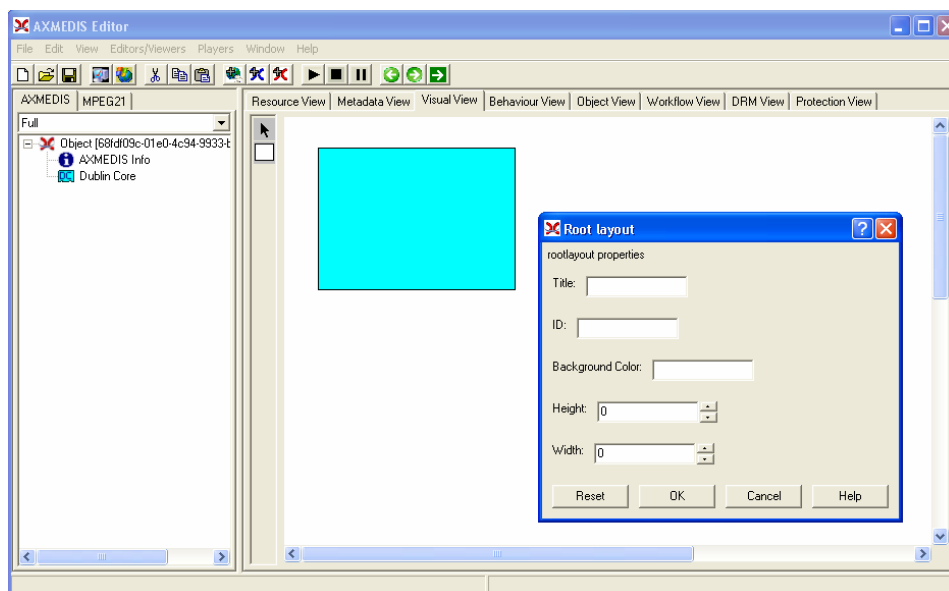


With one media resource selected, the visual business logic will extract its ID from the SMIL hierarchy structure, return the reference or ID of this internal resource and store it inside the SMIL component.

The user right clicks on an area where there no rectangulars; another context menu will pop up. This context menu will contain the “root-layout” attributes such as title, id, width and height, etc. These SMIL attributes control the display area of the presentation.



(The above visual editor is not integrated with the SMIL Hierarchy Editor and Viewer)



(The above visual editor is not integrated with the SMIL Hierarchy Editor and Viewer)

8.4 Technical and Installation information

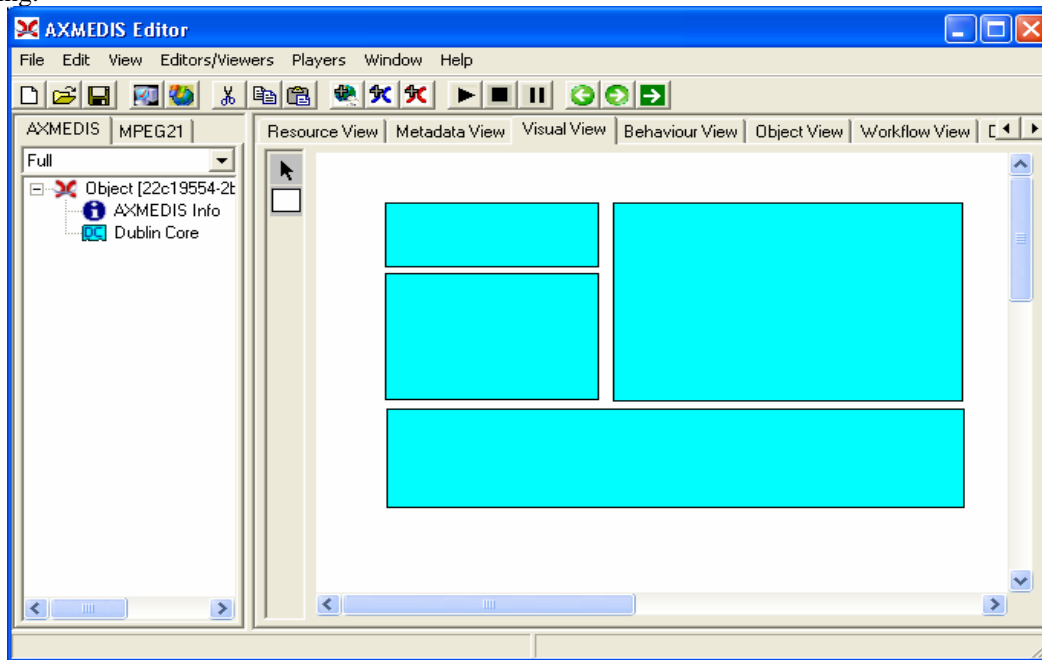
References to other major components needed	<ul style="list-style-type: none"> • Behaviour editor and viewer
Problems not solved	<ul style="list-style-type: none"> • Root layout

	<ul style="list-style-type: none"> Loading media resources from list on the left instead of loading from the disk.
Configuration and execution context	

8.5 Draft User Manual

8.5.1 Editing the visual scene for SMIL (create/delete, resize/move)

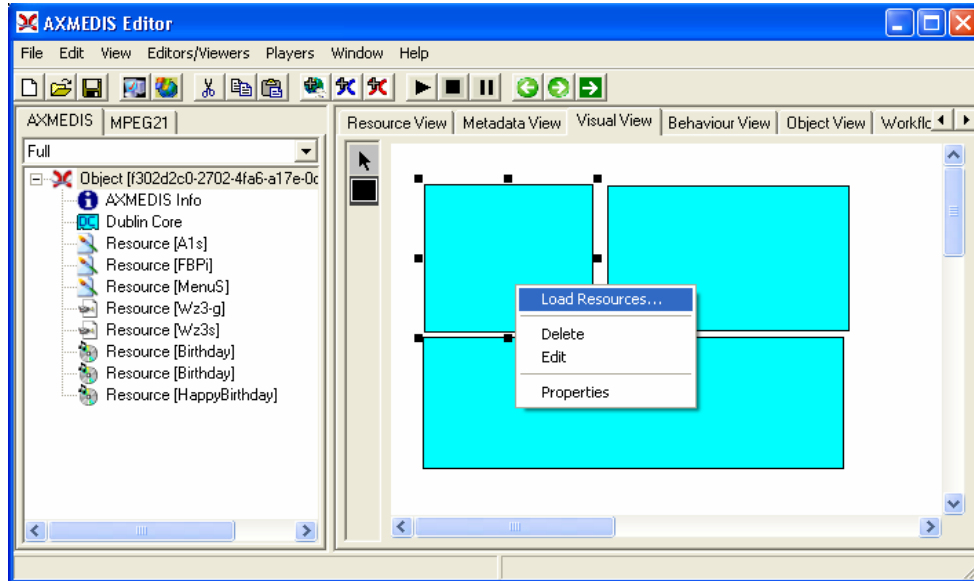
The user left clicks on the palette on the left of the Visual Editor. When the Square button is toggled, the user can create an Element (e.g., rectangular shape) on the canvas of the visual editor. The user left clicks on the canvas to create an element (e.g., rectangular shape) to represent the element area in the SMIL. Many of them can be arranged on the same scene. Some of them may not have a visual rendering but only an audible rendering.



(The above visual editor is not integrated with the SMIL Hierarchy Editor and Viewer)

8.5.2 Association of media resources within an element of SMIL

The user can click on the right button of the mouse (or choose the menu on the frame, in both cases) with the options of “load resources...” to include media resources.



(The above visual editor is not integrated with the SMIL Hierarchy Editor and Viewer)

8.6 Examples of usage

See the 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 Module Behaviour and Functional Editor and Viewer (EPFL)

Module/Tool Profile		
Behaviour and Functional Editor and Viewer		
Responsible Name	Beilu Shao	
Responsible Partner	EPFL	
Status (proposed/approved)	proposed	
Implemented/not implemented	Implemented	
Status of the implementation	60%	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows, Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/behaviour_editor_viewer	
Reference to the AXFW location of the demonstrator executable tool for internal download		
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	yes	
Usage of the AXMEDIS Error Manager (yes/no)	yes	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a

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

9.1 General Description of the Module

The Behaviour Editor together with the Visual Editor and Hierarchy Editor will provide the user with the infrastructure to produce multimedia presentations. A multimedia presentation can be composed of many media objects (text, audio, video, vector graphics...). The user will use the Hierarchy, Behaviour and Visual Editors to organize the media objects in space and time. As explained in the section 8, the user will first use SMIL Hierarch Editor to create the structure of the elements and then use the Visual Editor to place these media elements in different positions of the screen. There is a timeline at the top of the GUI of behaviour editor and viewer to indicate the time unit and entire length of demonstration for each container. The author can modify the parameters of this timeline by using the context menu.

The Behaviour Editor will complement the Visual Editor by adding time boundaries to the media objects. This means that every media object will be visible only for a period defined by the user. The simplest example for this is a slide show: the user specifies a group of slides and each slide is only visible during a slot of time defined by the user.

The Behaviour Editor will use a subset of the SMIL language to describe the multimedia presentation. SMIL (pronounced "smile") is defined as a set of XML modules which are used to describe the temporal, positional, and interactive behavior of a multimedia presentation.

Behaviour and Functional View is intended first as a View where the main modalities and layers of the Editor can be activated. The AXMEDIS Object can be a heterogeneous, multi-layer piece of information for which different modalities of exploration/manipulation are possible and for which different layers may be accessible according to preferences. In this sense the Functional View displays the available possibilities and allows selecting a text view, other than composited media view (scene layer) or a media-by-media view. The functional view may also allow displaying available modes of operation, like the selection between file / broadcast (save later, transmit immediately) and related configuration.

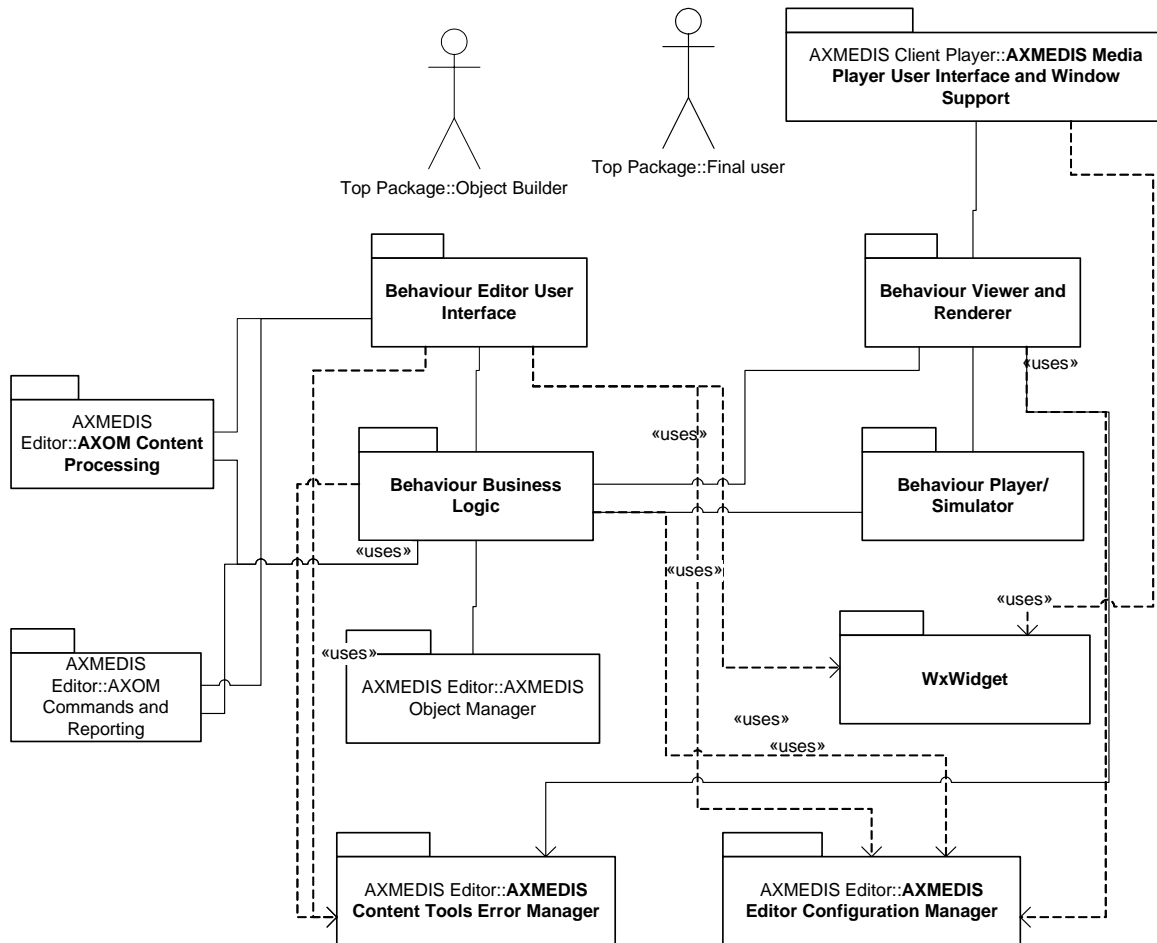
Behaviour and Functional View is also an interfacing view to external editors affecting synchronization and timing among different media objects. An AXMEDIS object may include multimedia and cross-media scenes like those that can be produced by MPEG-4 BIFS, SMIL, etc. Through suitable plug-ins and interfaces the AXMEDIS Editor may allow inserting and taking out portions or elements of these composited elements. Direct internal editing through internal functions and menus could be limited to a minimum of simple straightforward cases. Overall, the Behaviour and Functional View will:

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

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

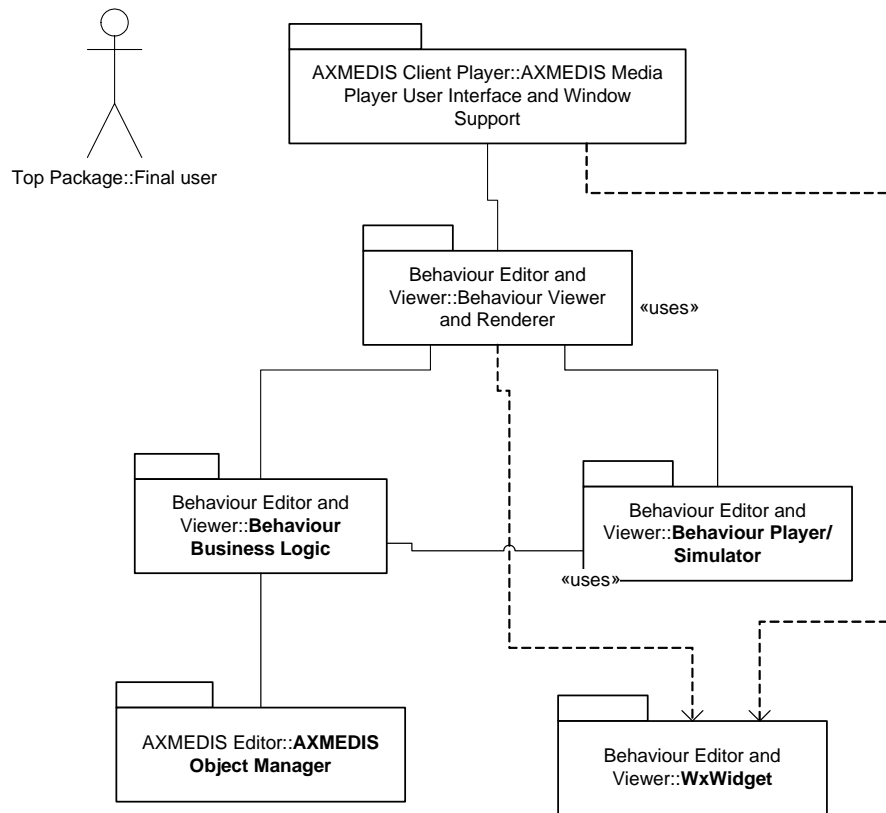
Behaviour Editor and Viewer

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



Behaviour Viewer/Player

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



9.1.1 Behaviour Business Logic

The Behaviour Business Logic together with the SMIL Hierarchy Business Logic will maintain an XML document. This XML document will describe the schedule of a group of media objects. It will be possible to specify when a media object will be displayed, for how long, and when the media object will abandon the display area. The temporal description will also be done according to the SMIL W3C (see Visual Editor Section). The user will interact with the Behaviour GUI to define, in a graphical way, the temporal information of the media objects inside each container (par/seq). The Behaviour Business Logic will receive notifications from the Behaviour GUI and it will modify the SMIL document accordingly. In this way, the SMIL document will become a text description of the temporal planning that the user will design in a graphical manner.

It is important to notice that the SMIL document edited by the Behaviour Business Logic is the same document edited by the Visual Editor as well as the SMIL Hierarchy Editor. However, there will be no conflict because the SMIL Hierarchy Editor, the Behaviour Editor and the Visual Editor will modify different parts of the same document. The SMIL Hierarchy Editor is concerned with the entire hierarchy structure of SMIL elements, The Visual editor is concerned with the spatial editing, and the Behaviour Editor regards the temporal editing. Fortunately, a SMIL document keeps in separate sections the schedule and the layout. The SMIL document will be held inside the AXMEDIS document and will be treated as other components. This means that the SMIL document will be accessed via the Command Manager.

9.1.1.1 Temporal Behavior of SMIL

The Behaviour Business Logic will be able to handle a subset of the SMIL language. To incorporate media objects in the presentation we will need the following media tags:

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

The synchronization elements:

- *par:* the [<par>](#) element plays child elements as a group (allowing "parallel" playback).
- *Seq:* the [<seq>](#) element plays child elements as a group(allowing "sequential" playback)

The attributes about the temporal information:

- *begin:* defines when the element will be visible (start playing).
- *end:* defines when the element will be invisible (stop playing).

Putting it all together, we could have something like this:

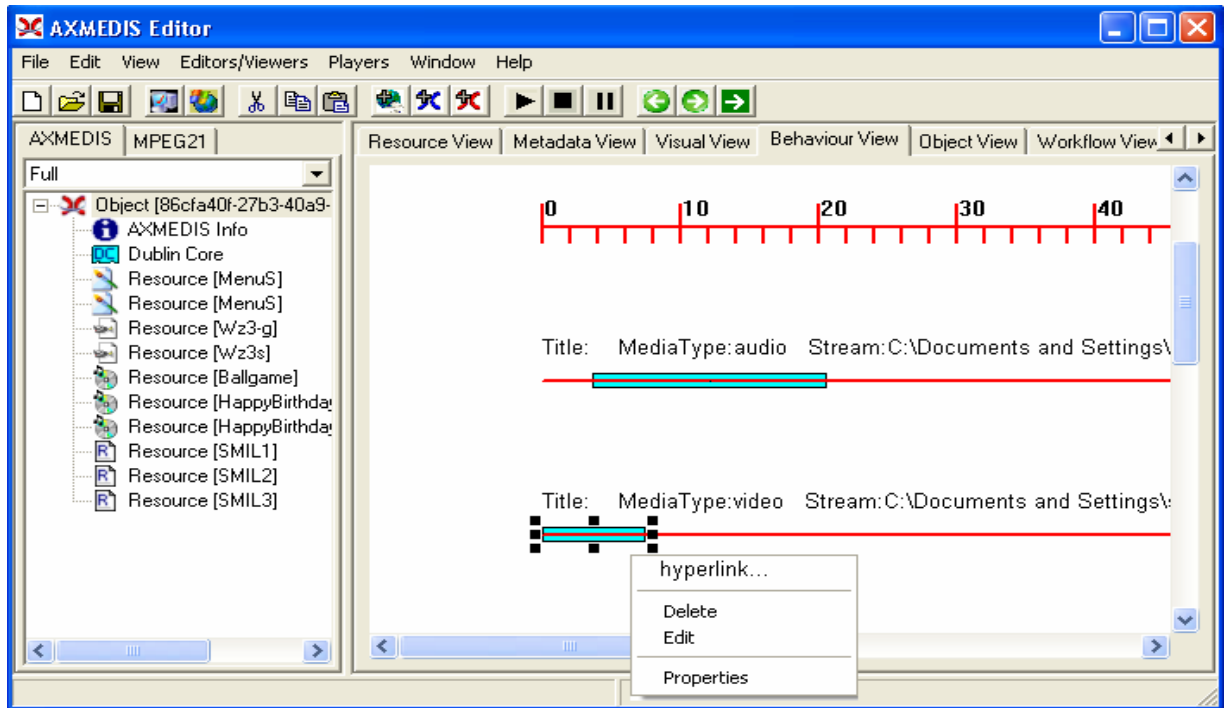
```
<body>
  <par>
    <par dur="30s">
      
      <audio id="Welcome-US" region="LogoRegion"
src="LITEdataCE/Welcome-US.mp3" dur="21.4s" />
    </par>
    <seq>
      
      <audio id="Welcome" region="LogoRegion" src="LITEdataCE/Welcome-
US.mp3" dur="2s" />
    </seq>
  </par>
</body>
```

9.1.1.2 Navigation and hyperlinking with multiple SMIL scenes

The link elements allow the description of navigational links between objects. SMIL provides only for in-line link elements. Links are limited to uni-directional single-headed links (i.e. all links have exactly one source and one destination resource). All links in SMIL are actuated by the user. Navigation and hyperlinking with multiple SMIL scenes is supported in the behavior editor. This function is supported by [<a>](#) tag in the SMIL specification. This enables the user to connect another SMIL scene by clicking "link" which is at the displaying scene. For the hyperlinking feature, the editor will support the following structure of the SMIL:

```
<a href= "somewhereelse.smil">
  <video src=" party1.mpg " >
</a>
```

href attribute contains the URI of the link's destination. The second line will be replaced by a reference to a new SMIL presentation with the URI "somewhereelse.smil". Therefore when a viewer clicks the video, a new presentation will start up in presentation.



(The above visual editor is not integrated with the SMIL Hierarchy Editor and Viewer)

After clicking “hyperlink...” the user will have a menu list of SMIL components on the left panel to choose the target file for the presentation. And then the behaviour business logic will write the information into the SMIL file as follows:

```
<a href= "somewhereelse_for_hyperlinking.smil">
  <video src=" party1.mpg " >
</a>
```

Once the SMIL is under execution for preview, it includes some buttons at which another SMIL scene is connected via a link and thus the current presentation is replaced by another SMIL presentation. On the other hand, if the user does not want to link to other resources, the user does not need to do anything and the current SMIL presentation will repeat itself.

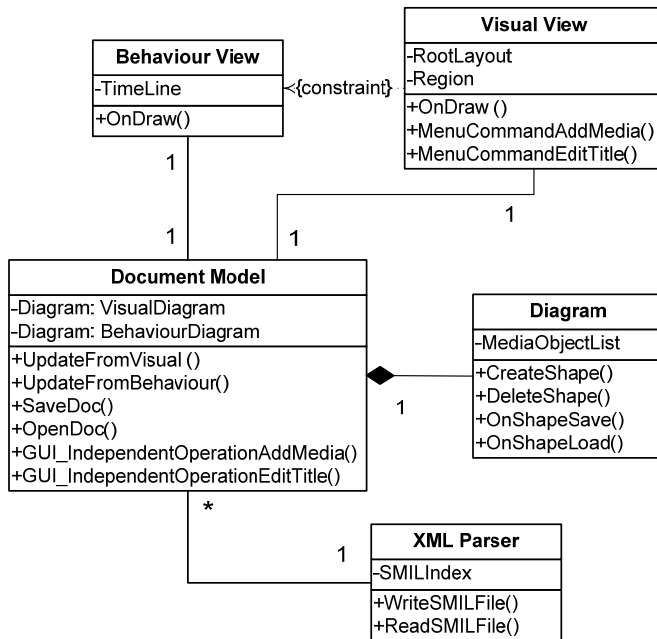
To make the SMIL repeat, the tag <repeat> is supported. The repeat="indefinite" attribute causes a clip or group to repeat until another SMIL attribute or user event stops the playback loop. However, this attribute works only with some of the SMIL player. In the following example, the audio clips repeats continuously until the viewer clicks the Player **Stop** button:

```
<audio src="music/newsong.rm" repeatCount="indefinite"/>
```

For Ambulant SMIL player (version 1.6.2), this attribute is partially supported for seq and par elements, for these elements it is needed to specify the dur attribute like in the following example:

```
...
<par dur="9s" repeatCount="indefinite">
  <img begin="0s" end="3s" ...>
  <img begin="3s" end="6s"...>
  <img begin="6s" end="9s"...>
</par>
```

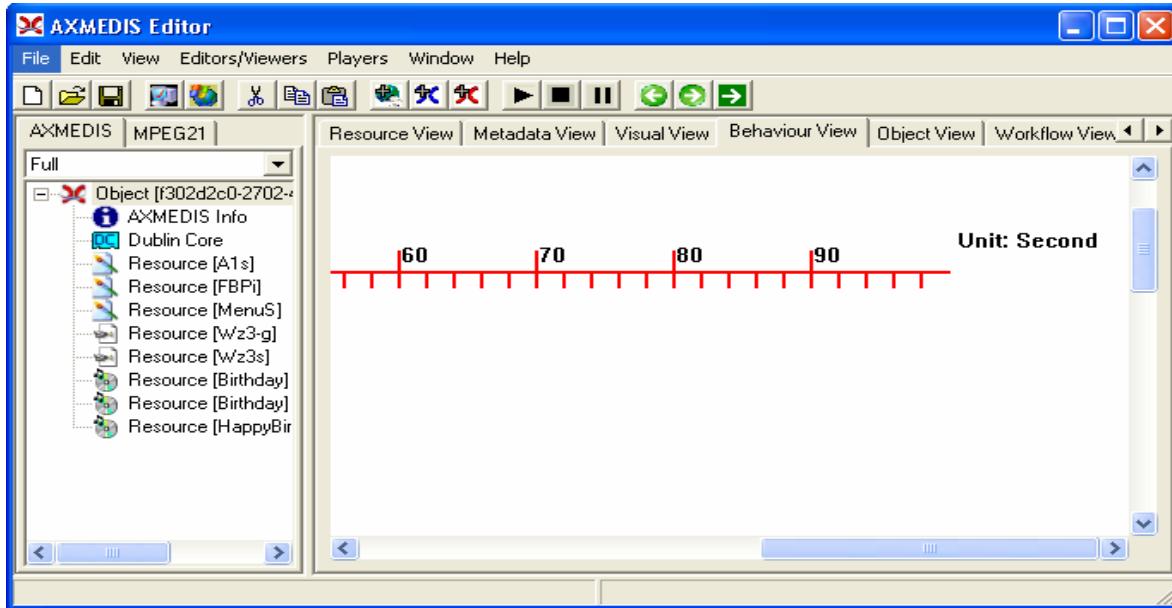
9.2 Module Design in terms of Classes



9.3 User interface description

The Behaviour GUI will display graphically the temporal boundaries of every media object. This GUI will have a horizontal time line similar to a ruler (see next picture) with marks on it that will indicate the time. The media objects will be represented as horizontal bars of different lengths according to the duration of the media object. The bars will be drawn under the time line in such a way that the limits of a bar will be aligned with the start and end times. This way, every bar will give a graphical impression of when a media object will start playing and when will it stop.

To add, modify or remove bars in this GUI the user will use a contextual menu accessible via right click. To associate a media object with a bar, the user will have to select a media object from the Hierarchical View and drag it onto the bar. To define the region of the screen where the media object will be rendered, the user will have to select a rectangle from the Visual Editor and drag it onto the bar.

