Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE3.1.2.2.3 Specification of AXMEDIS Object Manager and Protection Processor, first update of DE3.1.2 part B

Version: 2.0
Date: 08-05-2006
Responsible: DSI (Vallotti) (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: DE3.1.2.2.3 Contractual Date of Delivery: M18 Actual Date of Delivery: 17/5/2006 Title of Deliverable: Specification of AXMEDIS Object Manager and Protection Processor, first update of DE3.1.2 part B Work-Package contributing to the Deliverable: WP3.1 Task contributing to the Deliverable: WP3, WP2 Nature of the Deliverable: report Author(s): DSI, EPFL (MISSING Contributions), FUPF, FHGIGD

Abstract: this part includes the specification of components, formats, databases and protocol related to the AXMEDIS Framework area including AXOM, loading and saving MPEG-21 objects, formats, protection processor.

Keyword List: AXOM, Object Model, MPEG-21, BIM, IPMP, Protection Information

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 ... 7
1.1 THIS DOCUMENT CONCERNS (DSI) ... 8
1.2 LIST OF MODULES OR EXECUTABLE TOOLS SPECIFIED IN THIS DOCUMENT (DSI) ... 8
1.3 LIST OF FORMATS SPECIFIED IN THIS DOCUMENT (DSI) ... 9
2 GENERAL ARCHITECTURE AND RELATIONSHIPS AMONG THE MODULES PRODUCED (DSI, ALL) ... 10
3 MPEG-21 OBJECT MODEL (DSI) ... 11
3.1 GENERAL DESCRIPTION OF THE MODULE ... 12
3.2 MODULE DESIGN IN TERMS OF CLASSES ... 12
3.2.1 DI's Object Model Capabilities Overview ... 14
3.2.2 Classes and Methods Overview ... 16
3.3 EXAMPLES OF USAGE ... 20
3.4 INTEGRATION AND COMPILATION ISSUES ... 20
3.5 ERRORS REPORTED AND THAT MAY OCCUR ... 21
4 MPEG-21 LOADER (DSI) ... 22
4.1 GENERAL DESCRIPTION OF THE MODULE ... 23
4.2 MODULE DESIGN IN TERMS OF CLASSES ... 24
4.2.2 Class Methods Overwiew ... 26
4.3 EXAMPLES OF USAGE ... 30
4.4 INTEGRATION AND COMPILATION ISSUES ... 30
4.5 CONFIGURATION PARAMETERS ... 30
4.6 ERRORS REPORTED AND THAT MAY OCCUR ... 30
5 MPEG-21 SAVER (DSI) ... 32
5.1 GENERAL DESCRIPTION OF THE MODULE ... 33
5.2 MODULE DESIGN IN TERMS OF CLASSES ... 33
5.2.1 MPEG-21Saver Capabilities Overview ... 34
5.2.2 Class and Methods Overview ... 35
5.3 EXAMPLES OF USAGE ... 37
5.4 INTEGRATION AND COMPILATION ISSUES ... 37
5.5 ERRORS REPORTED AND THAT MAY OCCUR ... 37
6 AXMEDIS OBJECT MODEL (DSI) ... 38
6.1 GENERAL DESCRIPTION OF THE MODULE ... 39
6.2 MODULE DESIGN IN TERMS OF CLASSES ... 39
6.2.1 AxMetadata ... 41
6.2.2 AxInfo ... 41
6.2.3 AxDublinCore ... 45
6.2.4 AxOID ... 46
6.2.5 AxContent ... 46
6.2.6 AxObject ... 46
6.2.7 AxResource ... 47
6.2.8 AxPublicMetadataTree ... 48
6.2.9 AxLoader ... 50
7 AXMEDIS OBJECT PREPROCESSOR AND POSTPROCESSOR (EPFL) ... 52
7.1 GENERAL DESCRIPTION OF THE MODULE ... 53
7.2 MODULE DESIGN IN TERMS OF CLASSES ... 53
7.2.1 Reference Solver ... 53
7.2.2 MPEG-21 Binarization ... 55
7.3 TECHNICAL AND INSTALLATION INFORMATION ... 55
7.4 DRAFT USER MANUAL ... 55

7.5 EXAMPLES OF USAGE 55
7.6 INTEGRATION AND COMPILATION ISSUES 55
7.7 ERRORS REPORTED AND THAT MAY OCCUR 55
7.8 FORMAL DESCRIPTION OF ALGORITHM <…………> 55

8 PROTECTION PROCESSOR (DSI) 57

8.1 GENERAL DESCRIPTION OF THE MODULE 58
8.1.1 AXMEDIS tool registration and certification 58
8.1.1.1 Software and hardware fingerprint 58
8.1.1.2 Tool certificate 59
8.1.1.3 Tool Registration Certificate 60
8.1.1.4 User certificate 61
8.1.1.5 User Identifier/Identification 61
8.1.1.6 Date and time 62
8.1.1.7 Action history 62
8.1.1.8 Enabling code 62
8.1.1.9 Trustiness of a tool 62
8.1.1.10 Certified software 63
8.1.1.11 Execution controls 63
8.1.2 Robustness against malicious user actions 63
8.2 MODULE DESIGN IN TERMS OF CLASSES 64
8.2.1 General Structure 64
8.2.2 ProtectionProcessor and CipherDataSourceManager 65
8.2.3 Cipher streams 67
8.2.4 Protection Information interpretation 68
8.2.5 Threads 69
8.2.6 Certificate Interface Module 69
8.2.7 Fingerprint Module 70
8.2.8 Tool certification 71
8.2.9 Grant Authorisation Requests 73
8.2.10 Tool recovery/recertification 77
8.2.11 Protection Tools as AXMEDIS Plug-ins 77
8.3 INTEGRATION AND COMPILATION ISSUES 78
8.4 CONFIGURATION PARAMETERS 78
8.5 ERRORS REPORTED AND THAT MAY OCCUR 78