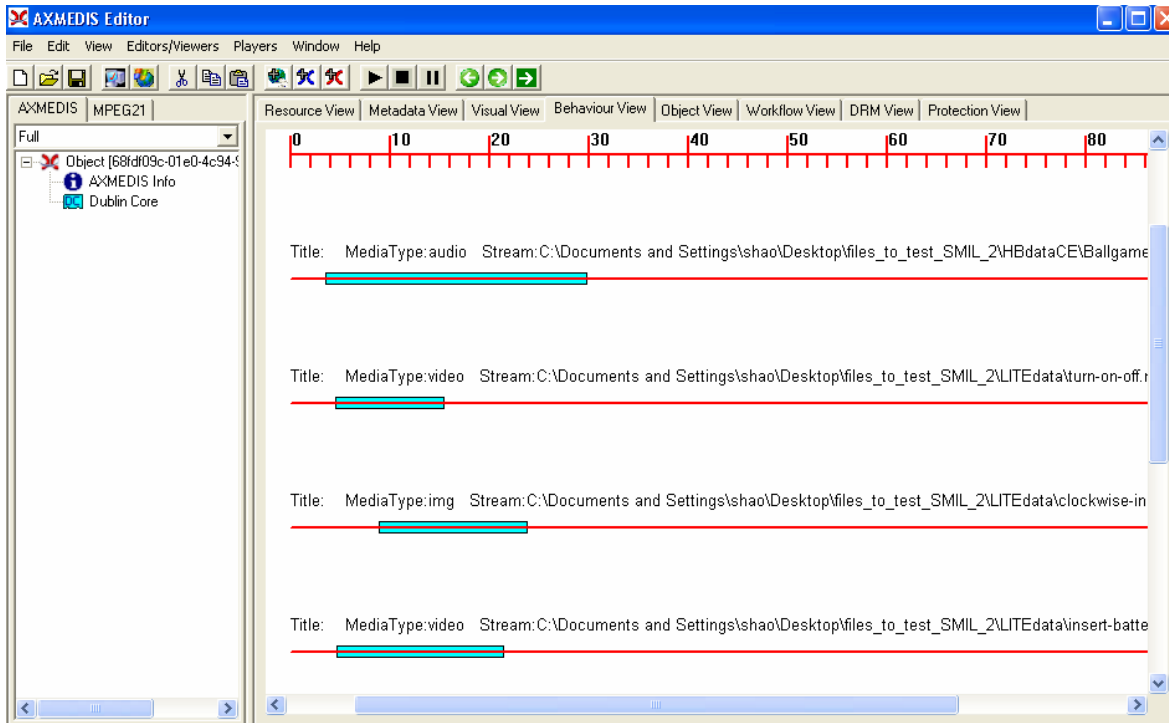


(The above visual editor is not integrated with the SMIL Hierarchy Editor and Viewer)

A draft contextual menu can be used to modify the parameters of the timeline in terms of the entire demonstration time and the unit (second/minute/hour)

The following figure demonstrates the GUI to set the temporal attributes of the synchronization elements of each media resource based on the timeline.





(The above visual editor is not integrated with the SMIL Hierarchy Editor and Viewer)

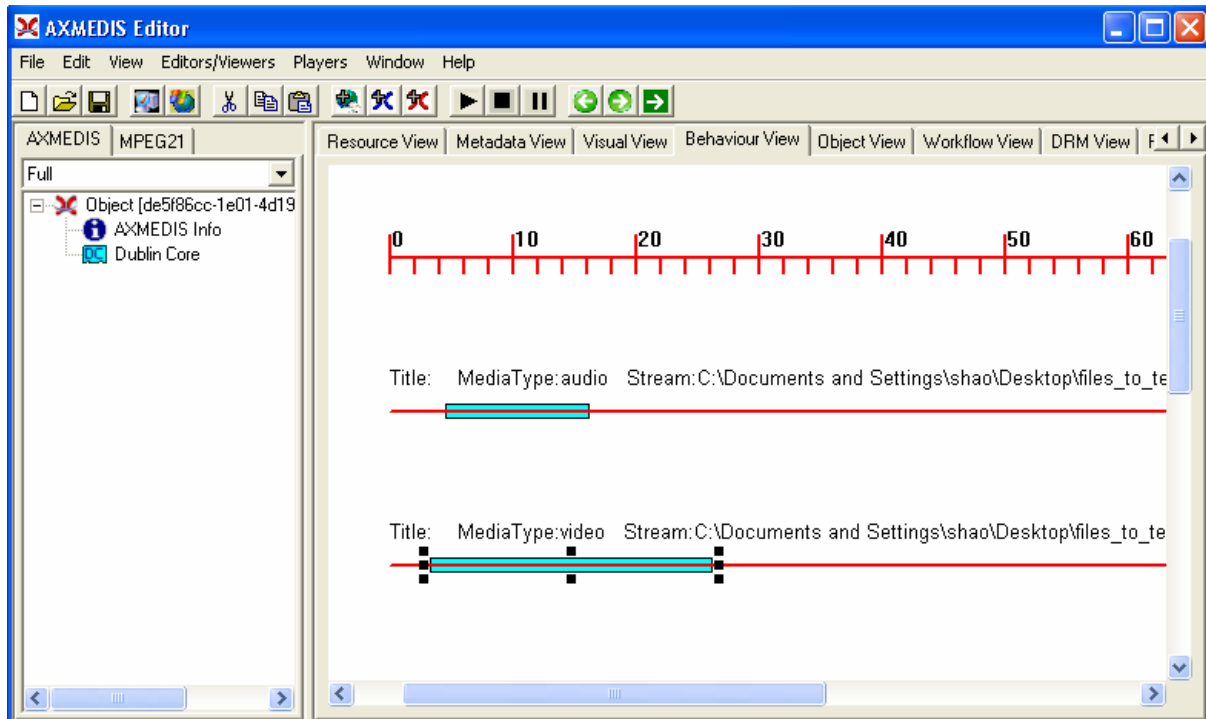
9.4 Technical and Installation information

References to other major components needed	<ul style="list-style-type: none"> Visual editor and viewer
Problems not solved	<ul style="list-style-type: none"> Support of <par> tag for the simultaneous display of objects. Navigation and hyperlinking with multiple SMIL Scenes
Configuration and execution context	

9.5 Draft User Manual

9.5.1 Editing the temporal information of media resources

The user can modify the length and position of the rectangle on the time line to indicate the different starting time and display duration



9.6 Examples of usage

See the Draft User Manual

9.7 Integration and compilation issues

None

9.8 Configuration Parameters

Config parameter	Possible values

9.9 Errors reported and that may occur

Error code	Description and rationales

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

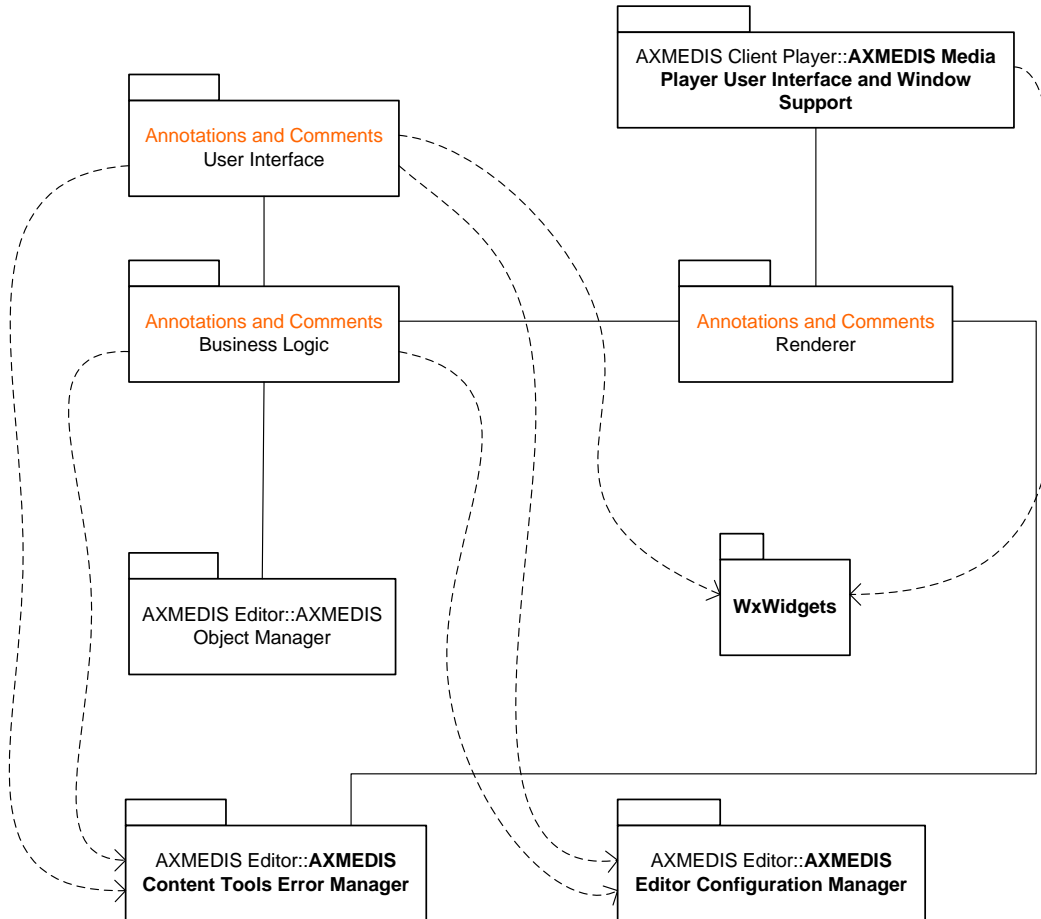
Module/Tool Profile		
AXMEDIS Object Editor and Viewer (Descriptions and Comments)		
Responsible Name	Claudio Alberti and Beilu Shao	
Responsible Partner	EPFL	
Status (proposed/approved)	proposed	
Implemented/not implemented		
Status of the implementation		
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows, Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/object_editor_viewer	
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)	yes	
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.
	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

10.1 General Description of the Module

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

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



10.1.1 Business Logic

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

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

10.2 Module Design in terms of Classes

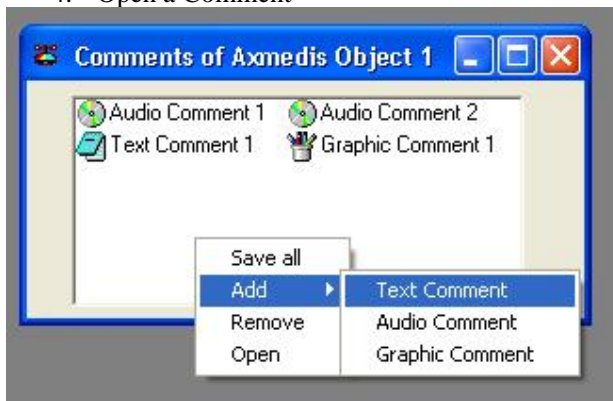
10.3 User interface description

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

10.3.1 Main GUI

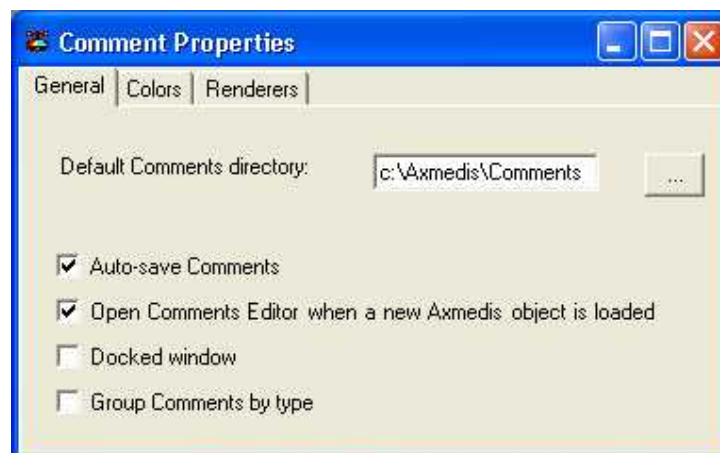
The main GUI will show the list of Comments associated with one AXMEDIS object. This GUI will permit the following operations by means of a contextual menu (see next picture) :

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



10.3.2 Configuration GUI

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



10.3.3 Renderer GUIs

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

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



10.4 Draft User Manual

10.5 Examples of usage

10.6 Integration and compilation issues

10.7 Configuration Parameters

Config parameter	Possible values

10.8 Errors reported and that may occur

Error code	Description and rationales

11 Module Metadata Editor and Viewer (UNIVLEEDS)

Module/Tool Profile	
Metadata Editor and Viewer	
Responsible Name	Kia Ng and Royce Neagle
Responsible Partner	UNIVLEEDS

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

User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWindows	wxWindows, v-2.5.3 or v-2.4.2	LGPL
Xerces	Xerces-c++ v-2.6.0	Apache Software Licence, v2.0
Xalan	Xalan-c++ v-1.9	The Apache Software License, v1.1

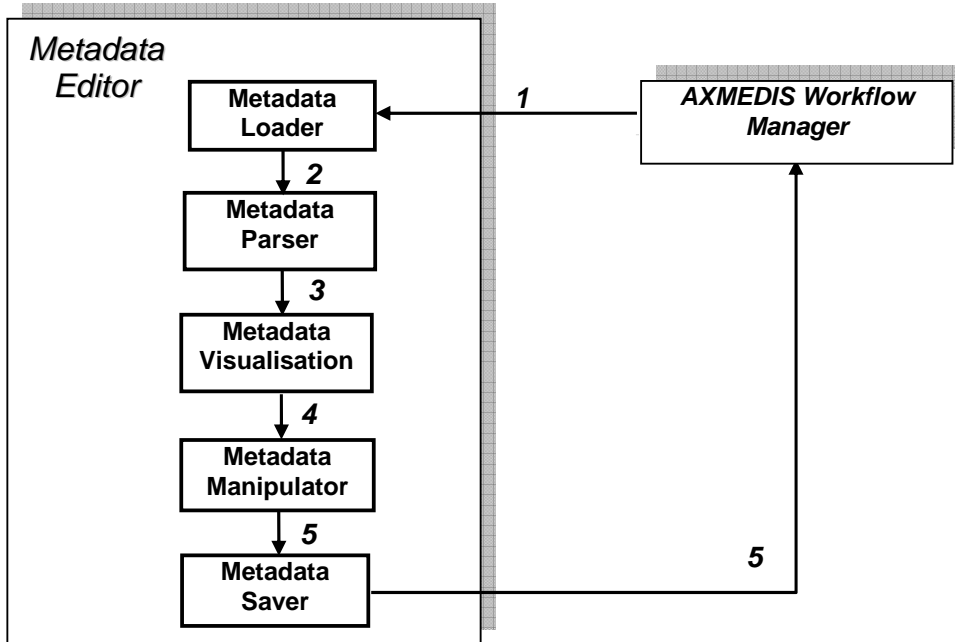
11.1 General Description of the Module

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

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

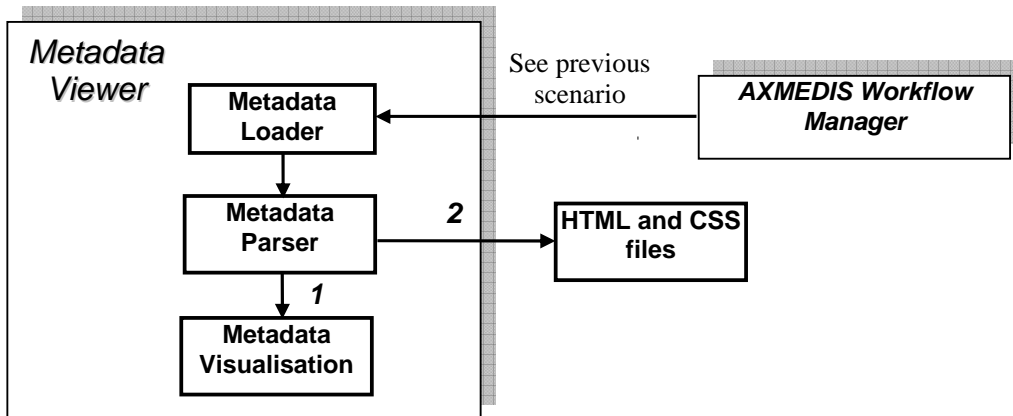
Scenario for editing metadata

1. Receive metadata information in XML from WF
2. Parse XML data to obtain a list of metadata tag and value
3. Generate visualisation for metadata
4. Provide functionality for the actor to modify the metadata
5. Provide functionality to send back the revised metadata to the originator (via WF)

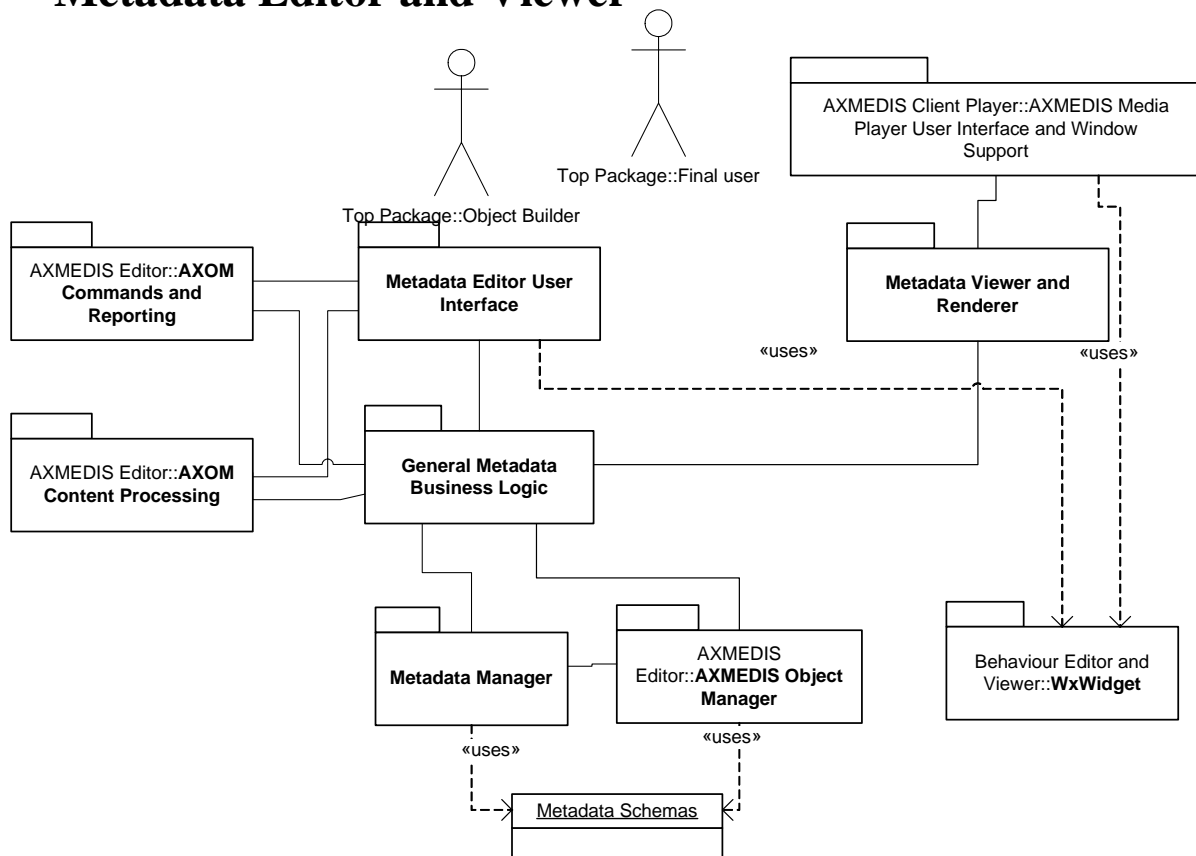


For Metadata visualisation, two possible approaches can be done:

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

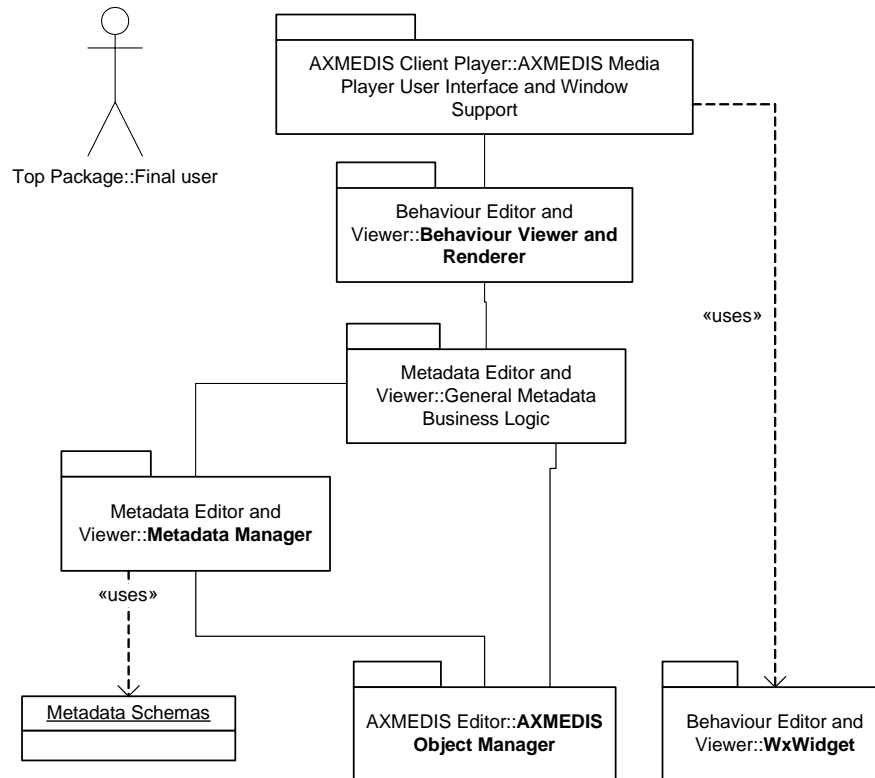


Metadata Editor and Viewer



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

Metadata Viewer



11.1.1 General Metadata Business Logic

The General Metadata Business Logic provide the navigation functionalities to traverse a given XML document based on the structure and relationships modelled by the Metadata manager using a schema. It is particularly important for the Metadata editor to know what is the valid child for a particular nodes depending on the context, where the user intended to add an elements. The Business Logic preserves the structure integrity and ensure the correctness of the updated XML.

11.1.2 Metadata Manager

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

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

11.1.3 Metadata Schemas

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

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

11.1.4 Metadata Viewer and Renderer

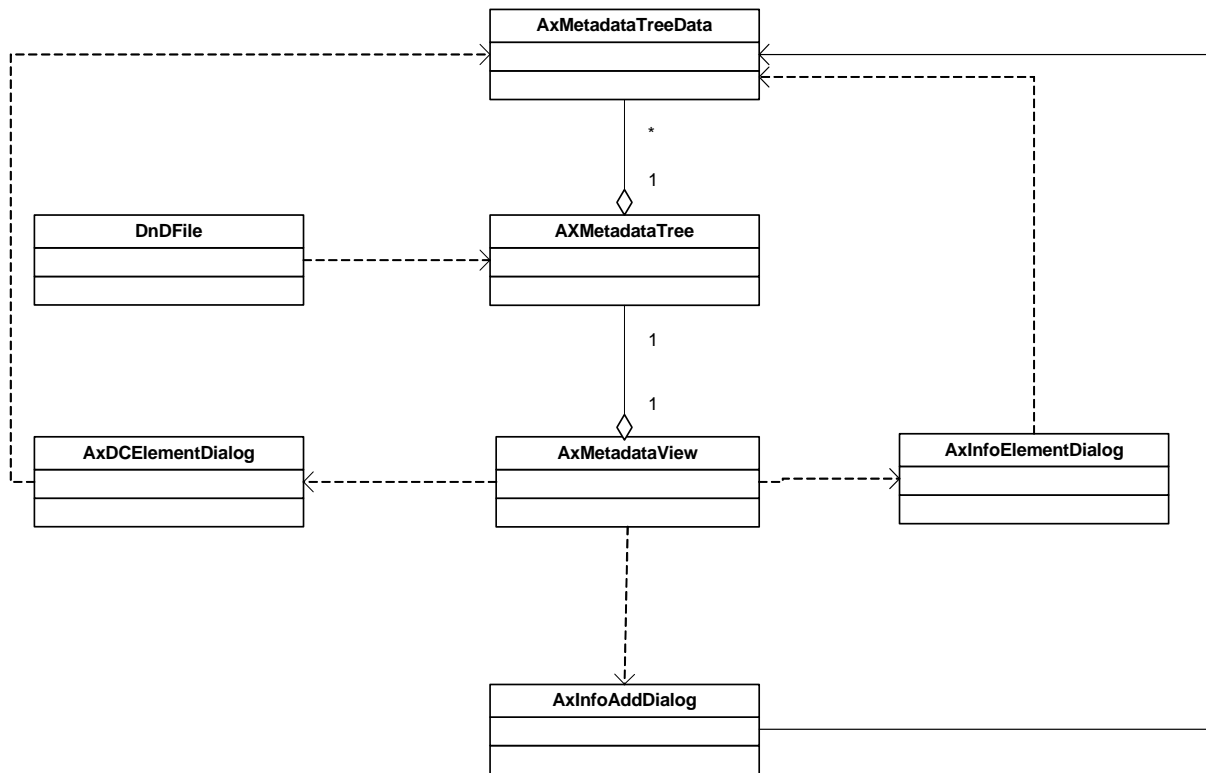
There are two approaches to achieve the metadata visualisation:

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

The Metadata Editor (1) has been integrated into the AXEditor (see section User Interface for screenshots) where editing functions are achieved through the use of an editing dialogue as presented in the User Interface section.

The generation of HTML will be accomplished using the menu option to open a web browser and display the XML with generated with either pre-defined or user-defined CSS.

11.2 Module Design in terms of Classes



11.3 User interface description

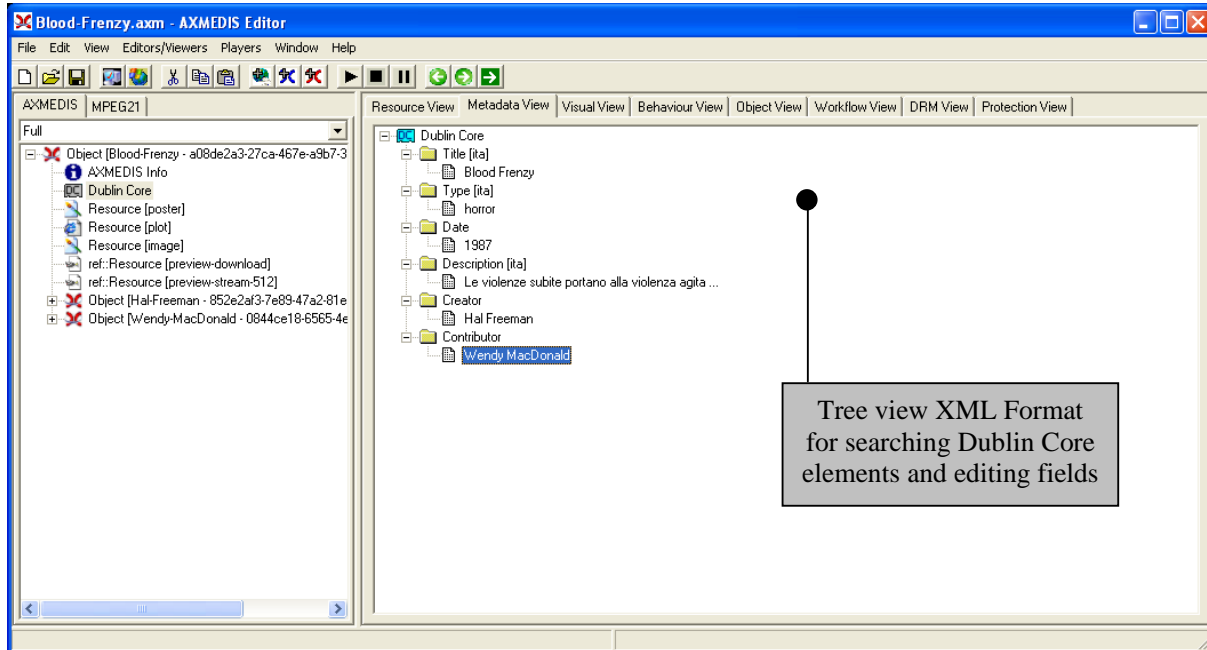


Figure: Screenshot of the AxEditor open in the Metadata View showing the tree view of the metadata elements (right window). The metadata view can be selected by double clicking on the metadata elements in AXMEDIS View of the AXMEDIS Object (left window).

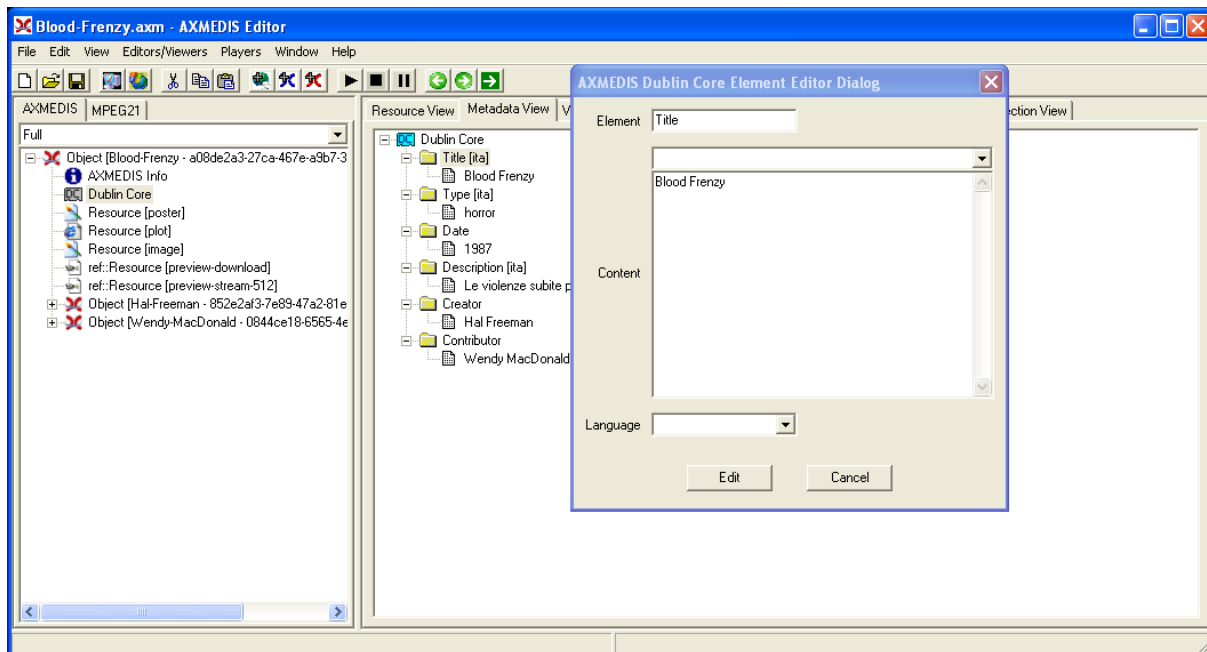


Figure: Screenshot of the editing window of the metadata elements using the editing metadata dialogue box to add and edit both the content and attributes.

Tree view area in the Metadata View

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

Metadata Editing Dialogue

This view presents the element name, content editing window and element attribute drop down box to edit the metadata element selected.

11.4 Technical and Installation information

The editor is integrated into the AXEditor, please see AxEditor technical and installation information for more details.

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

11.5 Draft User Manual

11.6 Examples of usage

11.7 Integration and compilation issues

none

11.8 Configuration Parameters

Config parameter	Possible values

11.9 Errors reported and that may occur

Error code	Description and rationales

12 Module - Metadata Mapper Editor and Viewer (UNIVLEEDS)

Module Profile	
Metadata Mapper Editor and Viewer	
Responsible Name	Garry Quested and Royce Neagle
Responsible Partner	UNIVLEEDS

Status (proposed/approved)	Proposed	
Implemented/not implemented	Prototype with basic GUI implemented	
Status of the implementation	Prototype with basic GUI	
Executable or Library/module (Support)	Executable	
Single Thread or Multithread	Multi Thread	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Applications/metadatamapperGUI https://cvs.axmedis.org/repos/Framework/source/metadatamapper/ https://cvs.axmedis.org/repos/Framework/include/metadatamapper/	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/repos/Applications/metadatamapperGUI/bin/win32/metadatamapperGUI.exe	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)	Absent	
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved		
Major pending requirements	GUI improvements	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
yes	metadatamapper.lib https://cvs.axmedis.org/repos/Framework/bin/	

	metadatamapper/win32/metadatamapper.lib	
Formats Used	Shared with	format name or reference to a section
XSLT	metadatamapper	XSLT, Standard format
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
yes	C++	wxWidgets, Object Graphics Library, PC,
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
yes	wxWidgets-2.4	wxWindows licence (http://www.wxwidgets.org/newlicen.htm)
	Object Graphics Library of wxWidgets-2.4	wxWindows licence (http://www.wxwidgets.org/newlicen.htm)

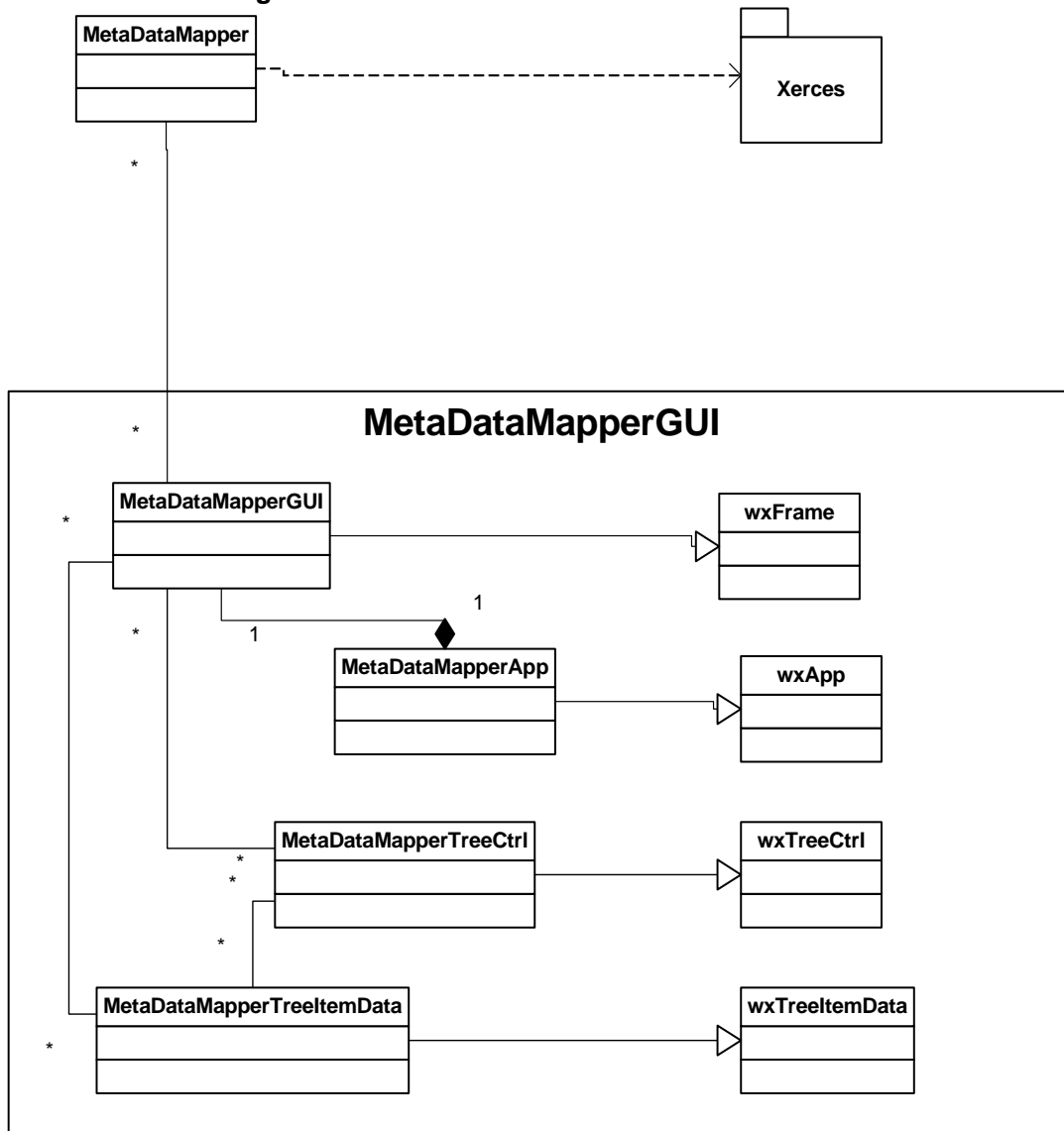
12.1.1 General Description of the Module

The MetaDataMapperGUI allows a user to generate metadata mappings from one XML language to another using a graphical interface. A source and destination metadata file are loaded into the application and then the user defines links with pair(s) of a source element to a destination element. The mappings can be saved to file and this file can be used to convert a metadata file in the source language to a metadata file in the destination language.

The Mapper comprises of two parts:

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

12.1.2 Module Design in terms of Classes



12.1.3 User interface description

The user interface consists of three components:

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

12.1.4 Technical and Installation information

- Once compiled, the executable of the metadatamapperGUI can run provided the appropriate DLLs are available. These can all be found in the metadatamapperGUI section of the repository

Problems not solved	The metadatamapper is currently a prototype. Two main areas need work: <ol style="list-style-type: none"> 1. The mapping of elements can be improved to handle more
---------------------	--

	complex mapping rules 2. The GUI can be improved to provide a better user experience 3. The prototype can only support one layer Metadata tree structure at this stage
--	--

12.1.5 Draft User Manual

There is no user manual currently available

12.1.6 Examples of usage

First a user loads a source and target metadata file. Next they connect elements to create mapping information. When the user has mapped all the elements they require, they save a map file by clicking the save toolbar button. An XSLT file will be saved on the users system which can be used for metadata adaption

12.1.7 Integration and compilation issues

None

12.1.8 Configuration Parameters

Config parameter	Possible values

12.1.9 Errors reported and that may occur

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

13 Module Workflow Editor and Viewer (DSI)

Module/Tool Profile	
Workflow User Interface	
Responsible Name	Bellini
Responsible Partner	DSI
Status (proposed/approved)	Approved
Implemented/not implemented	Implemented
Status of the implementation	First Prototype Completed
Executable or Library/module (Support)	Library
Single Thread or Multithread	Multithreaded
Language of Development	C++
Platforms supported	Windows
Reference to the AXFW location of the source code	https://cvs.axmedis.org/repos/Framework/source/workflow_editor_viewer

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

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

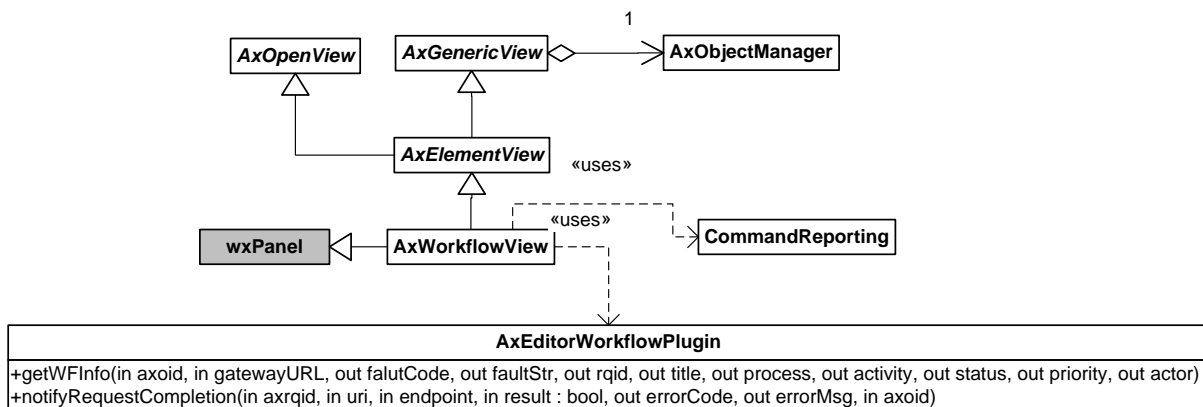
13.1 General Description of the Module

The workflow editor and viewer integrated inside the AXMEDIS Editor is the gateway interface for accessing to the workflow management system.

The functionalities supported by the module are:

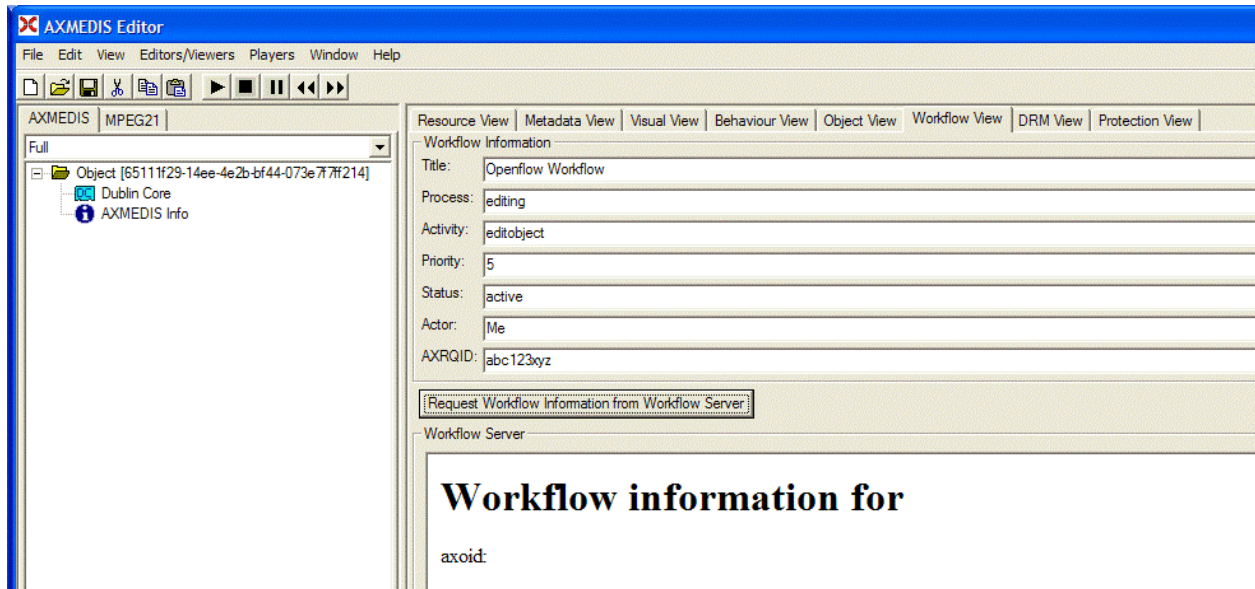
- view the status in the Workflow Management System of the currently opened object
- update inside the object the workflow status that is present in the AXInfo metadata
- give access to the workflow web interface

13.2 Module Design in terms of Classes



13.3 User interface description

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



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

The information acquired is:

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

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

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

13.4 Technical and Installation information

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

13.5 Draft User Manual

See User Interface Description

13.6 Examples of usage

See User Interface Description

13.7 Integration and compilation issues

None

13.8 Configuration Parameters

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

WORKFLOW/gatewayUrl	the url of the gateway

13.9 Errors reported and that may occur

Error code	Description and rationales

14 Module - AXMEDIS Content Tool Error Manager (DSI)

Module/Tool Profile	
<name of the module>	
Responsible Name	Vallotti
Responsible Partner	DSI
Status (proposed/approved)	
Implemented/not implemented	not implemented
Status of the implementation	
Executable or Library/module (Support)	Library
Single Thread or Multithread	Single thread
Language of Development	C++
Platforms supported	
Reference to the AXFW location of the source code demonstrator	NA
Reference to the AXFW location of the demonstrator executable tool for internal download	NA
Reference to the AXFW location of the demonstrator executable tool for public download	
Address for accessing to WebServices if any, add accession information (user and Passwd) if any	
Test cases (present/absent)	absent
Test cases location	
Usage of the AXMEDIS configuration manager (yes/no)	no
Usage of the AXMEDIS Error	no

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
Error Coding		Section 26
Error Log		Section Errore. L'origine riferimento non è stata trovata.
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
WxWIDGET	wxWidgets 2.4.2	LGPL
log4cxx		Apache License version 2.0

14.1 General Description of the Module

AXMEDIS Content Tool Error Manager is an interface through which all software modules are allowed to log errors in an independent way w.r.t. to the language. Moreover, the error manager allow the user to graphically visualize the error logs.

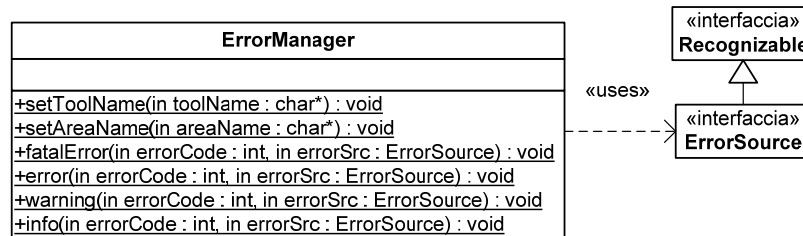
To achieve language independence errors have to be pre-defined and identified and, at runtime, they should be handle through identifiers instead of using their text descriptor. Therefore, in AXMEDIS Framework errors will be uniquely identified by the following information:

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

As stated after in this section, error identifiers (area name, full class name, error code) are someway associated with other useful information such as a short description, recovery note, etc which are language dependant.

The above data can be statically defined in the code or dynamically determined at runtime. To determine that information at runtime an infrastructure for error management is needed. That infrastructure should consist of one or more classes which has to be used by all modules of AXMEDIS Framework (see below).

14.2 Module Design in terms of Classes



ErrorManager class exposes the following static functions

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

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

14.3 User interface description

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

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

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

The GUI exposes the following functionalities:

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

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

14.4 Draft User Manual

See User Interface Description

14.5 Examples of usage

See User Interface Description

14.6 Configuration Parameters

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

14.7 Errors reported and that may occur

Error code	Description and rationales

15 Module - AXMEDIS Editor Configuration Manager (DSI, EPFL)

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

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

15.1 General Description of the Module

AXMEDIS Editor Configuration Manager will be the unique access point to AXMEDIS Editor modules configurations. Each configurable AXMEDIS Editor modules (and sub modules) will respect AXMEDIS Editor Configuration Manager requirements.

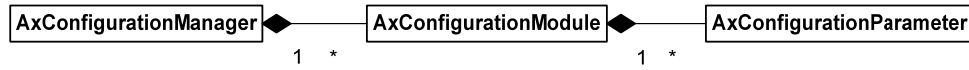
AXMEDIS Editor Configuration Manager User Interface will be flexible enough to manage the widest range of settings possible. That should be possible by means of plug-in support.

Further, AXMEDIS Editor Configuration Manager will manage and provide AXMEDIS Editor general configurations, such as multi-language.

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

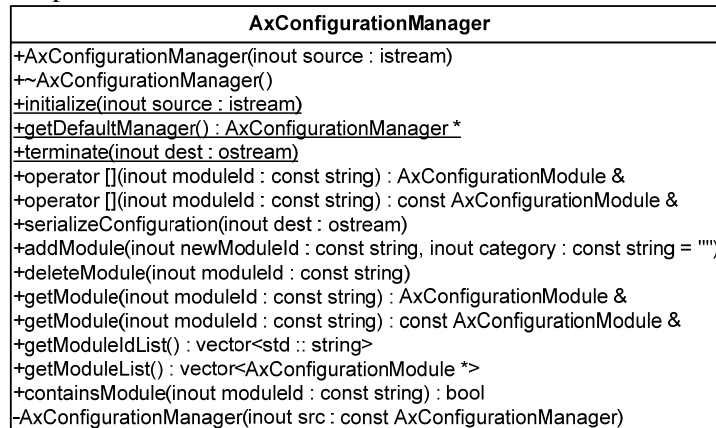
15.2 Module Design in terms of Classes

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



The Configuration Manager is composed of three main classes:

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



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

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

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

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

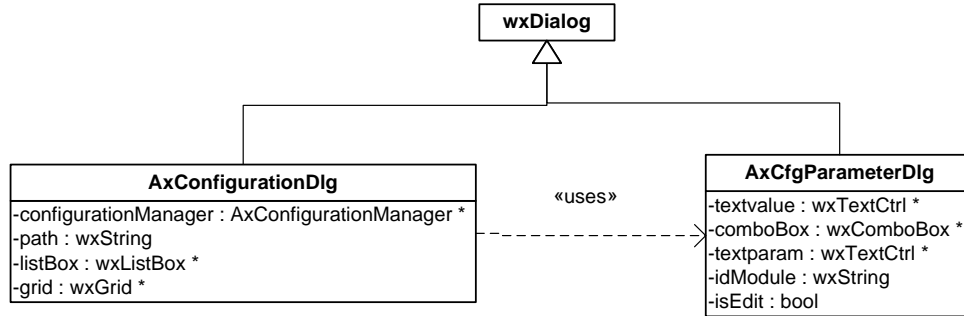
- *addParameter* – creates a new parameter having the given name (*newParamName*). The parameter name have to be unique within the module than the function does not create two parameter with the same name.
- *containsParameter* – checks if the module contains a parameter with the given name (*paramName*).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

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

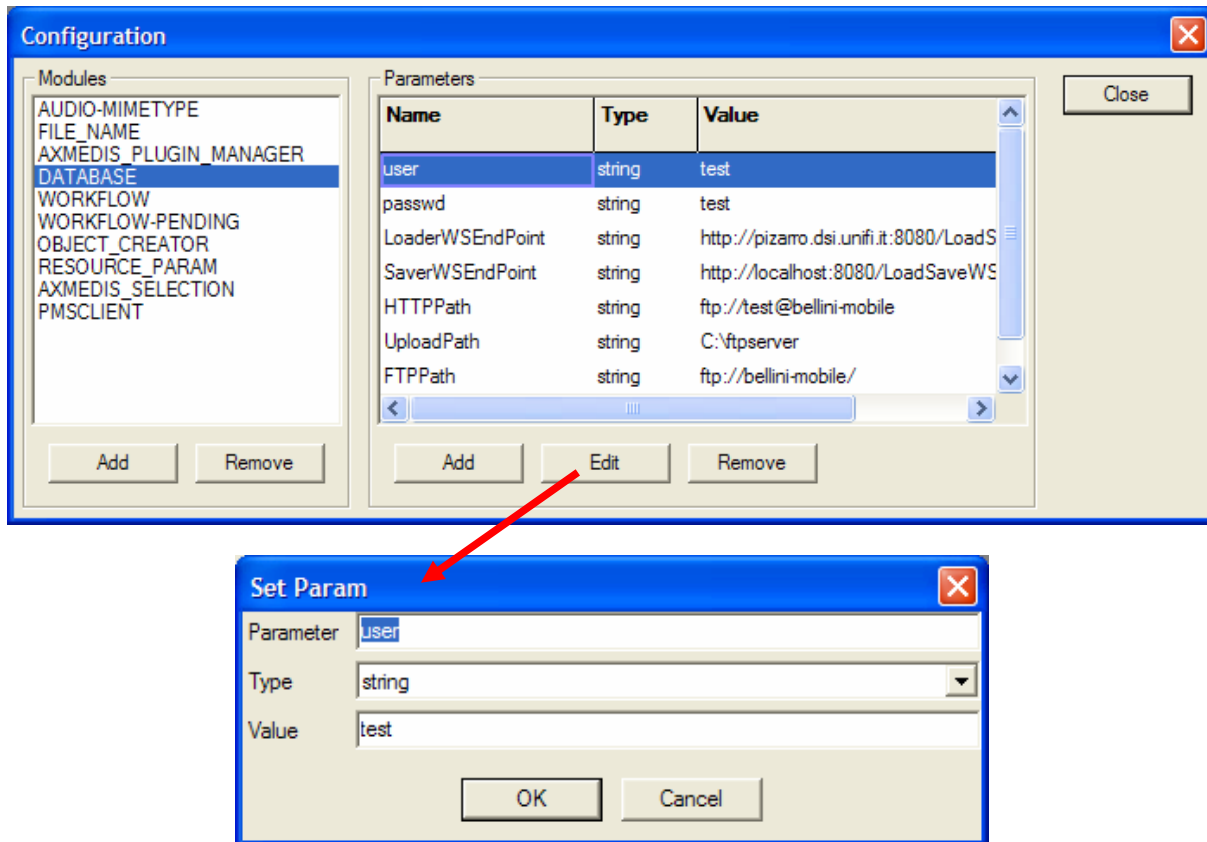
AxConfigurationParameter exposes several functions to allow management of different type of parameter in an easy manner. In particular, it exposes methods to set and get the value of a parameter and to obtain the name and the type of the parameter itself.

15.3 User interface description

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



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



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

15.4 Technical and Installation information

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

15.5 Examples of usage

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

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

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

15.6 Integration and compilation issues

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

15.7 Configuration Parameters

Config parameter	Possible values

15.8 Errors reported and that may occur

Error code	Description and rationales

16 Module - AXMEDIS Editor Plug-in Manager (DSI, EPFL)

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

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

16.1 General Description of the Module

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

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

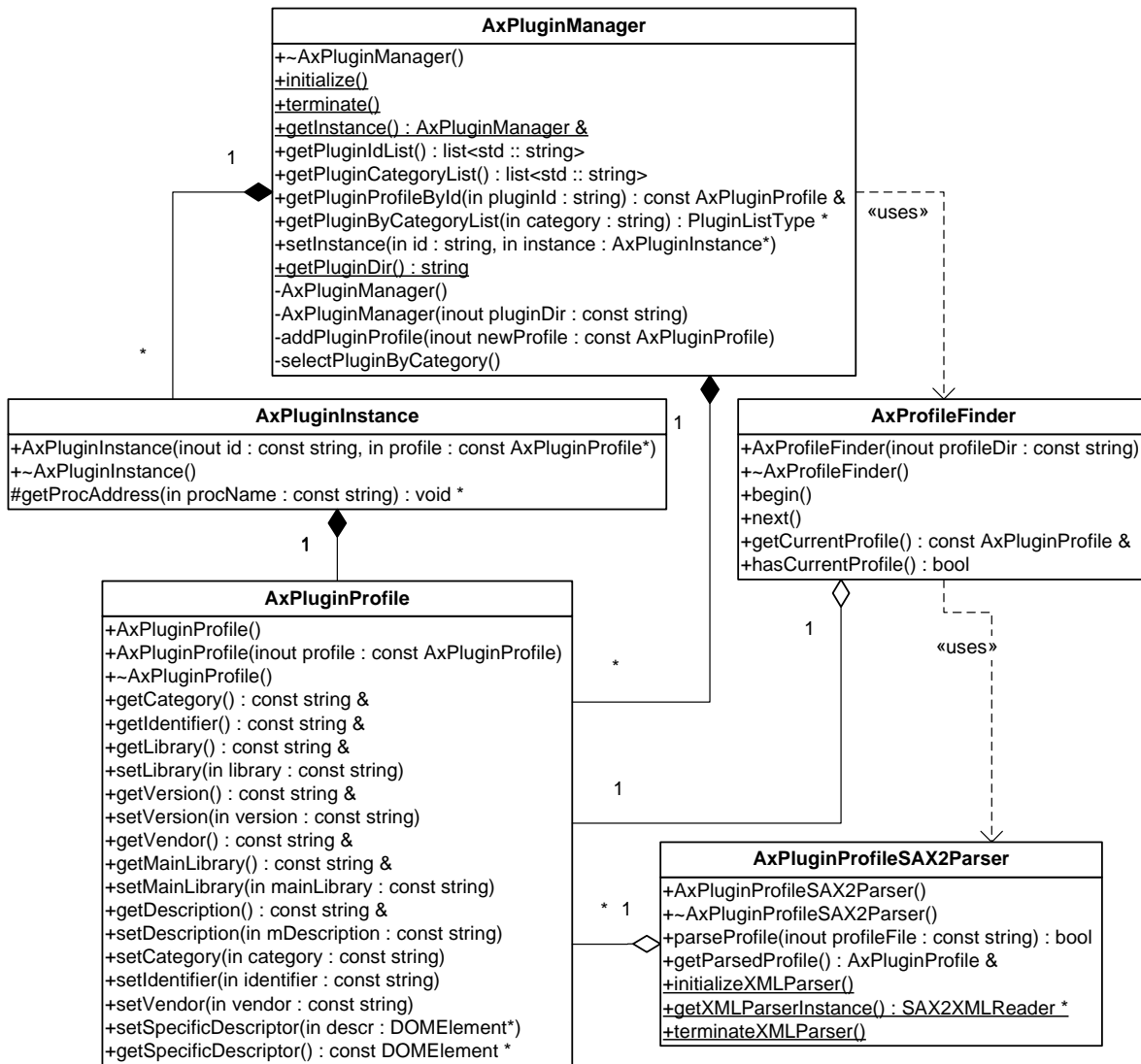
Each plug-in is described by a XML file (namely a profile) which contains the following common information:

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

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

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

16.2 Module Design in terms of Classes



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

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

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

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

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

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

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

16.2.1 Fundamental classes overview

AxPluginManager

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

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

The others are utility functions.

AxPluginProfile

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

AxPluginInstance

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

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

16.3 Integration and compilation issues

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

16.4 Configuration Parameters

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

16.5 Errors reported and that may occur

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

17 Module - AXOM Content processing (DSI, EPFL)

Module/Tool Profile		
AXOM Content processing		
Responsible Name	Vallotti	
Responsible Partner	DSI	
Status (proposed/approved)		
Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Single thread	
Language of Development	C++	
Platforms supported	Windows, Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/contentprocessing/	
Reference to the AXFW location of the demonstrator executable tool for internal download		
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)		
Usage of the AXMEDIS Error Manager (yes/no)		
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Content Processing Plug-ins specific description		Section 30
Parameter description		Section 31

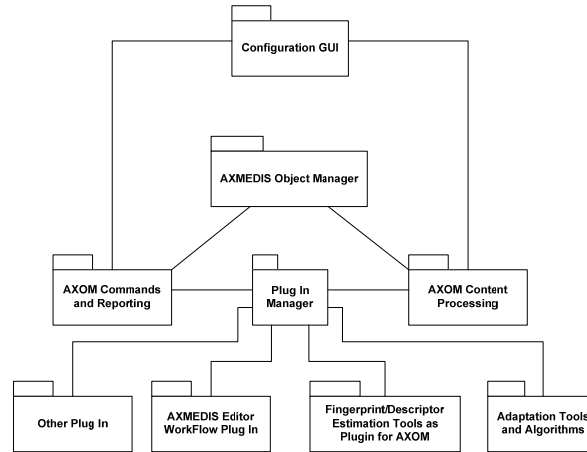
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

17.1 General Description of the Module

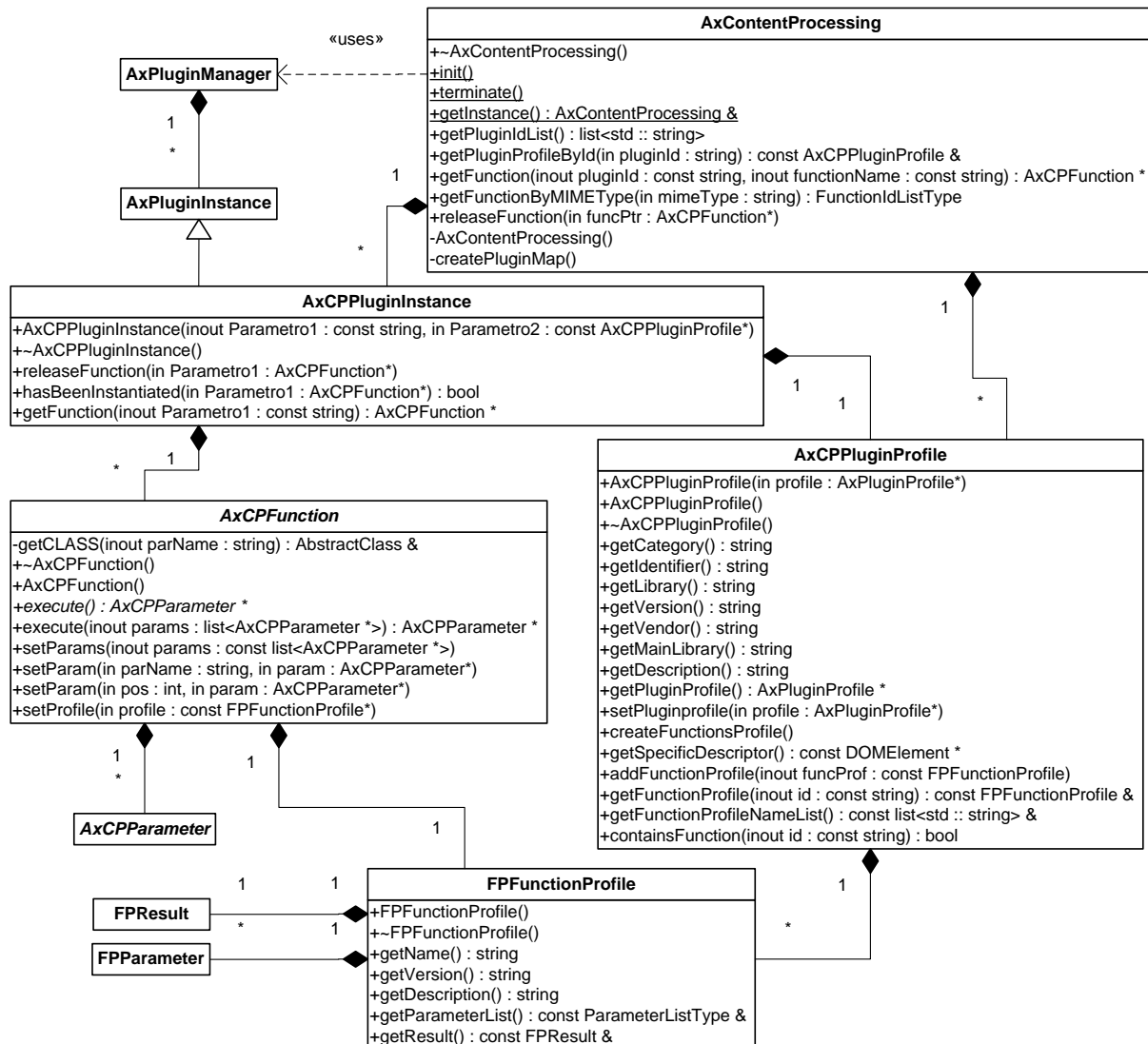
AXOM Content Processing is the interface the AXOM uses to call dynamic functions for content processing. Content processing function belongs to the following categories: Fingerprint estimation, Descriptor estimation, Adaptation algorithm, etc...

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

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



17.2 Module Design in terms of Classes

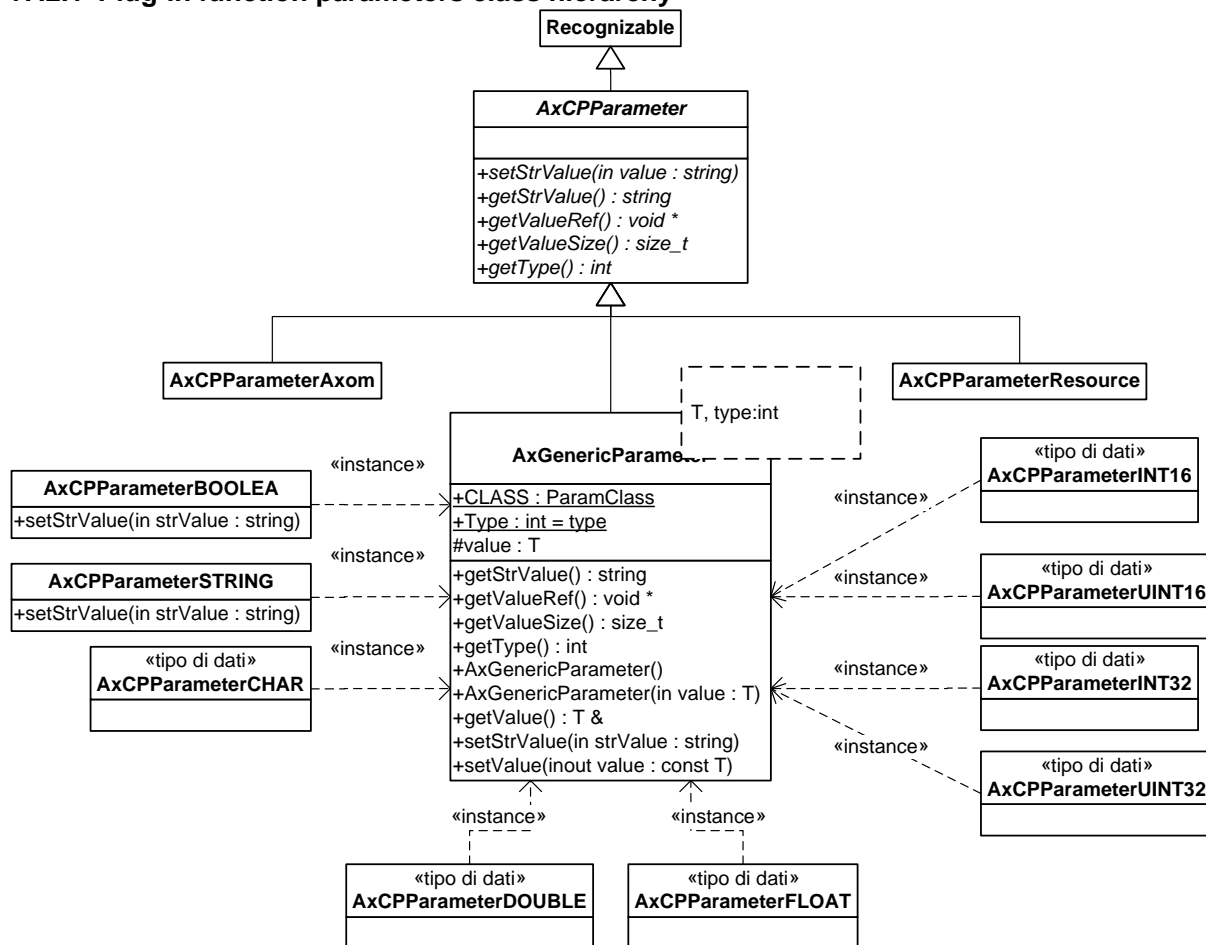


As shown in the class diagram, the Content Processing module is composed by the following classes:

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

- *AxCPPPluginInstance* – it is an extension of *AxPluginInstance*. An instance of this class represents a loaded content processing plug-in. This class allows exploiting specific functions of this kind of plug-ins (e.g. create and release Content Processing functions).
- *AxCPPFunction* – it represent Content Processing Function. It provides method: to set function parameters; to execute the function; and to obtain the result. It checks the given parameter verifying they respect what is stated in the relate function profile.
- *AxCPPParameter* – it is the common class of all parameter classes which can be passed as argument to Content Processing Functions (see next section).
- *AxCPPPluginProfile* – it wraps *AxPluginProfile* in order to allow access to generic and specific data in the plug-in description. In particular, it provides methods to get the name of contained functions and to get the description of any of them (given the related name).
- *FPFunctionProfile* – it provides methods to access information about the related function, see section 30 for more details about function profile data.

17.2.1 Plug-in function parameters class hierarchy



For each parameter type defined in parameter description schema (see section 31), a specific class has to be used in the programming language. Those classes derive from a common base class (*AxCPPParameter*) which provides a basic interface for all possible parameter types. Moreover *AxCPPParameter* derives from *Recognizable* which is an interface which allows to perform type-related operations on the objects (e.g. something like reflection in Java and C#).

The associations among class and parameter types is depicted below:

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

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

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

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

GetPluginFunction allows creating Content Processing Function instances by providing the function name (funcName).

releasePluginFunction releases the given function instance. This function has been introduced for the function instances are created by the dynamic library and they have to be deleted by it.

17.3 Errors reported and that may occur

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

18 Module - AXOM Commands and Reporting (DSI, EPFL)

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

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

18.1 General Description of the Module

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

18.2 Module Design in terms of Classes

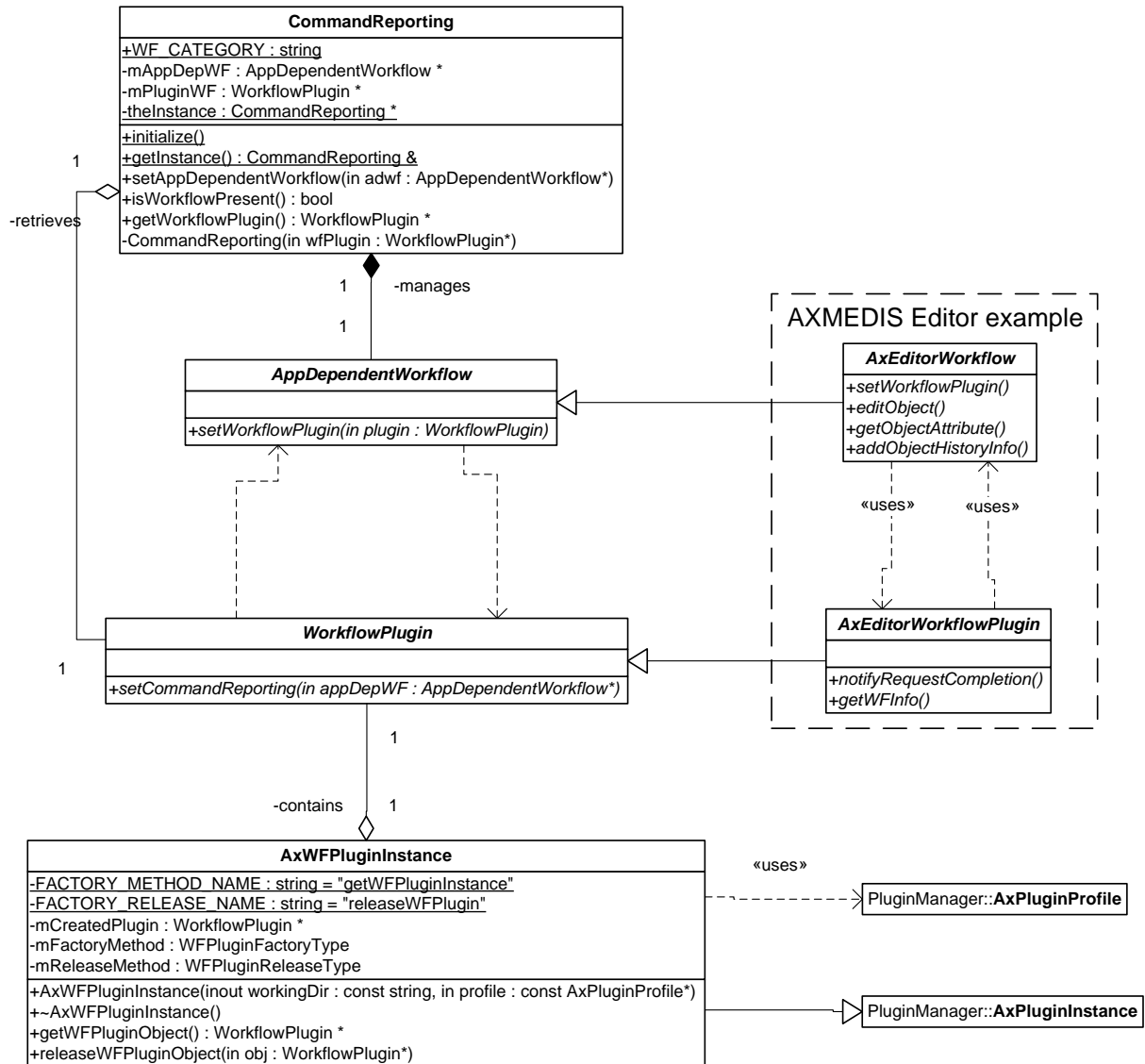
The main class in terms of exposed APIs is CommandReporting. This class implements a singleton capable of retrieving the workflow plug-in. This entity is responsible of connecting the workflow counter-parts (application and plug-in side). When it is initialized, it uses the AxPluginManager services in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

Two classes for workflow counterparts have been designed:

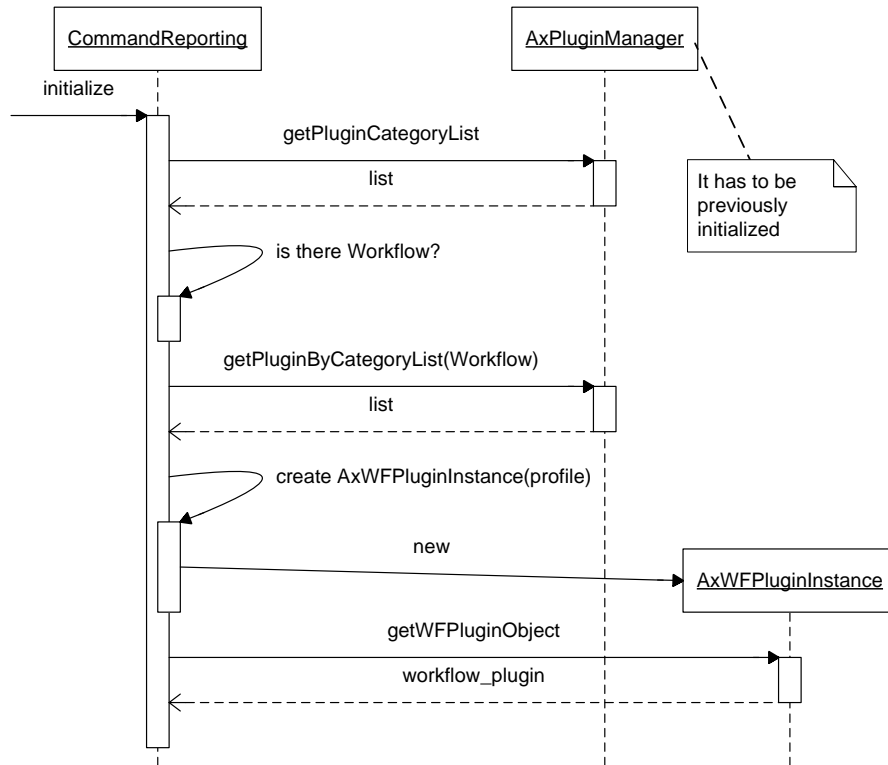
- AppDependentWorkflow: it models a generic workflow handler in the application (i.e. who is responsible of handling the request coming from the plug-in in the application context). It is an abstract class to be inherited by any AXMEDIS application. It has a general dependency to the general WorkflowPlugin.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

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

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

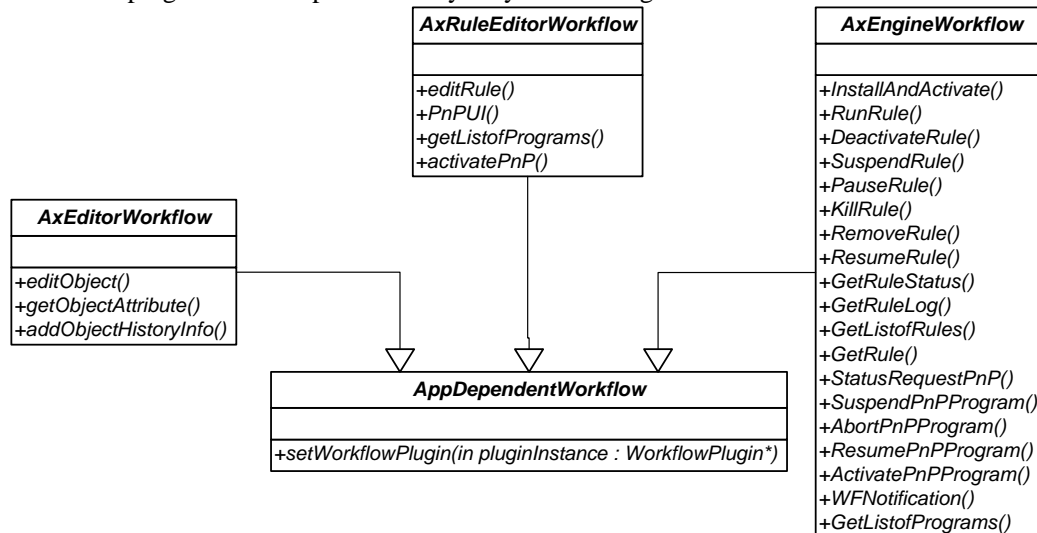


In the following diagram the initialization sequence of CommandReporting has been depicted. Please note that the AxPluginManager is invoked in order to have the list of the plugin which are present in the plugin directory.

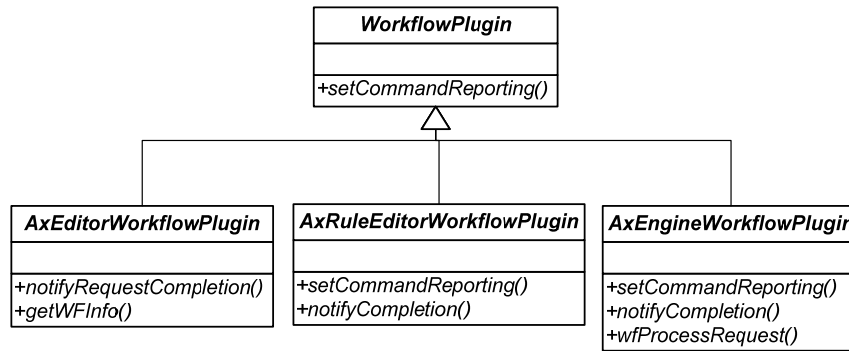


18.2.1 Different application dependent workflows interfaces

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



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



18.3 Technical and Installation information

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

18.4 Draft User Manual

see examples of usage

18.5 Examples of usage

In the following example it has been depicted an example of how to implement an editor initialization of workflow infrastructure via CommandReporting. After initialization the singleton can be obtained and the presence of a workflow plug.in can be checked.

In the last line the specific application workflow implementation is registered to the CommandReporting. After this action the workflow is ready to operate in the running application.

```

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

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

18.6 Integration and compilation issues

none

18.7 Configuration Parameters

Config parameter	Possible values

18.8 Errors reported and that may occur

Error code	Description and rationales

19 Module - Internal Audio Player (DSI)

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

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

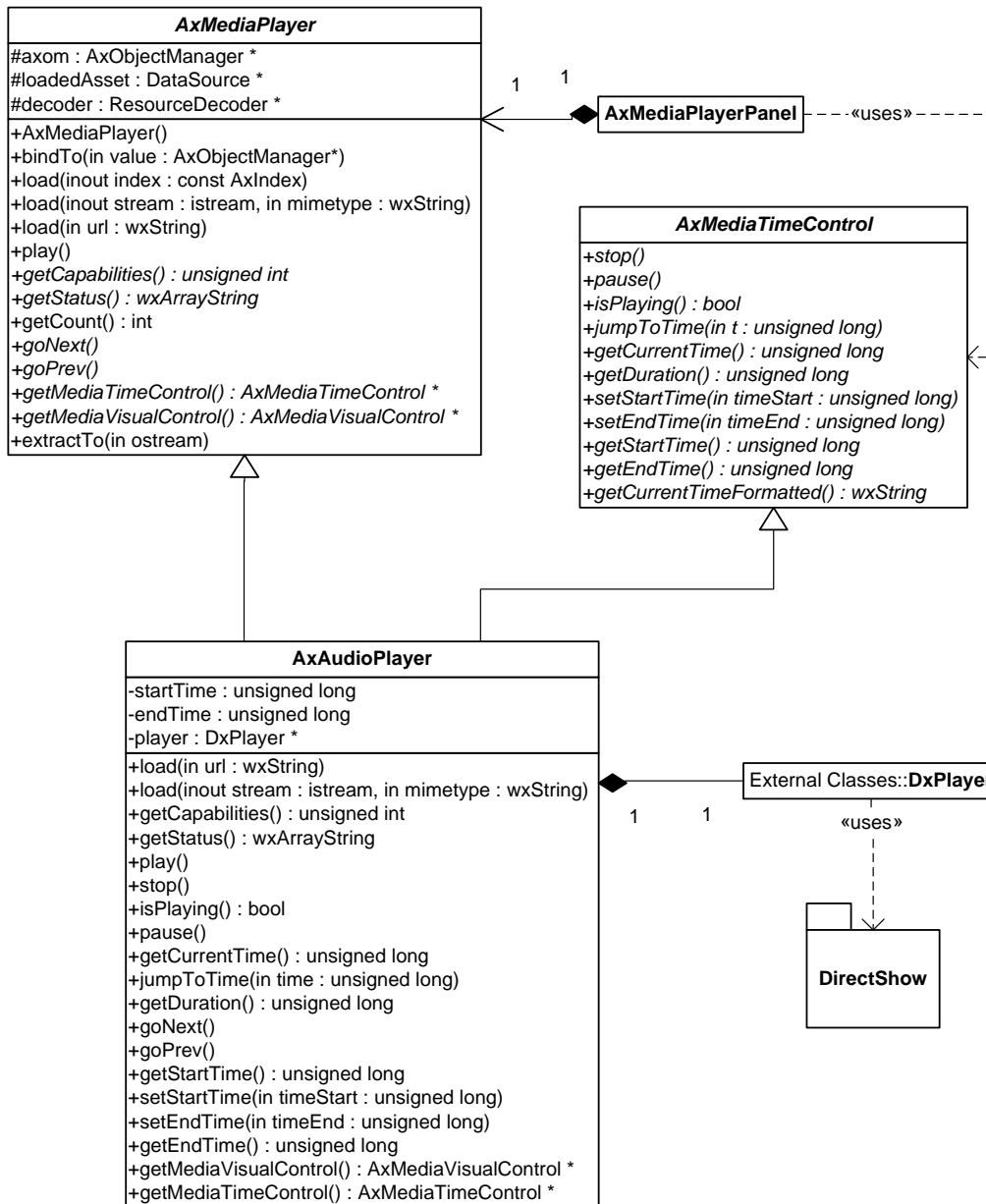
19.1 General Description of the Module

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

The functionalities supported will allow to:

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

19.2 Module Design in terms of Classes

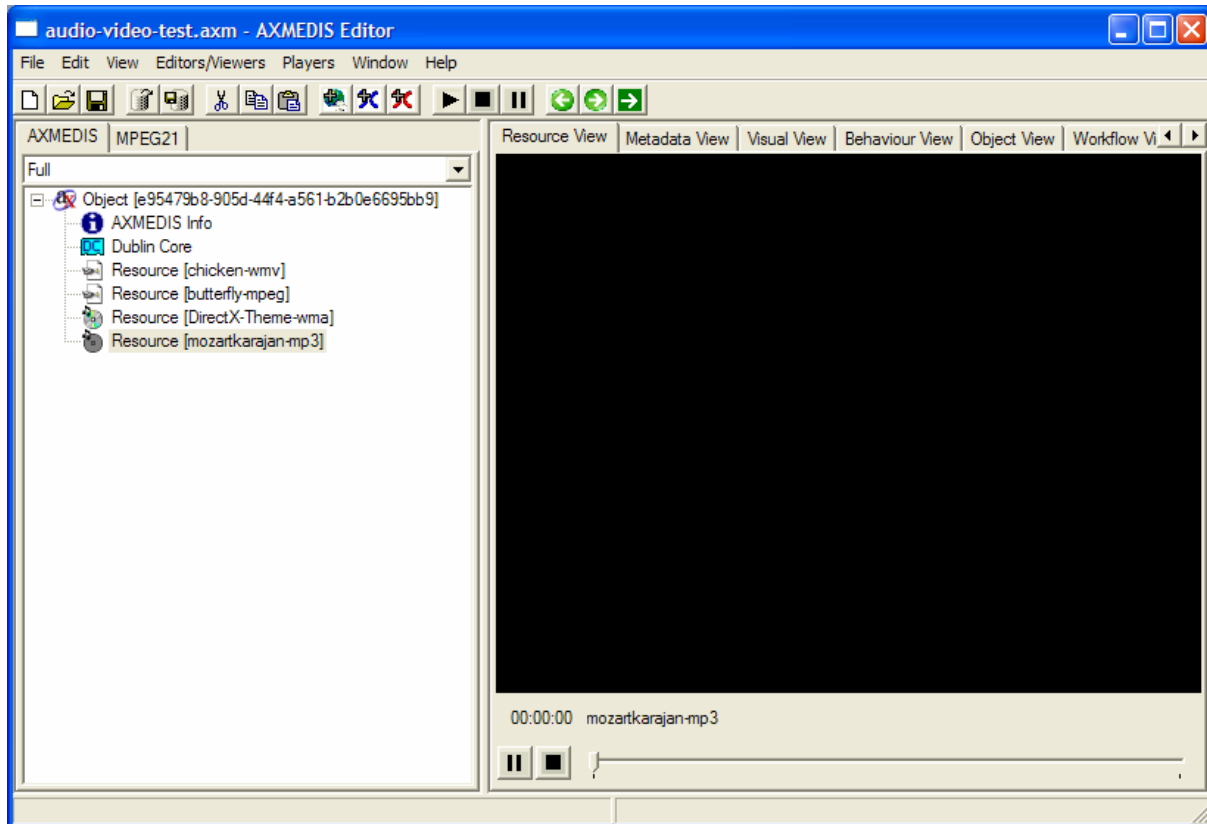


Class **AxMediaPlayerPanel** is a panel containing the basic user interface for a media player allowing the basic operations on media files (play/pause, stop, seek).

Class **AxMediaPlayer** is an abstract class containing the basic functionalities that should be implemented by a media viewer, other functionalities for time control and for visual control are in two specific interfaces (*AxMedisTimeControl*, *AxMediaVisualControl*) that the class derived from *AxMediaPlayer* may on may not implement. Class *AxMediaPlayer* is specialized in **AxAudioPlayer**, **AxVideoPlayer** and **AxImageViewer**. Class **AxAudioPlayer** implements the functionalities for audio using a library (splay) for MP3 and other code developed in WEDELMUSIC for WAV files. Another library to support AAC audio can be used. However this player may be substituted by the VideoPlayer since this one can also do audio playing only.

19.3 User interface description

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



19.4 Technical and Installation information

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

19.5 Draft User Manual

19.6 Examples of usage

19.7 Integration and compilation issues

None

19.8 Configuration Parameters

Config parameter	Possible values

19.9 Errors reported and that may occur

Error code	Description and rationales

20 Module - Internal Image Viewer (DSI)

Module/Tool Profile		
Internal Image Viewer		
Responsible Name	Bellini	
Responsible Partner	DSI	
Status (proposed/approved)	approved	
Implemented/not implemented	Implemented	
Status of the implementation	90%	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows/Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/axeditor/resource_editor	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/repos/Applications/axeditor/bin	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	yes	
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools,	Name of the communicating tools	Communication model and format

named as	References to other major components needed	(protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
	C++	wxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets	wxWidgets 2.4.2	LGPL
wxImagick		LGPL
ImageMagick		LGPL

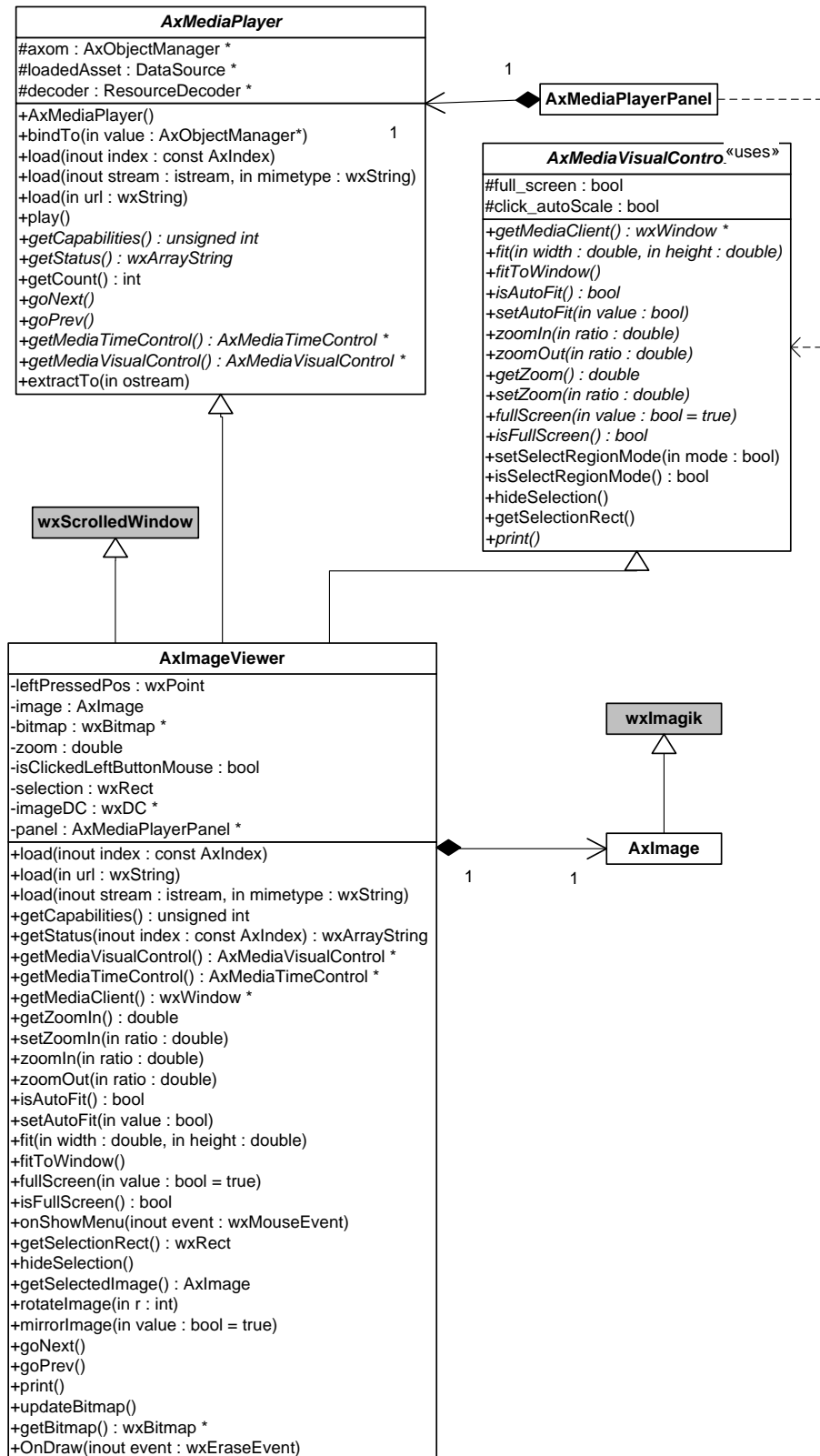
20.1 General Description of the Module

The Internal Image Viewer will allow to display images and sequences of images (like multi image TIFF format), it will support the image formats supported by the ImageMagick library.

Functionalities provided will allow to:

- zoom the image
- fit the image within a size
- select a region and save/copy it
- display the next/previous image in a sequence or display a specific one
- print an image

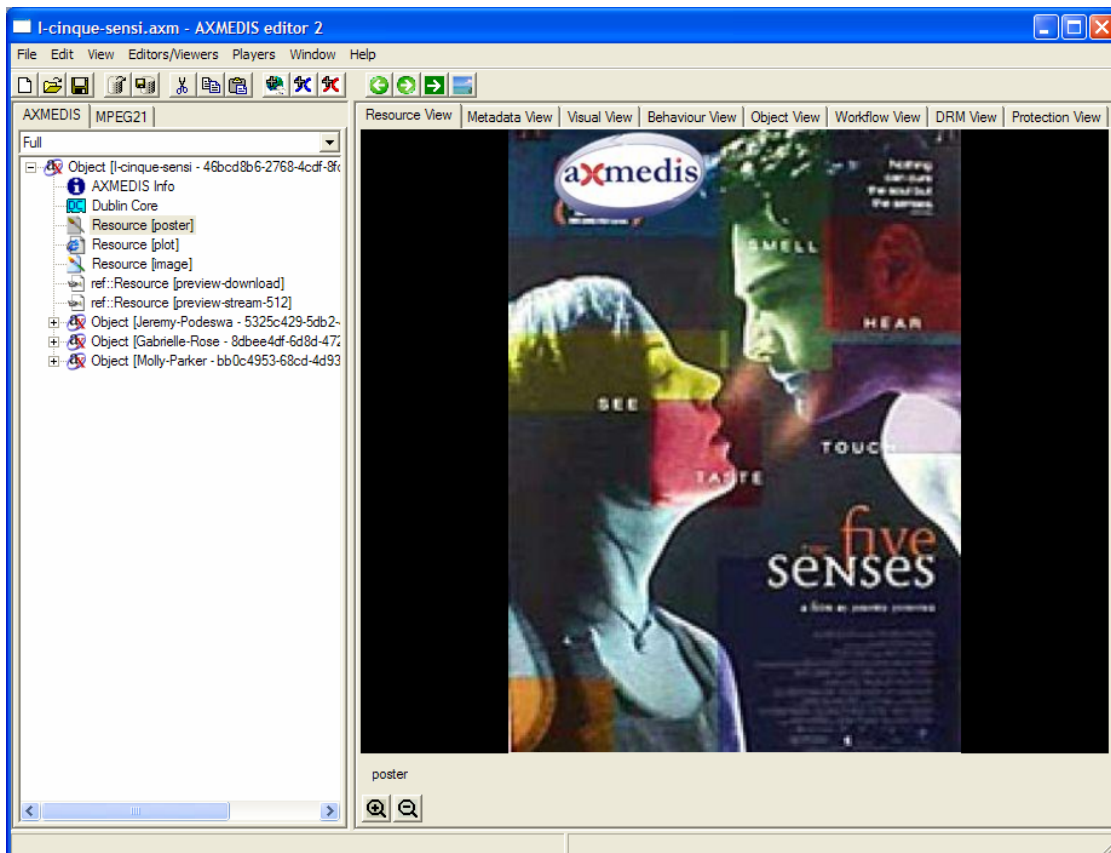
20.2 Module Design in terms of Classes



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

20.3 User interface description

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





20.4 Technical and Installation information

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

20.5 Draft User Manual

20.6 Examples of usage

20.7 Integration and compilation issues

None

20.8 Configuration Parameters

Config parameter	Possible values

20.9 Errors reported and that may occur

Error code	Description and rationales

21 Module - Internal Video Player (DSI)

Module/Tool Profile		
Internal Video Player		
Responsible Name	Bellini	
Responsible Partner	DSI	
Status (proposed/approved)	approved	
Implemented/not implemented	Implemented	
Status of the implementation	90%	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Single thread	
Language of Development	C++	
Platforms supported	windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/axeditor/resource_editor	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/repos/Applications/axeditor/bin	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location		
Usage of the AXMEDIS configuration manager (yes/no)	yes	
Usage of the AXMEDIS Error Manager (yes/no)	no	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)

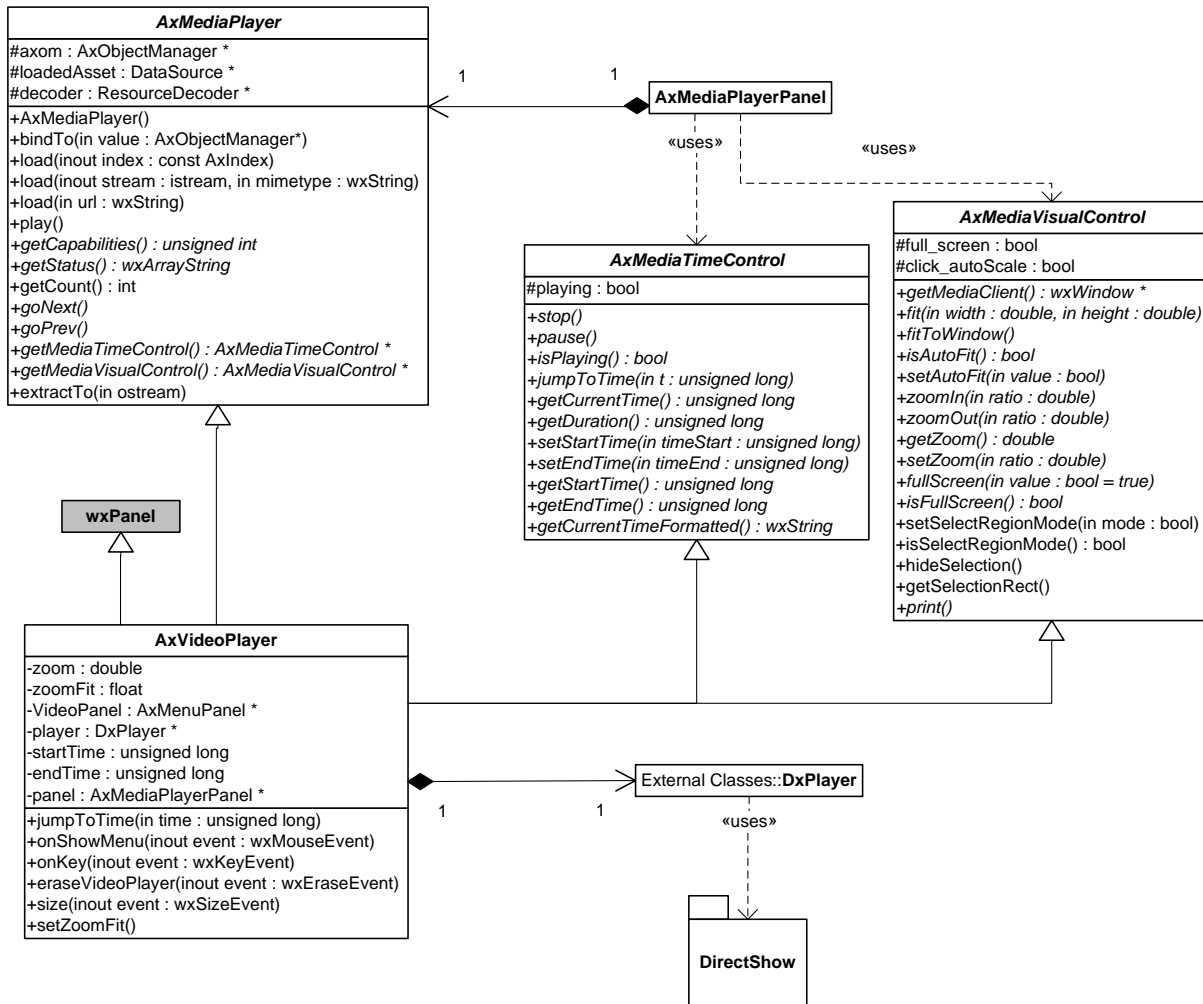
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
	C++	wxWidgets
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxWidgets	wxWidgets 2.4.2	LGPL
DirectX SDK	DirectX v.9	

21.1 General Description of the Module

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

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

21.2 Module Design in terms of Classes



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

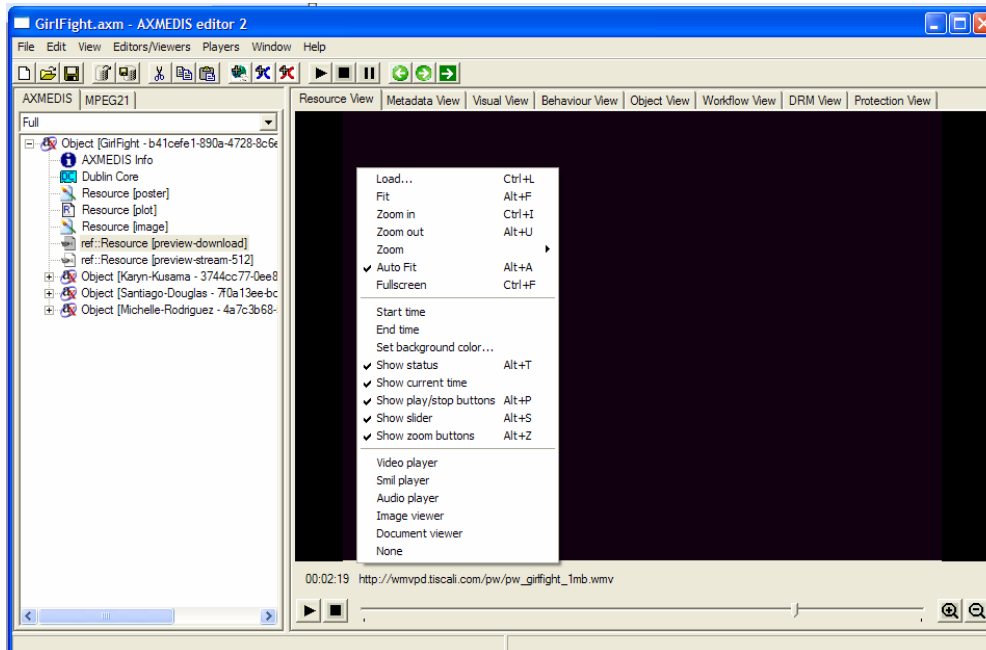
Class **AxVideoPlayer** has been implemented using **DirectShow** under Windows. For other platforms (Linux/MAC) the use of cross platform library will be investigated (like SDL – Simple DirectMedia Layer, <http://www.libsdl.org>).

The main issue is on how to access a protected video without writing it in clear as file. In DirectX it can be done writing a custom *AsyncSource* node to be used in the decoding Graph. Have to be noted that this node should not to be deployed as DLL otherwise a malicious user can build a Graph allowing to save in clear the whole video.

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

21.3 User interface description

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



21.4 Technical and Installation information

References to other major components needed	
Problems not solved	<ul style="list-style-type: none"> works only getting data from C++ streams with MP3 and MPEG video files
Configuration and execution context	

21.5 Draft User Manual

21.6 Examples of usage

21.7 Integration and compilation issues

None

21.8 Configuration Parameters

Config parameter	Possible values

21.9 Errors reported and that may occur

Error code	Description and rationales

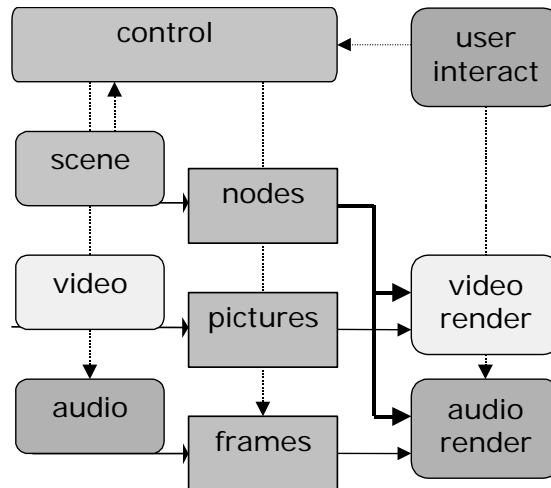
22 Module - Internal MPEG4 Player (EPFL)

Module/Tool Profile		
Internal MPEG4 Player		
Responsible Name	Antonio Romeo	
Responsible Partner	EPFL	
Status (proposed/approved)	Approved	
Implemented/not implemented	Partially implemented	
Status of the implementation	50%	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/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
mp4 (MPEG-4 File Format)		

Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
DirectX / DirectSound		MS Visual C++ (6, .NET 2002)
OpenGL		
OpenAL		
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
<ul style="list-style-type: none"> • faac (L-GPL) • im1_dmif_mp4 • im1_dmif_trif • im1_dmif_remote • im1_dmifclientfilter other players to be negotiated or changed (see below)	MPEG-4 r.s. MPEG-4 r.s. MPEG-4 r.s. MPEG-4 r.s.	L-GPL ISO ISO ISO ISO
	r.s. = reference software	
MPEG-4 Player		The OSMO player contained into Gpac 0.4.0 has been chosen as MPEG-4 Systems implementation.

22.1 General Description of the Module

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



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

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

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

Currently the MPEG-4 Player can allow two working modes:

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

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

22.2 Integration of MPEG-4 IPMP eXtensions into the MPEG-4 Player

The ENTHRONE terminal will be integrated with an implementation of ISO/IEC 14996 part 13 (IPMP eXtensions – IPMP-X). Such extensions standardize a messaging interface for the insertion of IPMP tools at each of the three MPEG-4 Systems control points.

To facilitate the cooperation of multiple tools in the protection and governance of content, a message based architecture is provided in IPMP-X. IPMP Tools communicate with each other and with the terminal by using the standard messages defined in the IPMP-X specifications. However, IPMP-X does not specify the interface to transfer messages. Therefore such an interface will be defined in an implementation-specific format.

By exchanging standard messages different IPMP-X tools can communicate and concurrently protect the same MPEG stream. At the same time the MPEG Systems layer can use the messaging interface to interact with the installed IPMP-X tools. This is mainly necessary to allow the exchange of information needed during components' authentication.

The MPEG-4 Systems architecture integrated with the IPMP-X is shown in Figure 3.

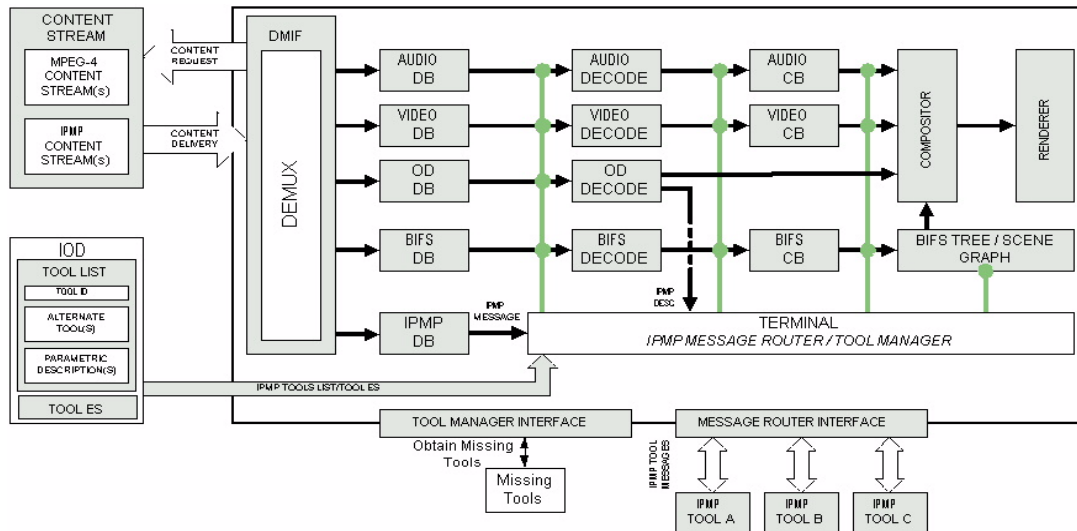


Figure 3 - Mapping of IPMP Extensions to MPEG-4 Systems Architecture

The components to be implemented or integrated in order to enable the MPEG-4 player with the IPMP-X will be:

- The IPMP-X messaging library.
To facilitate the cooperation of multiple tools in the protection and governance of content, a message based architecture is provided in IPMP-X. IPMP Tools communicate with each other and the terminal by using the standard messages defined in IPMP-X specifications.
- The IPMP-X message router (MR) and the MR interface.
The MR is a conceptual entity within the Terminal that implements the Terminal-side behavior of the Terminal-Tool interface.
- The IPMP-X tool manager (TM) and the TM interface.
The TM is a conceptual entity within the Terminal that processes IPMP Tool List(s) and retrieves the Tools that are specified therein.
- The IPMP-X compliant tools to be plugged at the given control points.

IPMP-X Tools are modules that perform (one or more) IPMP functions such as authentication, decryption, watermarking, etc. It can range from being simple processors to coordinating a set of IPMP modules.

22.2.1 Architectural elements of the IPMP-X framework

Figure 4 shows the main architectural conceptual entities in the IPMP Extension framework: the Message Router and the Tool Manager, and describes the interfaces between these and the MPEG-4 Terminal, from one side, and the IPMP Tools, from the other.

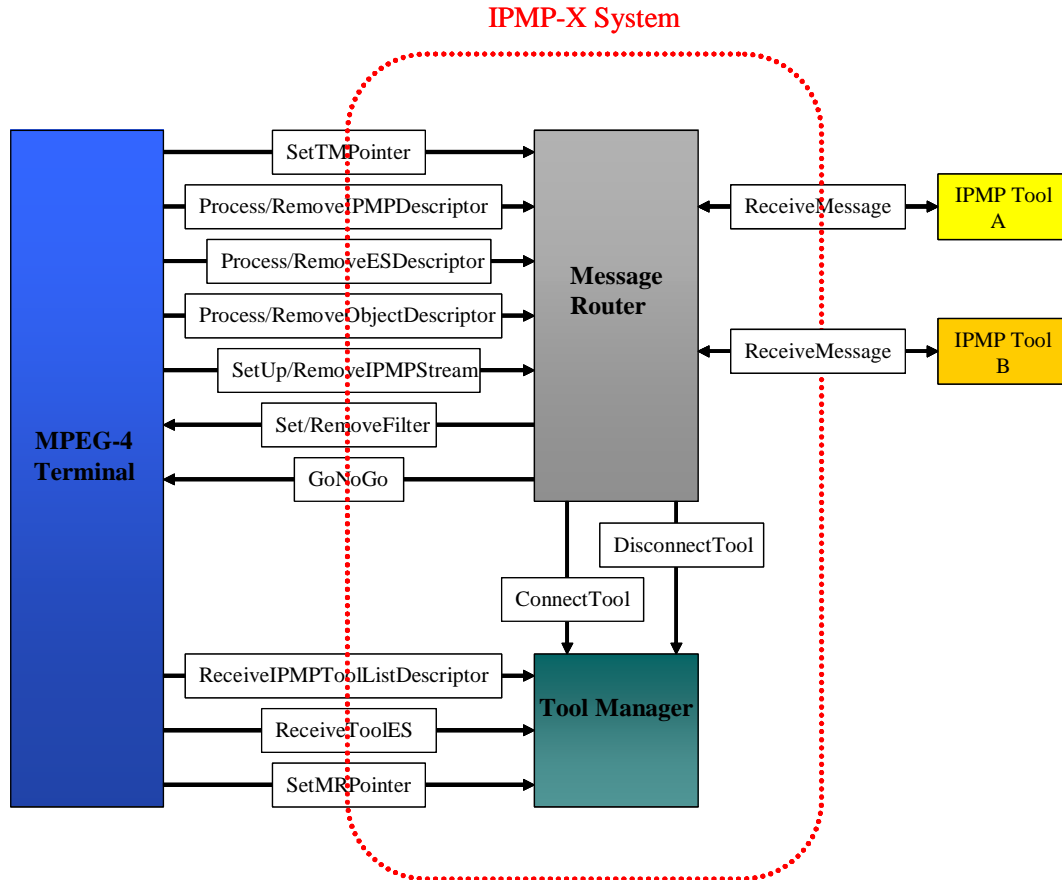


Figure 4 - MPEG-4 IPMP Extension Sample Architectural Diagram

The Message Router

The main interactions the Message Router has with the Terminal are the processing of Object Descriptors, Elementary Stream Descriptors and IPMP Descriptors. If the Message Router notices that an ES is protected, a request is sent to the Tool manager to instantiate an IPMP Tool in the specified Control Point. The MR performs this operation by calling the *SetFilter* method of the protected ES, inserting the requested IPMP Tool in the Control Point. The MR is also in charge of retrieving data from the *IPMPStream* in case this is present, and send those data to the IPMP Tool specified as the recipient of such information.

From the IPMP Tool side, the Message Router is the core of the message-based architecture. Physical routing of information and context resolution are handled by this conceptual entity, which enables the abstraction from platform-dependent routing and delivery issues. The interface between the Message Router and the IPMP Tools is non-normative, though a Registration Authority is in charge of defining the implemented API for an IPMP Tool and Terminal communication on a given platform. Communication between two IPMP Tools not performed through the Message Router is outside the scope of the standard.

The delivery of both bit stream sourced IPMP information as well as IPMP Tool and Terminal generated information is supported through the use of three separate messages exchanged between the Message Router and IPMP Tools. These are: *IPMP_MessageFromBitstream*, which is used to deliver IPMP stream data to an IPMP Tool, *IPMP_DescriptorFromBitstream*, which is used to deliver *IPMP_Descriptors* [1], and *IPMP_MessageFromTool*, which is used to deliver messages from either other IPMP tools or the Terminal itself. The latter, in particular, is a container of *IPMP_Data_BaseClass* messages. [2] Defines a set of messages extending *IPMP_Data_BaseClass*, which allow a normative communication between IPMP Tools; this is explained in more detail in section 0.

The Tool Manager

The Tool Manager is another conceptual entity within the IPMP-X Terminal which basically is responsible for receiving the *IPMP_ToolListDescriptor* from the MPEG-4 content the user wants to render, instantiating IPMP Tools and managing them during all their lifetime. Figure 5 shows the class *IPMP_ToolListDescriptor* that contains up to 255 IPMP Tool classes (Figure 6).

```
class IPMP_ToolListDescriptor extends BaseDescriptor :
    bit(8) tag= IPMP_ToolsListDescrTag
{
    IPMP_Tool ipmpTool[0 .. 255];
}
```

Figure 5 - The class *IPMP_ToolListDescriptor*

The necessary information required to instantiate all the specified IPMP Tool is then extracted from each class; the corresponding software module is then retrieved. The IPMP Tools (or their alternate, if any) if not present yet on the Terminal, could be downloaded from a remote location. In the *IPMP_Tool* class, in fact, a one of the fields may contain a location from which the Tool could be retrieved; however, the Tool downloading functionality of the TM is not normatively defined in [2]. IPMP Tools can also be carried in a particular Stream, the *IPMPToolStream*.

After the instantiation of a Tool, the TM assigns a unique 32-bit identifier (ContextID) to each instance of IPMP Tool. Such ID may be used to indicate the Message Router the intended recipient of a message, and to reference in an unambiguous way an instance of Tool.

IPMP Tools

Tools can range from being simple processors of content such as performers of decryption algorithms on encrypted content, watermarking/fingerprinting inserters or detectors, rights managers; some may also have co-ordinating functions over a set of other Tools.

The Tool Manager receives from the class *IPMP_Tool* (Figure 6) the required information to instantiate the corresponding software module. Each Tool is characterized by the 128-bits *IPMP_ToolID*, and other additional information: it may have alternates Tools to substitute it without any loss of functionality, it may be configured to operate according to a parametric description, and may be downloaded from a remote location which address is contained in the field *ToolURL*. Even though it is not normative, this functionality may be provided to the Tool Manager module.

```
AX class IPMP_Tool extends BaseDescriptor :
    bit(8) tag= IPMP_ToolTag
{
    bit(128) IPMP_ToolID;
    bit(1) isAltGroup;
    bit(1) isParametric;
    const bit(6) reserved=0b0000.00;

    if(isAltGroup){
```

Figure 6 - The class IPMP_Tool

The IPMP-X specification [2] defines a set of messages by means of which the IPMP Tools can exchange information among them or with the Terminal in a normative way, as described in the following chapter; their delivery is performed by the Message Router, which handler must be supplied to each Tool during an initialization phase. A Registration Authority is in charge of defining this process, as well as the implemented API for the Tool-Message Router communication on a given platform. This allows the data exchange between IPMP Tools, even if not developed by the same provider. Through a normative message exchange, moreover, mutual authentication may occur among IPMP Tool or between them and the Terminal, in order to increase the overall security of the system.

Normative messages

To facilitate the cooperation of multiple tools in the protection and governance of content, a set of messages has been standardized, able to cover many scenarios in which IPMP Tools may be involved. Tools requiring exchanging data with other entities need to have the capability to generate and interpret some of them.

By means of normative messages, a Tool may require, for instance, to be informed of all the others IPMP Tools next to it (*IPMP_GetTools* message) or to be notified when a new IPMP Tool is instantiated (*IPMP_AddToolNotificationListener* message). If a particular event occurs, this can be signalled by an *IPMP_NotifyToolEvent* message; moreover, a Tool can require another Tool to be instantiated by the means of an *IPMP_ConnectTool* message, while the *IPMP_DisconnectTool* triggers the dual action. Several messages have been defined for common IPMP processing: *IPMP_OpaqueData* for the carriage of user defined data, *IPMP_RightsData* for the carriage of rights expressions, *IPMP_KeyData* for the carriage of decryption key data as well as timing information to determine the validity period of time varying keys. User defined data can be delivered by means of an *IPMP_OpaqueData* message, while a set of messages can be used to instruct a watermarking/fingerprinting Tool on the operation to perform, or receive from it the data extracted from the content. Two other messages: *IPMP_InitMutualAuthentication* and *IPMP_MutualAuthentication* are defined for requesting and carry on a mutual authentication message protocol.

The actual implementation of the messages just briefly introduced in this document will be further specified during the following development and integration of the IPMP-X in the MPEG-4 player.

The AXMEDIS IPMP-X Interface

Table I provides a schematic of the interfaces between the MPEG-4 systems implementation running on the OSMO MPEG-4 player and the IPMP-X classes and methods. Such an interface is defined here in terms of C++ classes and methods with related inheritances, some modifications can be applied in course of integration if technical constraints will require it.

Class	Methods	Inherits from
IPMPSystem	CreateIPMPServices RemoveIPMPServices	
IPMPTool_Interface	ReceiveMessage	
MR_Interface	SetTMPinter ProcessObjectDescriptor RemoveObjectDescriptor ProcessIPMPToolDescriptor RemoveIPMPToolDescriptor ProcessESDescriptor RemoveESDescriptor SetUpIPMPStream RemoveIPMPStream	IPMPTool_Interface
TM_Interface	SetMRPointer DisconnectTool ConnectTool ReceiveToolES RemoveToolES ReceiveIPMP_ToolListDescriptor	
IPMPServices		TM_Interface MR_Interface

Table I – The AXMEDIS IPMP-X interface

22.2.2 Examples of scenarios to be demonstrated.

This section provides an example of the architecture of a possible scenario to be shown as demonstration of an IPMP-X powered MPEG-4 player. It is provided here only by way of example and not by way of limitation.

The demonstrators will have the ability to load, instantiate, insert, start and run multiple independent IPMP tools. These tools can be at control points 1, 2 or 3 (the post CB control point does not appear to be supported in the MPEG-4 reference software for Systems, but it should be supported in the MPEG-4 Systems implementation provided by OSMO) and multiple tools can be used at each control point.

Some of the scenarios to be demonstrated can comprise:

G723	Simple audio
H263_G723	Audio and video
H263_G723PROT	Audio, with post decoder IPMPTool, and video

H263_G723PROT2	Audio, with 2 pre decoder IPMPTools, and video
H263_G723PROT3	Audio, with 2 post decoder IPMPTools, and video, with pre decoder IPMPTool
AAC	Audio
AACPROT	Audio, with post decoder IPMPTool
AACPROTWM	Watermarked Audio, with post decoder CRL Watermark decoder IPMPTool

Figure 7 below shows the positions of each IPMPTool for the example bitstream called above H263_G723PROT3.

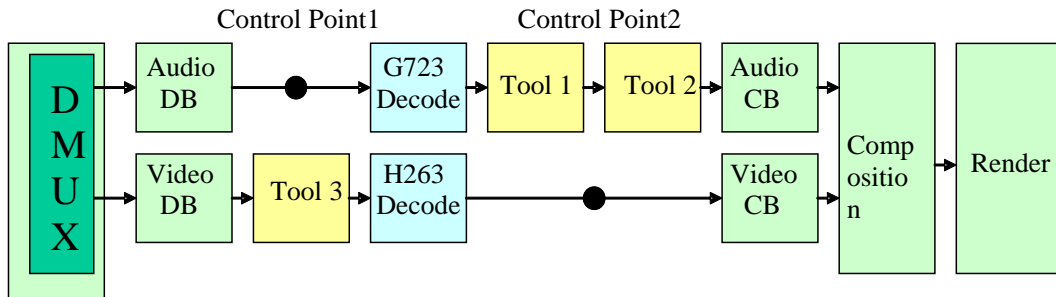


Figure 7 – Architecture of the Systems layer integrated with 3 IPMP-X tools. The audio stream is processed by 2 post decoder IPMP-X Tools, and the video by 1 pre-decoder IPMP-X Tool.

22.3 Translation of MPEG-4 IPMPX binary descriptors to XML based MPEG-21 IPMP Components

22.3.1 DMP Content Information overview

In DMP content is a combination of resources, metadata, content and rights expressions. DMP calls “Represent Content” the set of tools used to provide a digital representation that can be processed by a compliant device. DMP calls such a content representation *DMP Content Information (DCI)*. DCI provides the means to convey identifiers, associate information and metadata and associate information with governed content. DCI is an extended profile of MPEG-21 Digital Item Declaration (ISO/IEC 21000-2) and IPMP Components (ISO/IEC 21000-4).

Concerning content governance representation, the DMP Rights Expression Language is an extended combination of 3 MPEG-21 REL (ISO/IEC 21000-5) Profiles: Core, Standard Extension and Multimedia Extensions.

DMP Content can be identified by means of Identifiers that conform to the Uniform Resource Names (URN) scheme. This is based on MPEG-21 Digital Item Identification (ISO/IEC 21000-3).

22.3.2 MPEG-4 IPMPX descriptors

The IPMPX messages defined in ISO/IEC 14496-13:2004 specify a normative syntax for communication between IPMP Tools (modules performing encryption, watermarking, key management, rights parsing, etc.) and between them and the device on which they operate. There are several types of messages serving different purposes:

- Mutual Authentication: messages used by two parties to agree on a specific algorithm to authenticate each other and perform mutual authentication, as well as sharing a common secret for secure communication
- IPMP tool acquisition: messages to convey an IPMP Tool and to signal its type

- IPMP Tool connection and disconnection: messages to request the list of tools available, to query a given IPMP tool as to its capabilities and functionality, to request the connection of another IPMP tool, etc...
- Notification of IPMP Tool events: like signalling the instantiation of an IPMP Tool, to notify IPMP tools when other IPMP tools have either been connected, disconnected or processed watermark information, etc;
- Common IPMP processing: messages for carriage of keys, rights expressions, watermarking data, etc...
- IPMP tool to/from User interaction: to normatively specify the interaction between IPMP tools and Users.

22.3.3 IPMPX translation to XML

The goal of the DMP proposal to translate binary IPMPX descriptors to XML is twofold. On one side it provides a representation of IPMPX descriptors which is more easily readable, possibly extensible and editable in an automatic or semi-automatic way through existent toolset for XML authoring, on the other side it allows its adoption in MPEG-21 IPMP Components.

The conversion process performed by DMP has followed two main guidelines:

- Keeping the tightest possible relation between names/semantics of the variables and elements in the two representations.
- Using well-known elements to replace ad-hoc syntax defined in IPMPX spec.

For example:

- **bit(128) toolID** was translated in **xsd:anyURI**
- **ByteArray keyBody** was translated in **dsig:KeyInfo**

In order to achieve interoperability when implementing IPMP System adopting MPEG-4 IPMPX it is very important to know the characteristics of the involved devices. For this reason DMP has adopted and profiled the **mpeg4ipmp:TerminalID** namespace as shown in the figure below.

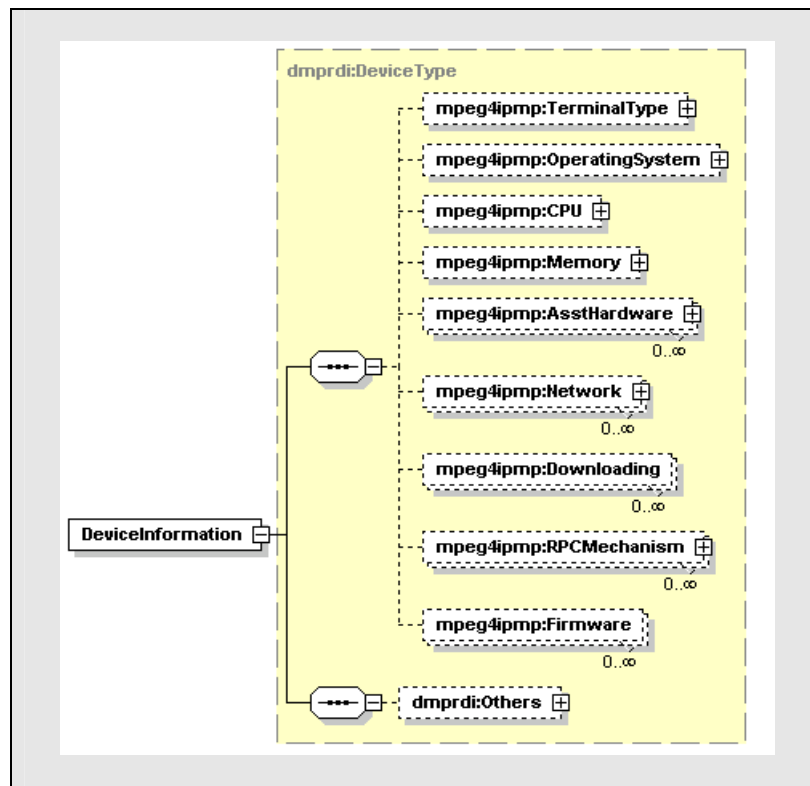


Figure 8 - The mpeg4ipmp:TerminalID namespace

In the **mpeg4ipmp:TerminalID** namespace profiled by DMP all elements are optional and it has been kept extensible so as to support the largest extent of devices characteristics.

Thanks to the translation of the binary MPEG-4 IPMP Descriptors to XML their adoption in MPEG-21 IPMP Components will be possible. As it is shown in the figure below, the IPMP Tool List Descriptor, IPMP Tool Descriptor and IPMP Descriptor used in MPEG-4 Streams to link the governed resources with the governing tools, can be embedded in a MPEG-21 Digital Item in accordance with the MPEG-21 IPMP Components schemas.

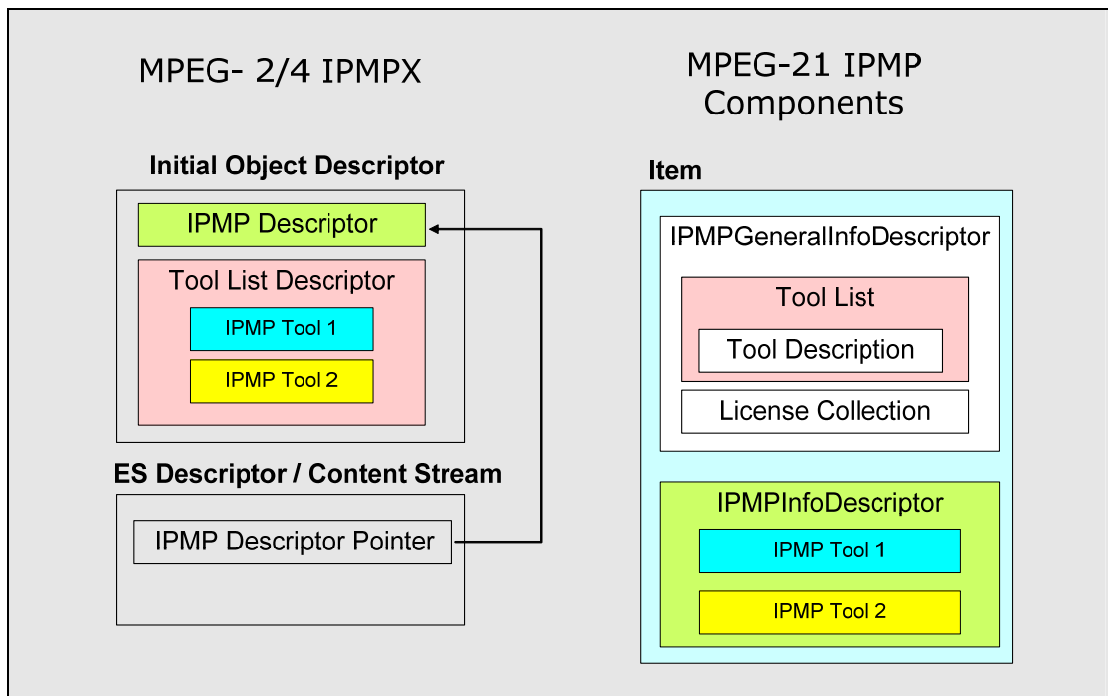


Figure 9 - Mapping from an MPEG-4 IPMPX IOD to an MPEG-21 DI with governed MPEG-4 resources.

In AXMEDIS this can be implemented by extending the current specification of the AXMEDIS player for the support of MPEG-21 IPMP Components and therefore MPEG-4 IPMPX resources carried in AXMEDIS objects. With the integration of the IPMPX interfaces in OSMO the support of MPEG-4 IPMPX will be present only at the level of pure MPEG-4 media, with this extension it would also be supported for resources carried in MPEG-21 compliant AXMEDIS objects.

The following pictures better detail how the MPEG-4 IPMPX descriptors used to finely specify the configuration of IPMPX tools and to let them communicate by means of a messaging infrastructure, can be simply mapped to the schemes specified in the MPEG-21 IPMP Components specification.

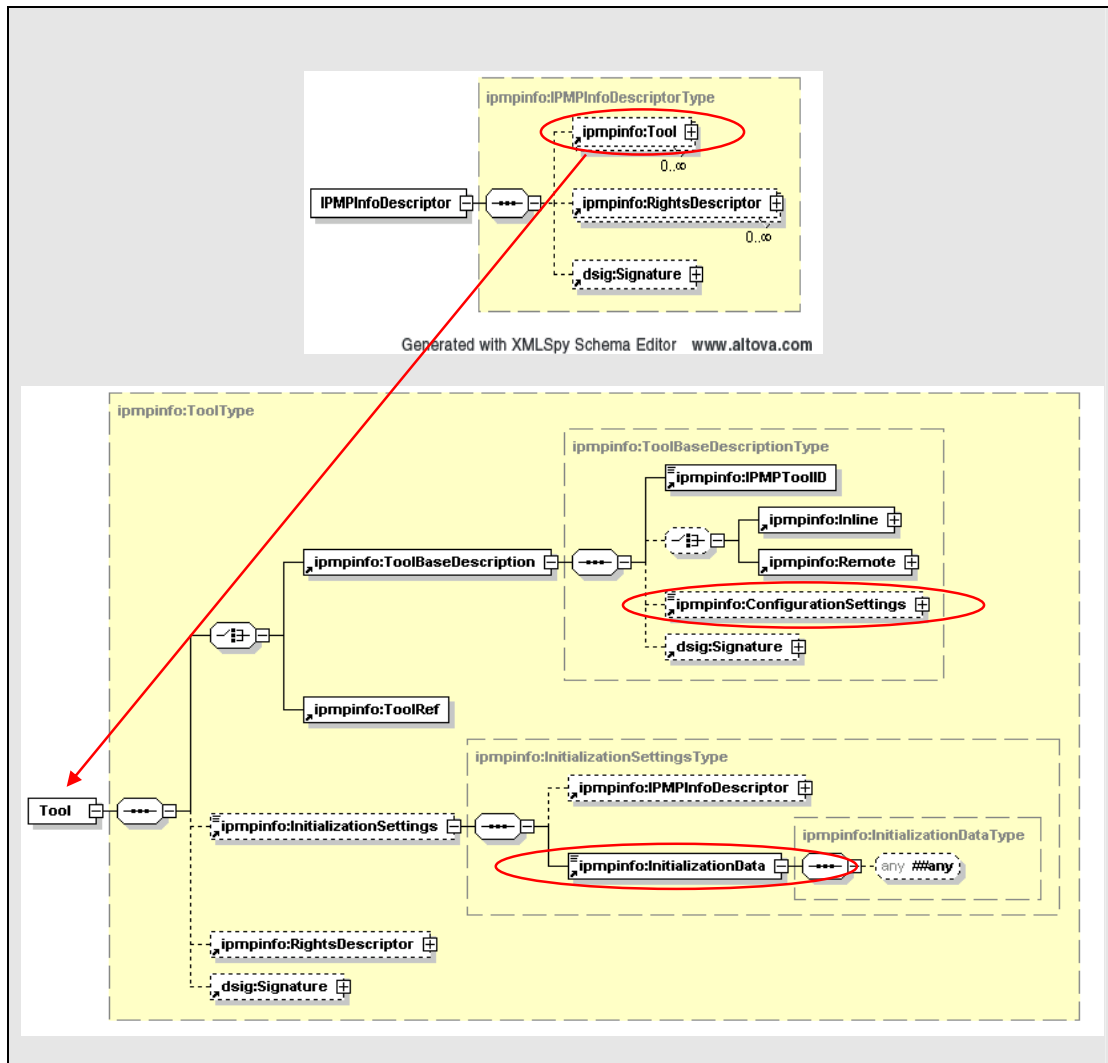


Figure 10 - The MPEG-21 IPMPInfoDescriptor and IPMPTool schemes

A typical set of IPMPX descriptors and messages to be supported in the process of governed MPEG-4 resources decoding is shown in the figure below. Messages used to exchange information among IPMPX tools and between a given tool and the terminal are highlighted in blue.

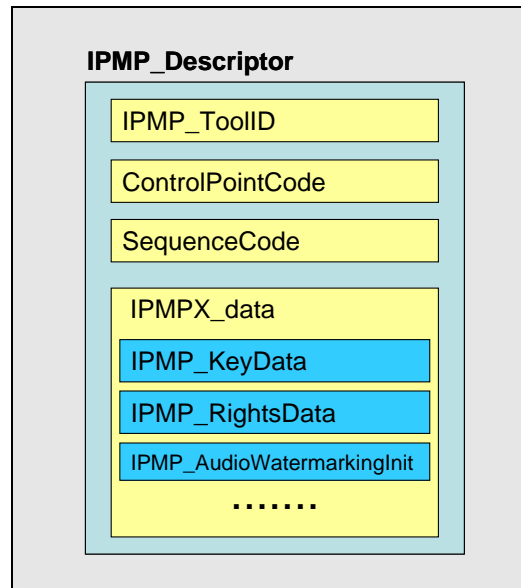


Figure 11 - Typical structure of an MPEG-4 IPMPX descriptor carrying IPMPX messages.

Starting from the already defined **ipmpinfo:Tool** schema, DMP has profiled the two elements highlighted in the figure above: **ipmpinfo:InitializationData** and **ipmpinfo:ConfigurationSettings**.

Messages and descriptors used to specify the way in which a tool has to be plugged in the MPEG-4 Systems architecture have been profiled in the **ipmpinfo:InitializationData** as depicted below. These descriptors include the control point to which a given tool has to be connected, messages governing the connection and disconnection of a tool and initialization parameter for content processing algorithms such as watermarking.

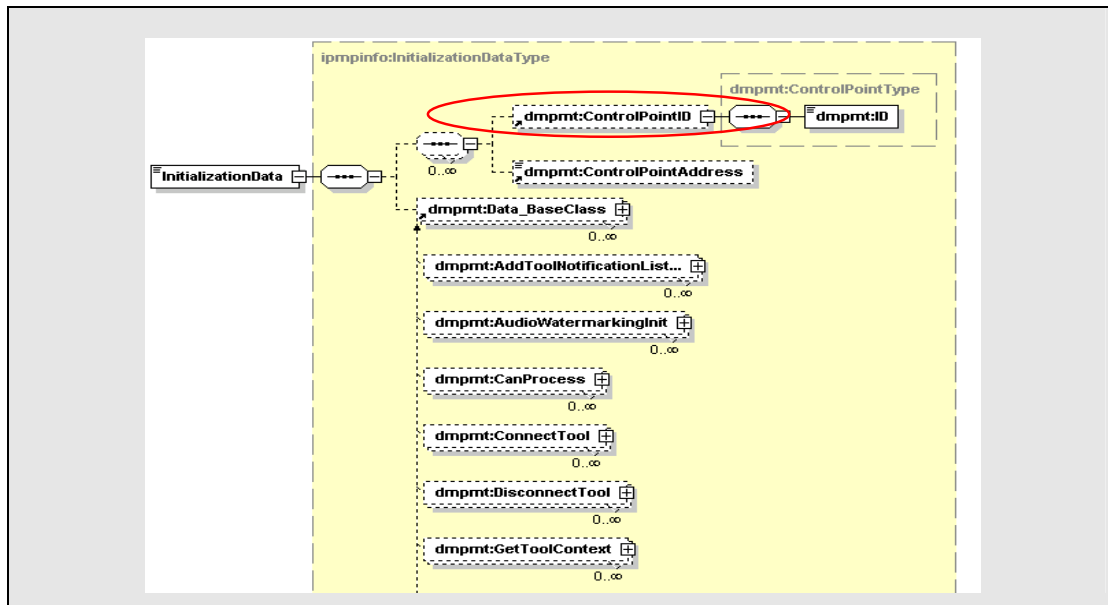


Figure 12 - Restriction to ipmpinfo:InitializationData

On the other hand, the restriction to **ipmpinfo:ConfigurationSettings** depicted in the figure below permitted to specify the supported platforms by means of **ipmpinfo:SupportedPlatform** namespace. This can be adopted in AXMEDIS to describe the characteristics of those terminals supporting the specified IPMP system and to require content adaptation/transcoding in case the concerned terminal is not among the supported platforms. This may be the case when a given cryptographic processor is not present and a different encryption algorithm needs to be used to protect the selected resource.

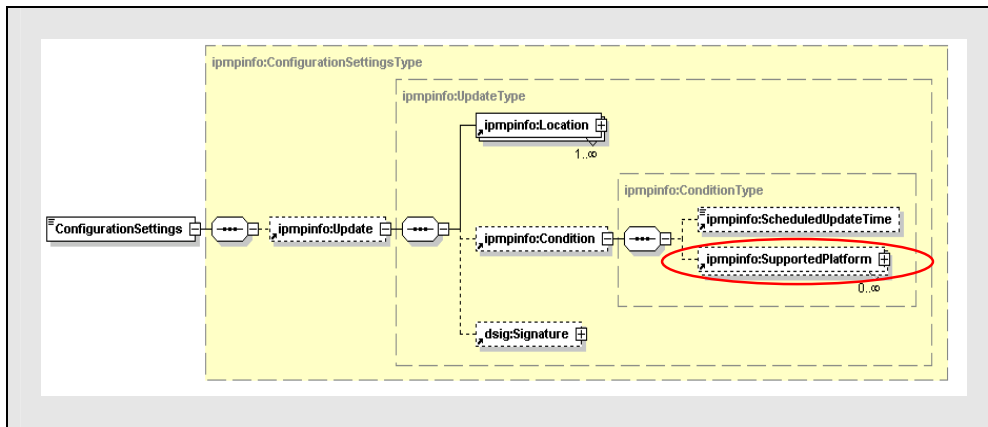


Figure 13 - Restriction to ipmpinfo:ConfigurationSettings

The figure below shows the restriction to the MPEG-21 **ipmpinfo:SupportedPlatforms** that can be used to describe the AXMEDIS player in details. The optional children of **dmpdr:DeviceInformation** can be used to finely profile the content to be delivered to the end device. In the AXMEDIS chain such information could be used in order to allow the adaptation of the required content and possibly to periodically update it according to (software and/or hardware) updates of the terminal platform.

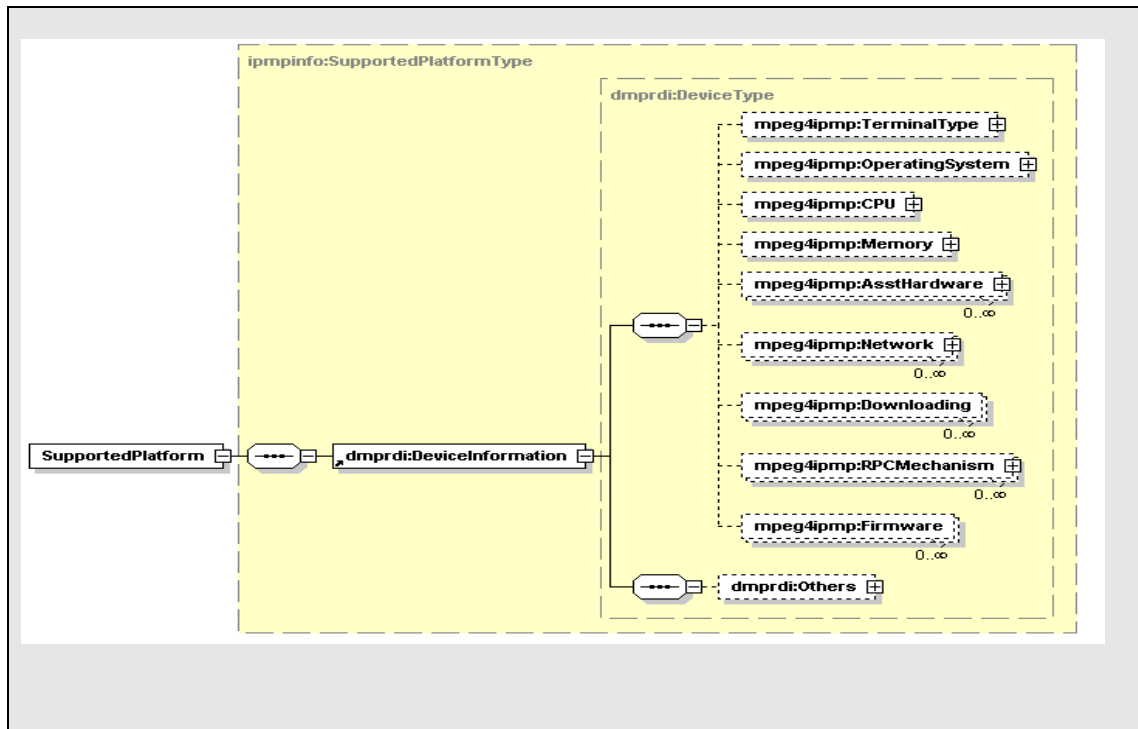


Figure 14 - Restriction to ipmpinfo:SupportedPlatforms

An example of adoption of the schema represented above to describe the AXMEDIS player is given below. Many can be the use of the defined namespaces in the framework of the AXMEDIS use cases. For example, some of the values of children of **ipmpinfo:TerminalID** can be used by the license servers to calculate access keys to be embedded into licenses. The variety of data carried by such namespaces will provide a high complexity and flexibility to the license management mechanisms adopted.

```
<ipmpinfo:Tool>
  <ipmpinfo:ToolBaseDescription>
```

```

<ipmpinfo:IPMPToolID>urn:mpegRA:mpeg21:IPMP:ABC002:56:79</ipmpinfo:IPMPToolID>
<ipmpinfo:Remote ref="urn:AXMEDISIPMPToolsServer:ToolPartEnc002-9090-v.1.0" />
<ipmpinfo:ConfigurationSettings>
  <ipmpinfo: Configuration>...</ipmpinfo: Configuration>
  <ipmpinfo:Update>
    <ipmpinfo:Location ref="urn:IPMPToolsUpdatingServer1:ToolPartEnc002-9090-
NewVersion" />
    <ipmpinfo:ScheduledUpdateTime period="P1D">2005-03-07T00:00:00
  </ipmpinfo:ScheduledUpdateTime>
    <ipmpinfo:SupportedPlatform xmlns:mpeg4ipmp="urn:mpeg:mpeg4:IPMPSchema:2002">
      <mpeg4ipmp:TerminalID>
      <mpeg4ipmp:TerminalType>
        <mpeg4ipmp:Vendor>EPFL</mpeg4ipmp:Vendor>
        <mpeg4ipmp:Model>OSMO0.4.0</mpeg4ipmp:Model>
      </mpeg4ipmp:TerminalType>
      <mpeg4ipmp:OperatingSystem>
        <mpeg4ipmp:Vendor>Microsoft Corporation</mpeg4ipmp:Vendor>
        <mpeg4ipmp:Model>Windows XP Professional</mpeg4ipmp:Model>
        <mpeg4ipmp:Version>XP SP2</mpeg4ipmp:Version>
      </mpeg4ipmp:OperatingSystem>
      <mpeg4ipmp:Vendor>Intel Corporation</mpeg4ipmp:Vendor>
        <mpeg4ipmp:Model>Intel® Pentium® M Processor</mpeg4ipmp:Model>
        <mpeg4ipmp:Speed>1060</mpeg4ipmp:Speed>
      </mpeg4ipmp:CPU>
      <mpeg4ipmp:Memory>
        <mpeg4ipmp:Vendor>Kingston</mpeg4ipmp:Vendor>
        <mpeg4ipmp:Model>DDR2 SDRAM</mpeg4ipmp:Model>
        <mpeg4ipmp:Size>512</mpeg4ipmp:Size>
        <mpeg4ipmp:Speed>800</mpeg4ipmp:Speed>
      </mpeg4ipmp:Memory>
      </mpeg4ipmp:TerminalID>
    </ipmpinfo:SupportedPlatform>
  </ipmpinfo:Update>
</ipmpinfo:ConfigurationSettings>
</ipmpinfo:ToolBaseDescription>
</ipmpinfo:Tool>

```

Figure 15 - Example of AXMEDIS player representation using the MPEG-21 ipmpinfo:Tool element

22.4 Module Design in terms of Classes

A preliminary study of IPMPX integration into the OSMO architecture has been performed. It may require some adaptation due to the recent update of some of the OSMO features.

The diagram below provide the results of such preliminary study.



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

23 Module - Internal SMIL Player (EPFL)

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

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

23.1 General Description of the Module

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

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

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

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



```

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

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

</smil>

```

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

```

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

```

However this is a temporary solution a way to get resources without saving them to disk have to be studied. It works but it consumes a lot of memory space. Considering the limited space in different clients, such as PDA or mobile phone, this may does not work.

How to play a SMIL component directly from AXMEDIS object?

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

1. Opening a SMIL component inside an AXMEDIS object

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

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

- A **window factory** is usually implemented by the main program itself. The player will call this when it needs a window. Usually the first request to create a window will actually return the document window (after resizing to the appropriate size).
- A **global playable factory** is created. This is the object the player will use to create renderers for the various media types. The global factory is filled with the various renderers this ambulant player supports. In effect, this is the step where you get to decide how various media are rendered.

- A **datasource factory** is created and filled with the factory functions that will create datasources for audio, video or other, raw, data such as text. The factory functions that are added to the datasource factory partially determine how data is retrieved over the net, which protocols and formats are supported and such. Partially, because some media items (audio and video, notably) may be rendered by simply passing the URL to some underlying media infrastructure such as DirectX or QuickTime. Note, **since we will directly retrieve the resource through “AxIndex” instead of using original URL, we will modify the code here. i.e., to generate the data source or data stream from “AxIndex”.**

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

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

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

2. Creating the player (same as AMBULANT)

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

- A timer and event processor are created. The timer is the master clock for the presentation, and the event processor is a runqueue object that is used for low-level scheduling. Whenever the high-level scheduler wants some code to be executed it will add an event to the event processor, possibly with a timeout and a priority. The event processor runs in a separate thread, waits for events in its 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 toplevel windows the new topsurface method of the window factory is used, and subregions are created using the new subsurface method of their parent surface template. The layout manager also contains mappings to be able to get from a node to the corresponding region node to the surface template, and this will be used during playback to play media items in the correct location.
- A timegraph is created. This is the internal representation of the <body> part of the DOM tree that will be used to play back the document. In addition a scheduler is created, which will interpret the data in the timegraph.

3. Starting playback

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

23.2 Module Design in terms of Classes

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

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

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

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

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

The **load** function will be used to load from the AXMEDIS document a resource with identifier **axoid** -the parameter of the function.

The **getMediaClient** function returns a wxWindow reference. This poses a problem to the AMBULANT player because, in its Windows version, it does not use wxWidgets but MFC. There is no easy conversion from an MFC window object to a wxWidget window object. One solution could be adding another function to the **axMediaPlayer** interface **getMediaClientMM()** which could return a reference to a custom object **axmedisWindow**. The **axmedisWindow** object should implement the set of functions from **wxWindow** that would be needed, for instance:

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

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

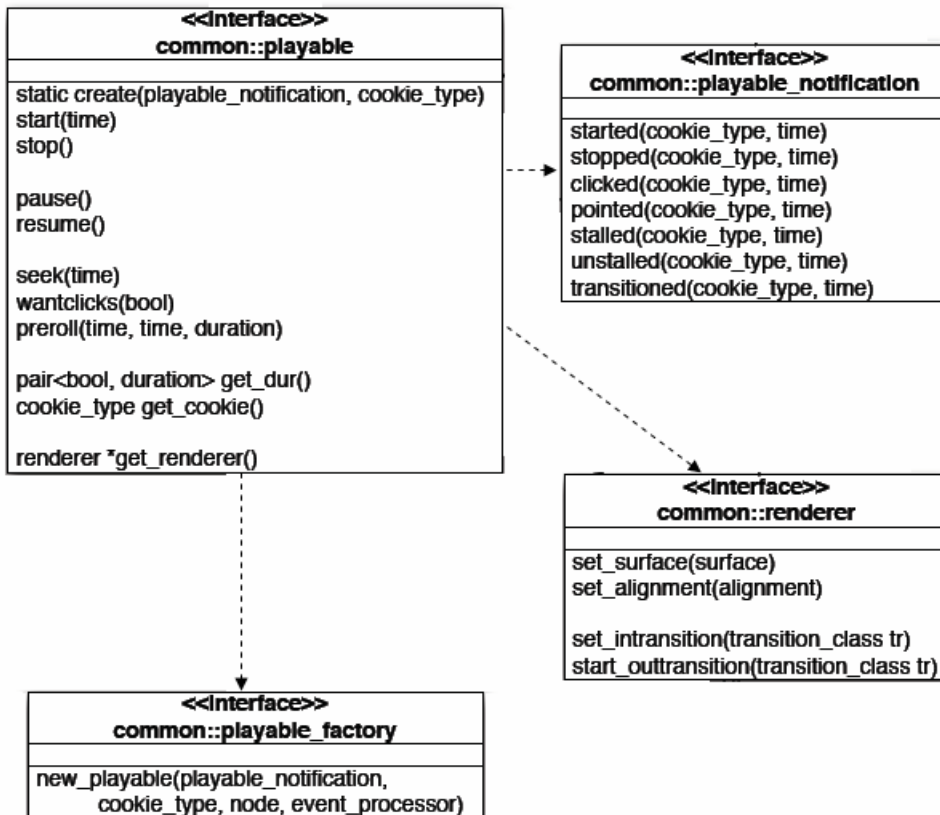
```
/// This is the API an embedding program would use to control the
/// player, to implement things like the "Play" command in the GUI.
```

```

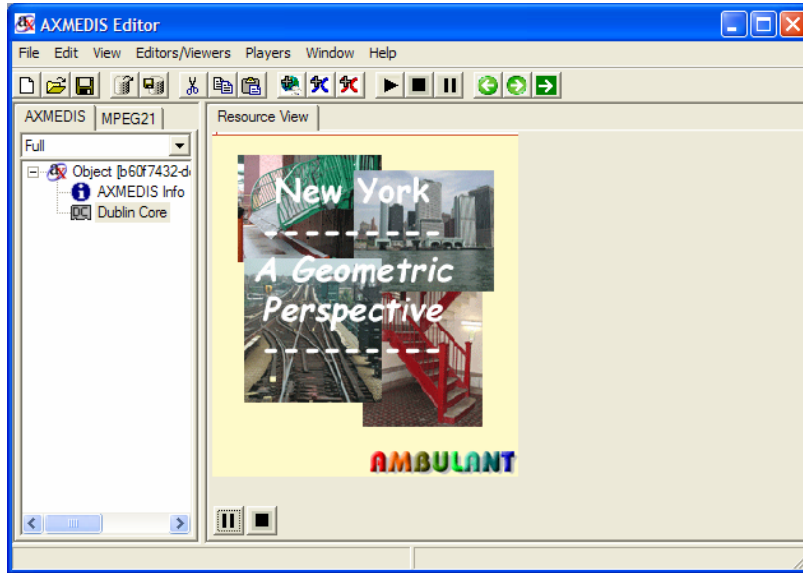
class player {
public:
    virtual ~player() {};

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

```



23.3 User interface description



23.4 Technical and Installation information

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

23.5 Draft User Manual

23.6 Examples of usage

23.7 Integration and compilation issues

None

23.8 Configuration Parameters

Config parameter	Possible values

23.9 Errors reported and that may occur

Error code	Description and rationales

24 Module - Internal Document Viewer (DSI)

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

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

24.1 General Description of the Module

The document viewer will support the visualization of documents like:

- HTML
- PDF
- MSWord Documents
- Postscript

The functionalities provided will be:

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

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

24.1.1 HTML

To view HTML files two possibilities are available:

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

On one hand, wxActiveX (<http://sourceforge.net/projects/wxactivex>) is a library providing a wrapper class to host ActiveX controls inside wxWidgets applications.

This library also provides an interface to host the Internet Explorer's ActiveX.

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

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

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

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

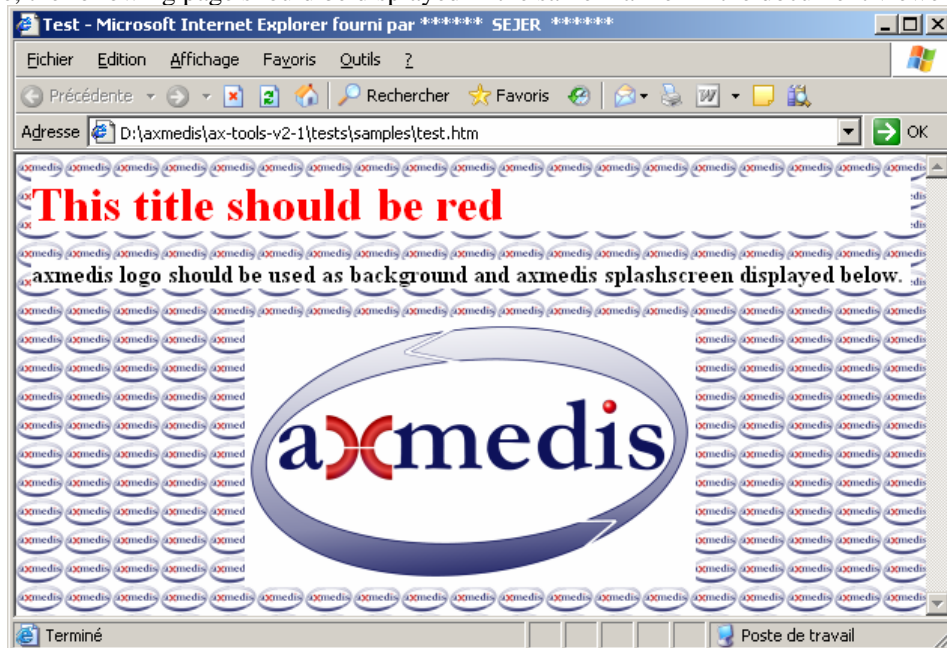
To access to protected content the use of streams will avoid saving the HTML files in clear on disk however it is not clear how to provide content referred from the protected html (e.g. images).

Some restrictions could be put to avoid the possibility of copy to clipboard selected text or to view the HTML. To avoid this some filters on events could be set to filter the Ctrl+C and to block the right click in case of protected content.

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

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

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



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

The basic principle is to register for a specific URL protocol, such as axmedis://, an object which responsibility is to retrieve data for a certain URL.

Also, AXMEDIS object identifier are URN but we can map them to URL using the following conversion:

urn:axmedis:<yyyy>:obj:<uuid> => axmedis://<yyyy>.obj.<uuid>/

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

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

24.1.2 MSWord Documents

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

24.1.3 PDF

To display PDF files the following possibilities are available:

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

The first three solution use directly on indirectly the Acrobat ActiveX control:

The wxActiveX library could used to host the Acrobat ActiveX inside a wx application.

The interface provided by the Acrobat ActiveX is very simple; basically it allows opening a file from the file system. In case of protected content no way was found to load the PDF file from a stream object avoiding to store the file in clear on the file system. However to protect pdf content the Acrobat DRM can be used and customized for the AXMEDIS needs.

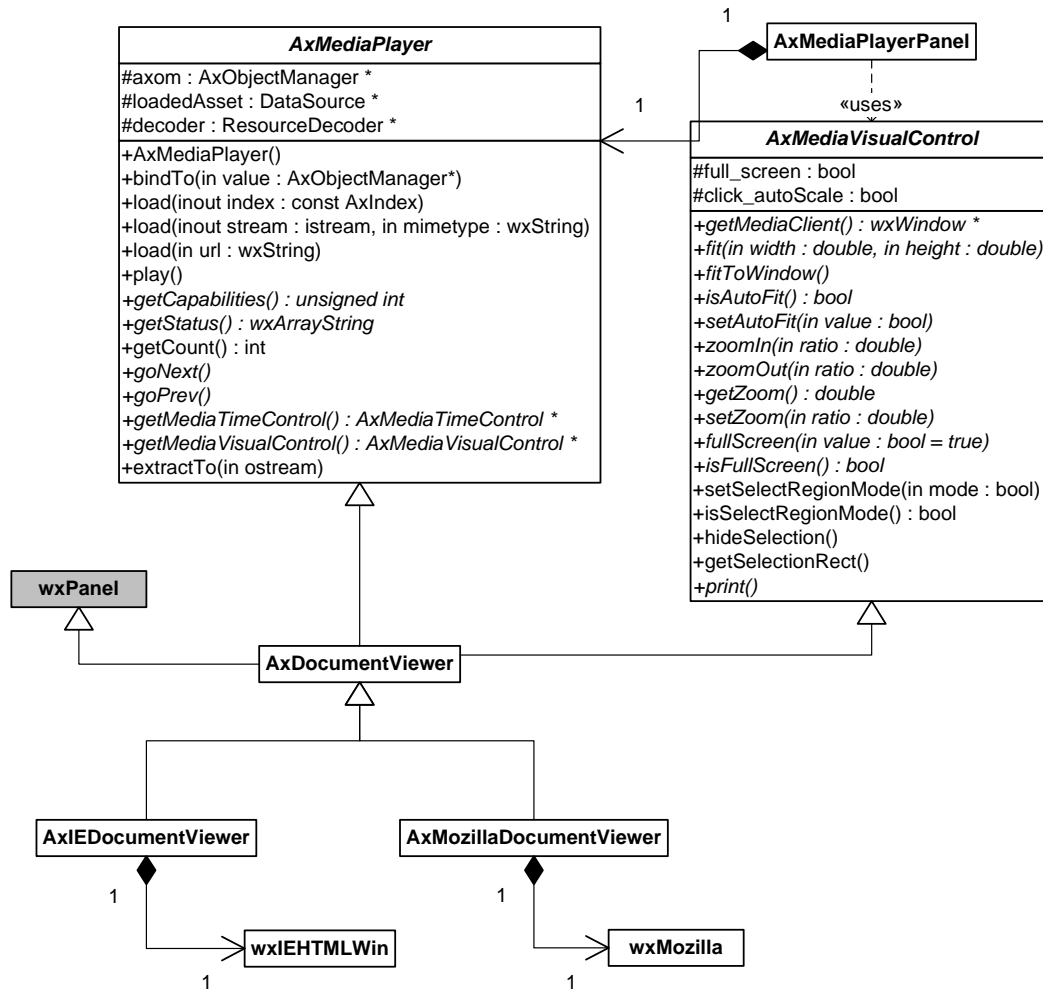
In particular Acrobat supports protection plug-in allowing defining the key to encrypt the PDF file. This key may be generated and stored with the PDF file on the file system. When the PDF is put inside an AXMEDIS object also this key is put inside and protected. When the protected PDF file is opened it is stored on disk (unusable without the key) and the Acrobat ActiveX is used to open it, the AXMEDIS specific protection plug-in for Acrobat will acquire the key from the AXEditor (e.g. via a protected channel) allowing it to display the document. Note that the use of the protection plugin inside AcrobatReader is not free and it has a not negligible cost.

Another solution is to use and adapt some free library for PDF rendering (Xpdf, ghostscript etc.) and to handle protection mechanisms in a more effective way. However these libraries may not support all the features of PDF files.

24.1.4 Postscript

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

24.2 Module Design in terms of Classes



24.2.1 Protocol handlers

wxMozilla provide a set of classes and functions to ease the implementation and registering of protocol handlers: wxProtocolHandler, wxProtocolChannel.

Mozilla separates the handling of the protocol handler and the retrieval of the data in two classes, while Internet Explorer queries for two interfaces on the registered protocol handler.

Therefore, the logic will be implemented in AxProtocolHandler, which derived from wxProtocolHandler, for all url management aspects, and AxProtocolChannel, which derived from wxProtocolChannel, for actual data retrieval of the AXMEDIS resources.

The AxIEDocumentViewer will implement IInternetProtocolRoot, IInternetProtocol and IInternetProtocolInfo and delegates the call to AxProtocolHandler and AxProtocolChannel.

For internet explorer, as the protocol handler is a COM Object, the AXIEDocumentViewer will implement also a class factory (IClasFactory) in order to be able to instantiate the protocol handler without having to register it in the windows registry.

24.2.2 AxIEDocumentViewer and AxMozillaDocumentViewer

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

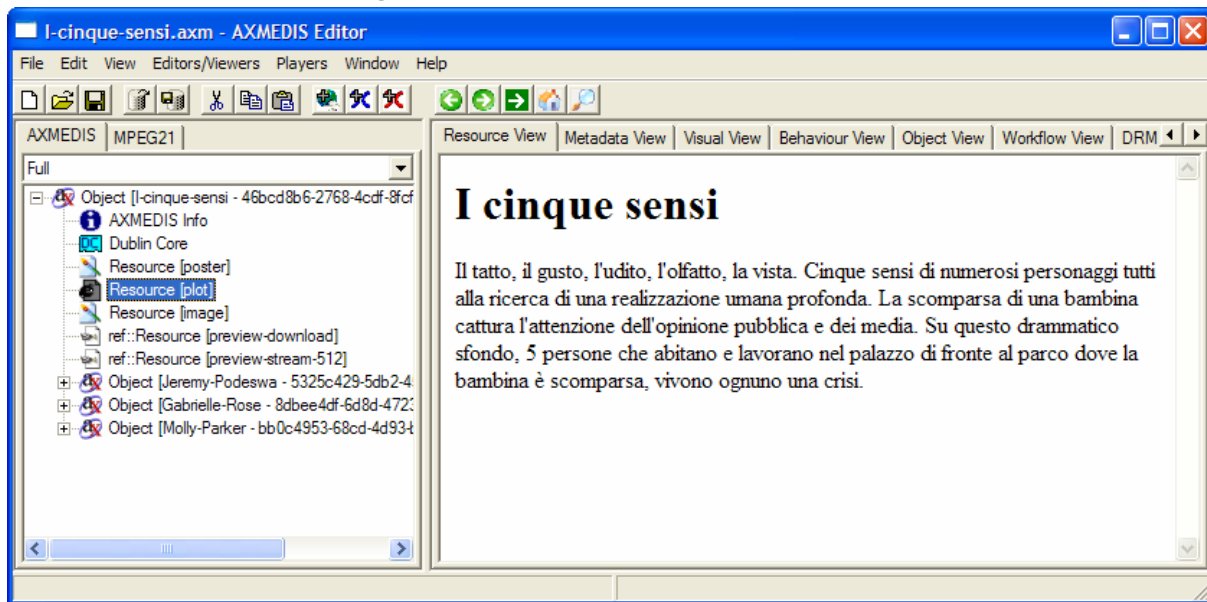
- play
- getCount
- goNext
- goPrevious

- fit
- zoom
- getSelectionRect

The zoom can be emulated by changing the size of the base font.

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

24.3 User interface description



24.4 Technical and Installation information

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

24.5 Draft User Manual

24.6 Examples of usage

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

24.7 Integration and compilation issues

None

24.8 Configuration Parameters

Config parameter	Possible values

24.9 Errors reported and that may occur

Error code	Description and rationales

25 Tool – MPEG4 Player (EPFL)

Tool	
MPEG4 Player	
Responsible Name	Antonio Romeo
Responsible Partner	EPFL
Status (proposed/approved)	
Implemented/not implemented	Implemented
Status of the implementation	
Executable or Library/module (Support)	Executable
Single Thread or Multithread	Multithread
Language of Development	C/C++, assembly (optional)
Platforms supported	MS 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	
Reference to the AXFW location of the demonstrator executable tool for public download	
Address for accessing to WebServices if any, add accession information (user and Passwd) if any	
Test cases (present/absent)	
Test cases location	
Usage of the AXMEDIS configuration manager (yes/no)	
Usage of the AXMEDIS Error Manager (yes/no)	

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

25.1 General Description of the Module

The MPEG-4 Tools and Services team at ENST has recently released version 0.4.0 of the GPAC multimedia suite containing the last release of Osmo4, an MPEG-4 player with the following features:

- MPEG-4 Systems player based on new, smaller, C-only, portable code that is released under LGPL.
- Optimized 2D graphics renderer compliant with the Complete2D Scene Graph and Graphics profiles.
- Video and audio presentation achieved through plugins based on famous OpenSource packages:

- Xvid for MPEG-4 Video Simple Profile.
 - MAD for MP3 audio.
 - JPEG and PNG for still images.
 - A Decoder Development Kit (DDK) is available for developers to interface additional codecs; a source code template for FAAD (MPEG-4 AAC audio) is included: the sources for the FAAD decoder have been successfully compiled and interfaced with the player using the DDK.
- Multimedia player features:
 - Timeline controls: play, pause, step.
 - Graphics features: antialiasing, zoom and pan, scalable resizing of rendering area, basic full screen support.
 - Support for Advanced Text and Graphics extension of MPEG-4 Systems under standardization.
 - Frame export to JPG, PNG, BMP.
- OSMO is currently being ported to PC Linux.

OSMO supports also subtitles and a very wide range of options such as:

- Rendering
 - Variable frame rate
 - Anti-aliasing level selection
 - 2D rasterizer
 - 3D renderer
- MPEG-4 Systems:
 - Preferred language for streams selection
 - Decoder threading selection
- Selection of preferred media decoders
- Selection of preferred media drivers
- Fine tuning of Real-Time streaming configuration
- Streaming Cache.

25.2 User interface description

OSMO comes with a user friendly interface allowing enhanced access to the decoded content both as locally stored file and as streamed content. Together with very basic functionality such as “play”, “stop” and “pause” it provides enhanced functionality such as frame resize (also full screen supported), deep inspection of the MPEG-4 stream structure in terms of Elementary Streams and Object Descriptors. In addition OSMO allows the creation of Playlists and stores an history of usage of the last decoded resources.

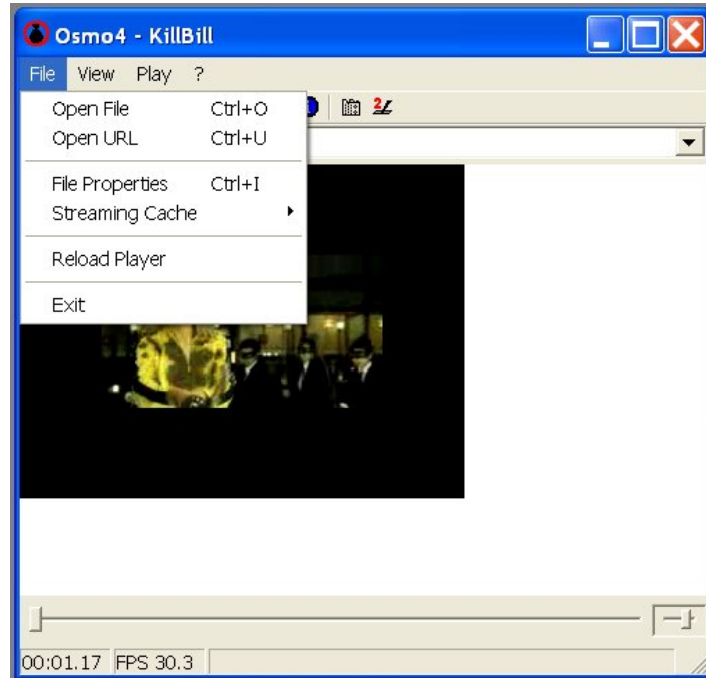


Figure 16 - OSMO supports both offline playing and streaming

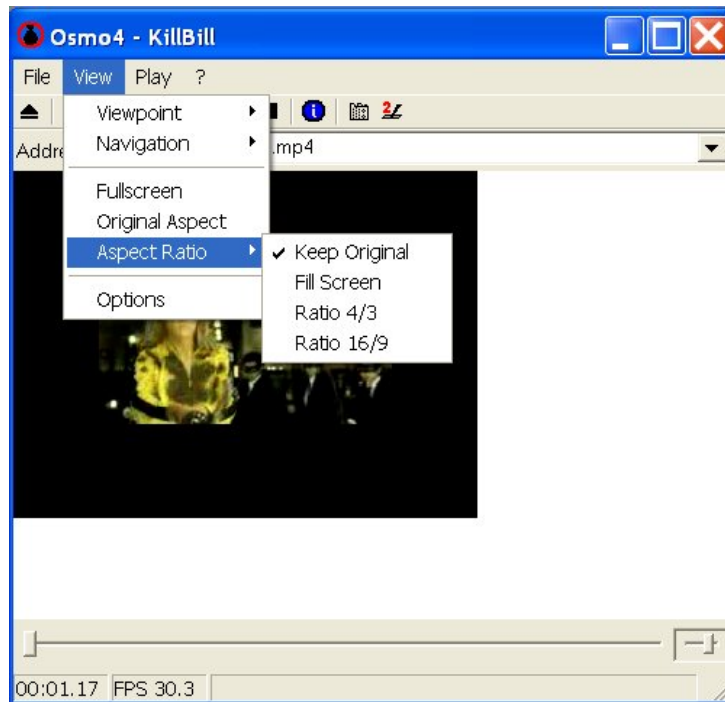


Figure 17 - OSMO supports frames resize

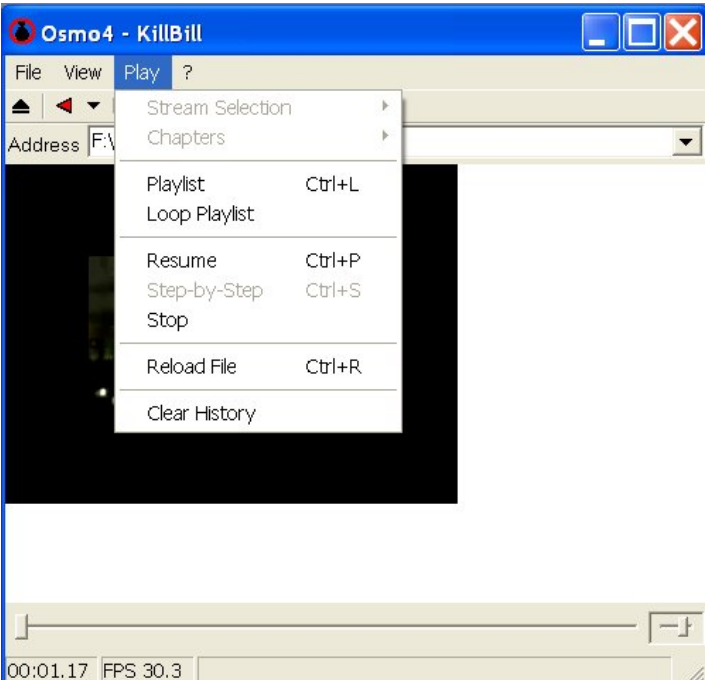


Figure 18 - OSMO GUI showing the available options to control content decoding.

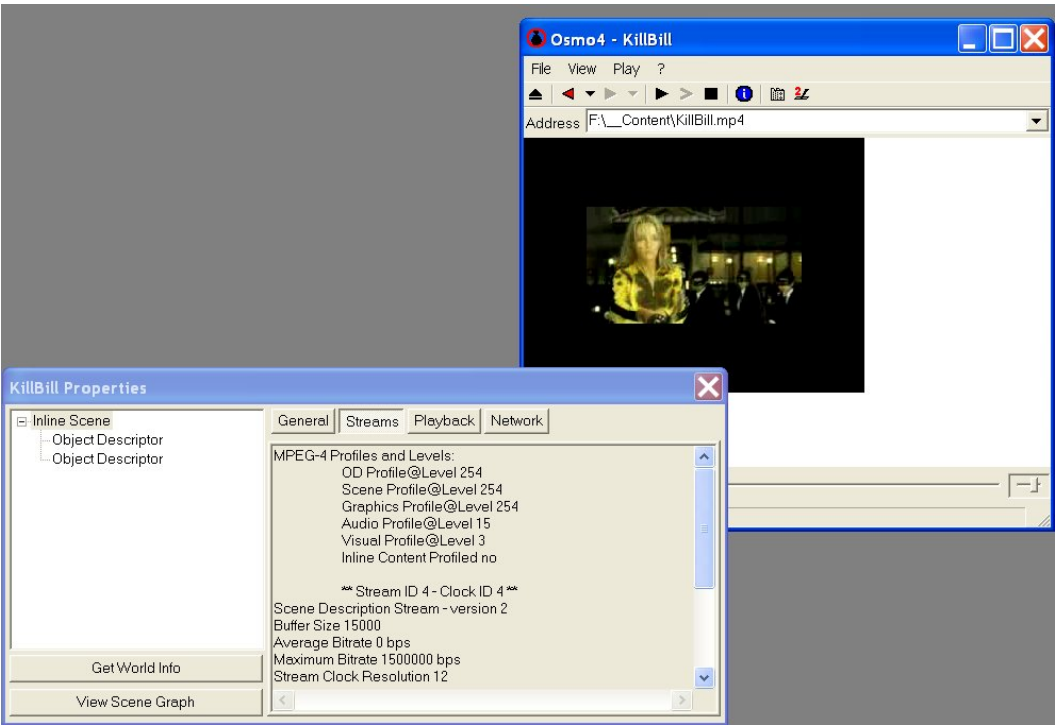


Figure 19 - OSMO allows deep inspection of the mp4 streams characteristics.

25.3 Technical and Installation information

References to other major components needed	
Problems not solved	<ul style="list-style-type: none">

Configuration and execution context	
-------------------------------------	--

25.4 Draft User Manual

see User Interface Description

25.5 Examples of usage

see User Interface Description.

25.6 Integration and compilation issues

none

25.7 Configuration Parameters

Config parameter	Possible values

25.8 Errors reported and that may occur

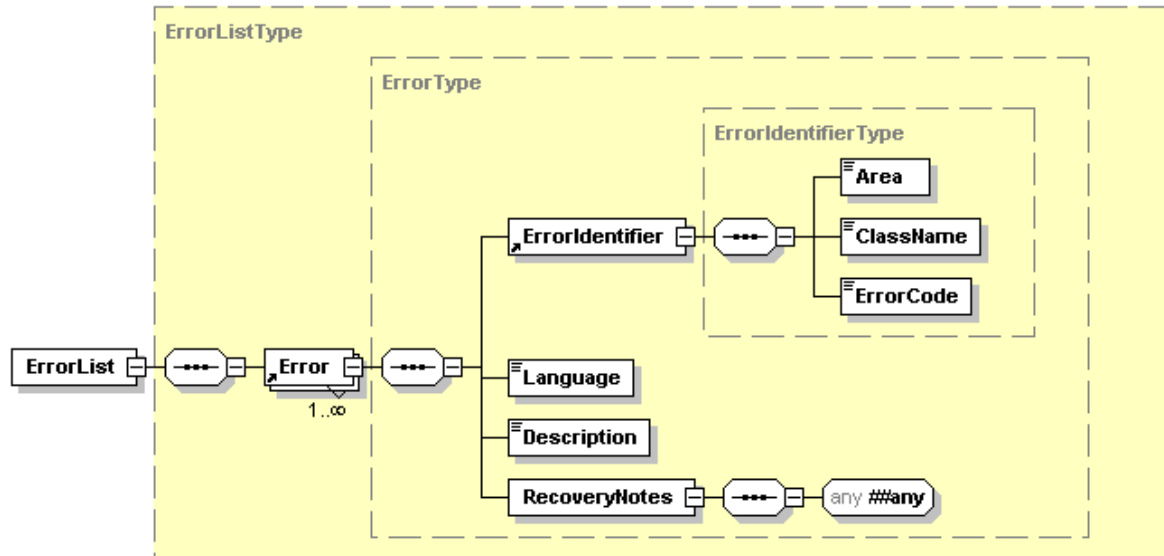
Error code	Description and rationales

26 Formal description of format – Error Coding (DSI)

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

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

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



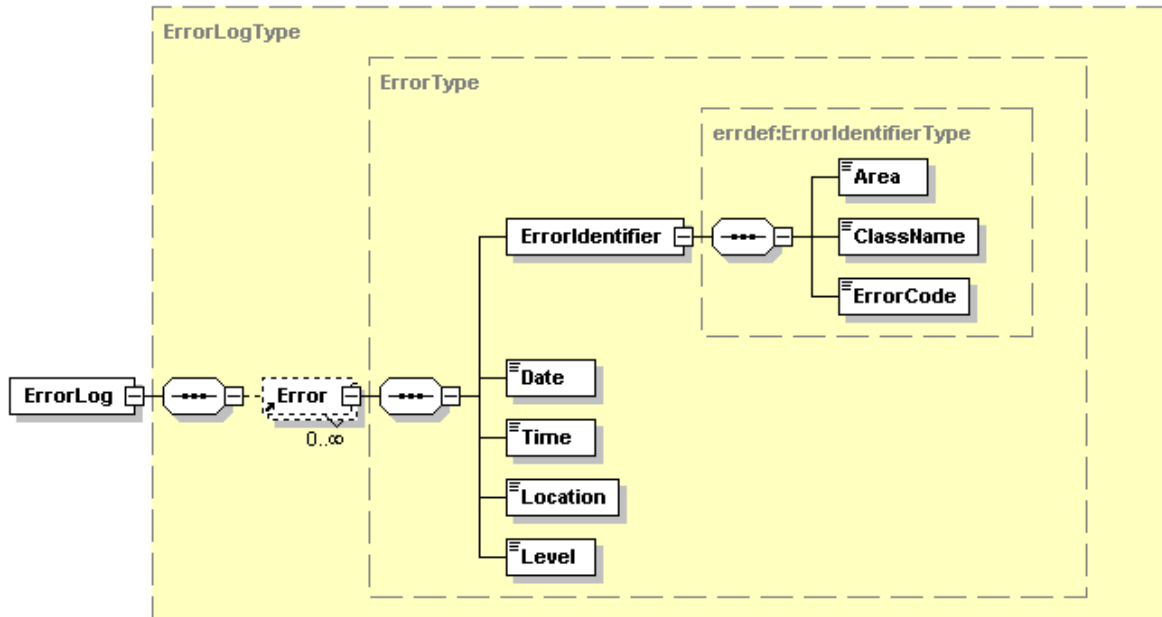
```

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

```

27 Formal description of format – Error Log (DSI)

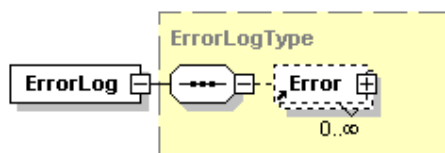
Every time an error is raised through the error manager interface the error is logged somewhere. The location of the log file can be configured using the AXMEDIS Configuration Manager. The log file is in XML format and has the following schema:



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

element ErrorLog

diagram

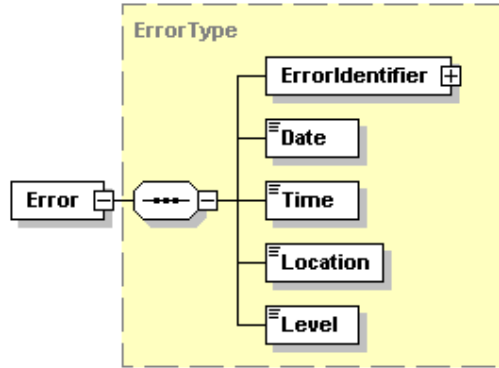


namespace <http://www.axmedis.org/error-log>

type [ErrorLogType](#)
 children [Error](#)
 source `<xs:element name="ErrorLog" type="ErrorLogType"/>`
 description This is the root element of the log file. It can contain none or more **Errors**.

element Error

diagram



namespace <http://www.axmedis.org/error-log>
 type [ErrorType](#)
 children [ErrorIdentifier](#) [Date](#) [Time](#) [Location](#) [Level](#)
 used by complexType [ErrorLogType](#)
 source `<xs:element name="Error" type="ErrorType"/>`
 description This element represent a logged error. It consist of the following information:

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

28 Formal description of format – Configuration (DSI)

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

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

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

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

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

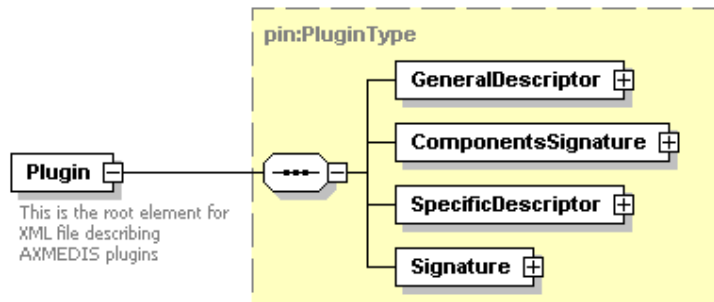
Plug-in profiles have to respect the following XML schema.

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

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

element **Plugin**

diagram



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

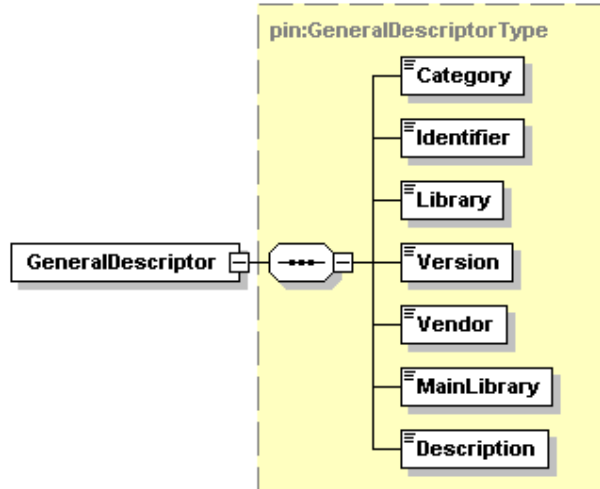
type **pin:PluginType**

children **GeneralDescriptor ComponentsSignature SpecificDescriptor Signature**

description This element is the root element of profiles for AXMEDIS plug-ins. It contains necessary information to manage and to use the associated plug-in.

element **GeneralDescriptor**

diagram

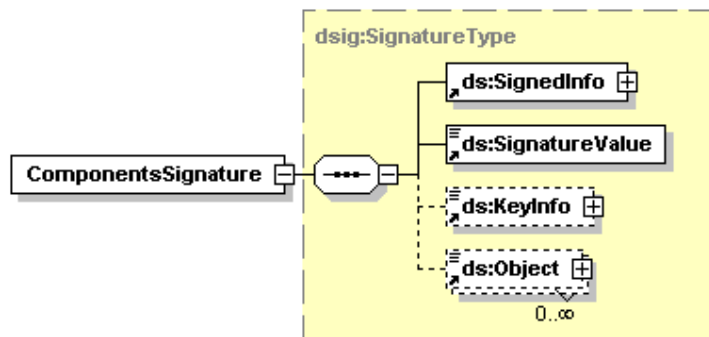
namespace <http://www.axmedis.org/plugin-schema>type **pin:GeneralDescriptorType**children **Category Identifier Library Version Vendor MainLibrary Description**

description This element contain general information on the plug-in the profile refers to. That information is mandatory and valid for all kind of plug-in. The fields are:

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

element **ComponentsSignature**

diagram

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

attributes	Name Id	Type ID	Use optional	Default	Fixed
description	This element is a dsig:SignatureType . It is the signature (estimated by an AXCS) of the entire plug-in. The signature comprises all relevant resources which compose the plug-in. Those resources are listed as Reference elements of ds:SignedInfo .				

element **SpecificDescriptor**

diagram

SpecificDescriptor

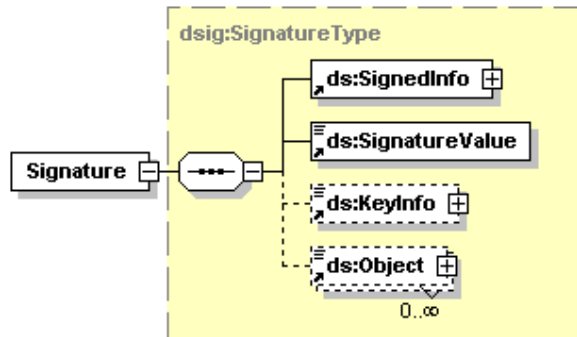
namespace <http://www.axmedis.org/plugin-schema>

type **pin:SpecificDescriptorType**

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

element **Signature**

diagram



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

type [dsig:SignatureType](#)

children [dsig:SignedInfo](#) [dsig:SignatureValue](#) [dsig:KeyInfo](#) [dsig:Object](#)

attributes	Name Id	Type ID	Use optional	Default	Fixed
description	This is the signature of the whole profile except the Signature element itself. It has been introduced to guarantee the dependability of the data contained in the manifest.				

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

```

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

```

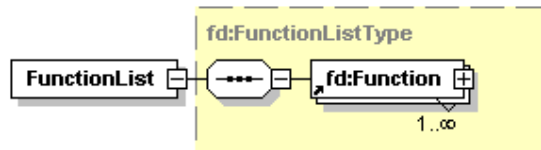
```

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

```

element FunctionList

diagram



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

type [fd:FunctionListType](#)

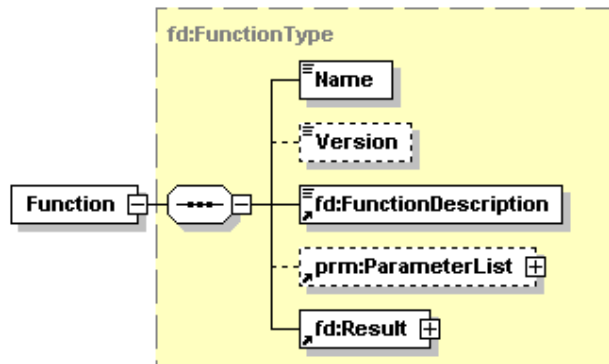
children [fd:Function](#)

source `<xs:element name="FunctionList" type="fd:FunctionListType" substitutionGroup="pin:SpecificDescriptor"/>`

description This element is the root element for the description of all function exposed by a content processing plug-in. It is a substitution of the abstract element *SpecificDescriptor* described in the section 29

element Function

diagram



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

type [fd:FunctionType](#)

children [Name](#) [Version](#) [fd:FunctionDescription](#) [prm:ParameterList](#) [fd:Result](#)

used by complexType [FunctionListType](#)

description This element represent a single function exposed by a plug-in. A function is characterized by a Name, which is used in *AXMEDIS Project*

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

element **FunctionDescription**

diagram



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

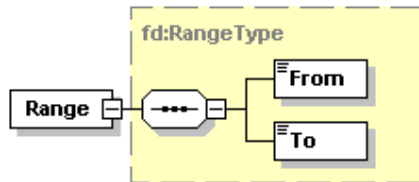
type **xs:string**

used by complexType [FunctionType](#)

description This element can contain any string and represents the description of the function (e.g. the help for the function).

element **Range**

diagram



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

type [fd:RangeType](#)

children [From To](#)

source `<xs:element name="Range" type="fd:RangeType" substitutionGroup="prm:Constraint"/>`

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

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

both extreme limits can be included or excluded. In fact, those elements have the **included** attribute which can be true or false. This element is a substitution group of the Constrain element which is explained in the section 31.

complexType **Limit**

diagram



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

type extension of **xs:string**

used by elements [RangeType/From](#) [RangeType/To](#)

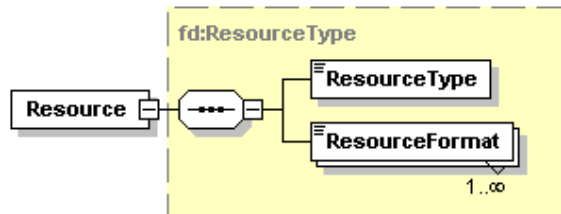
attributes	Name	Type	Use	Default	Fixed	Annotation
	included	xs:boolean	optional			

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

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

element Resource

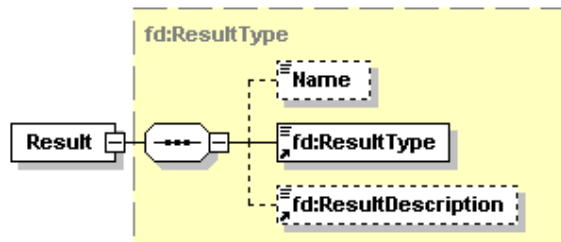
diagram

namespace <http://www.axmedis.org/plugin-function-schema>type [fd:ResourceType](#)children [ResourceType](#) [ResourceFormat](#)source `<xs:element name="Resource" type="fd:ResourceType" substitutionGroup="prm:Constrain"/>`

description This element represent a constraint on the type of resource which can be passed as parameter to a function. It has to be used in conjunction to a parameter of type **RESOURCE** to determine the acceptable MIME Type for it. The **Type** element corresponds to the MIME type while **Format** element corresponds to the MIME subtype. Information about the MIME content-types can be found in the RFC 2045 [RFC2045], 2046 [RFC2046] and 2077 [RFC2077]. *ResourceType* and *ResourceFormat* elements are of type *string*. This element is a substitution group of the Constrain element which is explained in the section 31.

element Result

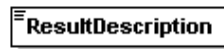
diagram

namespace <http://www.axmedis.org/plugin-function-schema>type [fd:ResultType](#)children [Name](#) [fd:ResultType](#) [fd:ResultDescription](#)used by complexType [FunctionType](#)

description This element represents the return value of the function. It is mainly described by its type but it can also own a short name and a description. The *Name* element is of type *ID*.

element ResultDescription

diagram

namespace <http://www.axmedis.org/plugin-function-schema>type **xs:string**used by complexType [ResultType](#)

description This element can contain any string and represents the description of the return value of the function

element ResultType

diagram

namespace <http://www.axmedis.org/plugin-function-schema>type [fd:ResultTypeType](#)used by complexType [ResultType](#)

facets	enumeration	UINT16
	enumeration	INT16
	enumeration	UINT32
	enumeration	INT32
	enumeration	FLOAT
	enumeration	DOUBLE
	enumeration	BOOLEAN
	enumeration	STRING
	enumeration	WSTRING
	enumeration	CHAR
	enumeration	RESOURCE
	enumeration	AXOM
	enumeration	AREA
	enumeration	VOID
description	This element represents the type of the return value. It is almost the same of type of parameter (see section 31) but it can be also <i>VOID</i> .	

31 Formal description of format – Parameter description

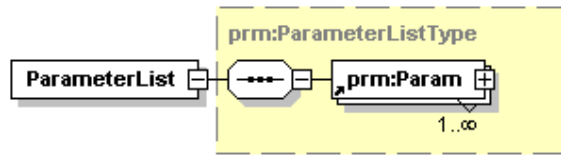
```

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

```

element **ParameterList**

diagram



namespace <http://www.axmedis.org/parameter>

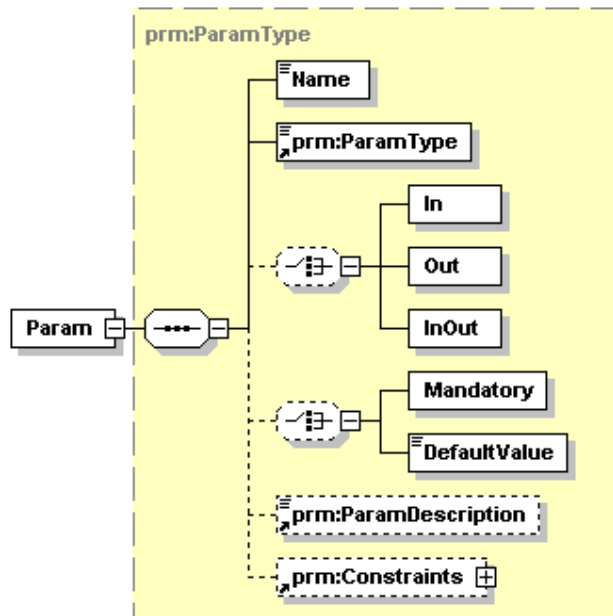
type [prm:ParameterListType](#)

children [prm:Param](#)

description This element represents a list of parameters.

element **Param**

diagram



namespace <http://www.axmedis.org/parameter>

type [prm:ParamType](#)

children [Name](#) [prm:ParamType](#) [In](#) [Out](#) [InOut](#) [Mandatory](#) [DefaultValue](#) [prm:ParamDescription](#) [prm:Constraints](#)

used by complexType [ParameterListType](#)

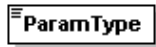
description This element represents a parameter of a plug-in function. Each parameter is characterized by the following field:

- **Name** – the name of the parameter. It is of type *ID*.
- **ParamType** – see below
- **In, Out, InOut** – only one of these field can be used for each parameter. They represent the direction of the parameter:
 - **In** – represents a parameter which will not be changed by the function
 - **Out** – represents a parameter whose value is not used by the function and which will be modified by it
 - **InOut** – represents a parameter whose value is used by the function and which will be modified by it by default the parameter will be considered of type **InOut**.
- **Mandatory, DefaultValue** – only one of these field can be used for each parameter. They represent the mandatoryness of the parameter:
 - **Mandatory** – it means that a value has to be given for this parameter
 - **DefaultValue** – it is used to provide a default value for the parameter if it is not explicitly given. It is of type *anySimpleType*, i.e. it can contain any value which can be represented as a simple type
- **ParamDescription** – see below

- **Constraints** – see below

element **ParamType**

diagram



namespace <http://www.axmedis.org/parameter>

type [prm:ParamTypeType](#)

used by complexType [ParamType](#)

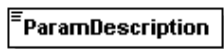
facets

enumeration	UINT16
enumeration	INT16
enumeration	UINT32
enumeration	INT32
enumeration	FLOAT
enumeration	DOUBLE
enumeration	BOOLEAN
enumeration	STRING
enumeration	WSTRING
enumeration	CHAR
enumeration	RESOURCE
enumeration	AXOM
enumeration	AREA

description This element represents the type of a parameter. The definition of this element fixes the set of parameter type which can be exchanged among AXOM and plug-ins. A non-exhaustive list is reported above. Each type reported in the list corresponds to a specific type in the programming language.

element **ParamDescription**

diagram



namespace <http://www.axmedis.org/parameter>

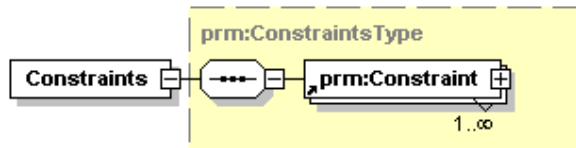
type **xs:string**

used by complexType [ParamType](#)

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

element **Constraints**

diagram



namespace <http://www.axmedis.org/parameter>

type [prm:ConstraintsType](#)

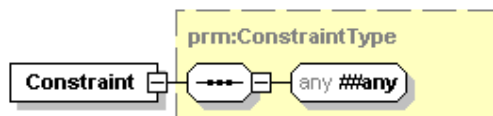
children [prm:Constraint](#)

used by complexType [ParamType](#)

description This element contains the constraints which the parameter is liable to. **Constraints** can contain several kind of constraints, see below.

element **Constraint**

diagram



namespace <http://www.axmedis.org/parameter>

type [prm:ConstraintType](#)

used by	complexType ConstraintsType
source	<code><xs:element name="Constraint" type="prm:ConstraintType" abstract="true"/></code>
description	This is an abstract element which is the base for all those elements which represent a constrain for a given parameter (see section 30 for some example of constraint).