9 ENCRYPTION/DECRYPTION SUPPORT (FUPF) 80

9.1 GENERAL DESCRIPTION OF THE MODULE 81
9.2 MODULE DESIGN IN TERMS OF CLASSES 81
9.2.1 Architecture for encryption / decryption support 81
9.3 IMPLEMENTATION OF THE ALGORITHMS 82
9.4 EXAMPLES OF USAGE 82
9.5 FORMAL DESCRIPTION OF ENCRYPTION / DECRYPTION SUPPORT OPERATIONS 83

10 COMPRESS/UNCOMPRESS SUPPORT (DSI) 85

10.1 GENERAL DESCRIPTION OF THE MODULE 86
10.2 EXAMPLES OF USAGE 86

11 SCRAMBLE/DESCRAMBLE SUPPORT (EPFL) 88

11.1 GENERAL DESCRIPTION OF THE MODULE 89
11.2 MODULE DESIGN IN TERMS OF CLASSES 90
11.3 TECHNICAL AND INSTALLATION INFORMATION 91
11.3.1 Initialisation 91
11.4 INTEGRATION AND COMPILATION ISSUES 92

12 MPEG-21 DIBO (EPFL) 93

12.1 MPEG-21 DIP IN AXMEDIS 94
12.2 GENERAL DESCRIPTION OF THE MODULE 95
12.3 CLASSES 95

13 MPEG-21 DIM (EPFL) 97

13.1 MPEG-21 DI METHODS IN AXMEDIS 98
13.1.1 Relationship between DIMs, DIBOs, and DIXOs 99
13.2 GENERAL DESCRIPTION OF THE MODULE 99

14 MPEG-21 DIA PROCESSING(EPFL) 100

14.1 GENERAL DESCRIPTION OF THE MODULE 101

15 MPEG-21 DIA (EPFL) 104

16 AXMEDIS DATA MODEL (DSI) 105

16.1 AXMEDIS OBJECTS AS MPEG21 OBJECTS 105
16.1.1 MPEG21 Digital Items 105
16.1.2 AXMEDIS Objects 107
16.1.3 Basic AXMEDIS Object: 108
16.1.4 Protected Basic AXMEDIS Object: 109
16.1.5 Composite AXMEDIS Object: 110
16.1.6 Protected Composite AXMEDIS Object: 111
16.1.7 Referred AXMEDIS Object 112
16.1.8 Governed AXMEDIS Object: 112
16.1.9 AXMEDIS Metadata Model (DSI, EPFL, …..) 113
16.1.9.1 Dublin Core Metadata 113
16.1.10 Examples of AXMEDIS Objects 119

17 AXINFO (DSI) 126

18 AXMEDIS TOOL FINGERPRINT (DSI, FUPF) 145

19 AXMEDIS PROTECTION INFO (DSI, FHGIGD) 148

20 PROTECTION TOOL DESCRIPTION (DSI) 149

21 RIGHTS AND ENFORCEMENT (FUPF, DSI, ALL) 152

21.1 DESCRIPTION OF TYPICAL CONTENT MANIPULATION 152
21.1.1 General issue on distinguish Adapt and Modify intents 152
21.2 EXAMPLES OF AXMEDIS OBJECT MANIPULATION 152
21.2.1 Adding to root level 152
21.2.2 Adding to nested level 154
21.2.3 Transformation of basic objects 154
21.2.4 Transformation of objects in a composition 155
21.2.5 Deletion of objects from a composition (from root level or nested) 155
21.2.6 Copying of objects from a composition to another 156
21.2.7 Moving of objects from a composition to another 158
21.2.8 Playing/rendering of objects (basic, composite) 158
21.2.9 Manipulation of object metadata 159
21.3 MAPPING RIGHTS ON USER ACTION 159

1 Executive Summary and Report Scope

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

DE number	Deliverable title	respons ible
DE3.1.2.2.1	Specification of General Aspects of AXMEDIS framework, first update of DE3.1.2 part A AXMEDIS-DE3-1-2-2-1-Spec-of-AX-Gen-Asp-of-AXMEDIS-framework-upA-v1-0.doc	DSI
DE3.1.2.2.2	Specification of AXMEDIS Command Manager, first update of DE3.1.2 part B AXMEDIS- DE3-1-2-2-2-Spec-of-AX-Cmd-Man-upB-v1-0.doc	DSI
DE3.1.2.2.3	Specification of AXMEDIS Object Manager and Protection Processor, first update of DE3.1.2 part B AXMEDIS-DE3-1-2-2-3-Spec-of-AXOM-and-ProtProc-upB-v1-0.doc	DSI
DE3.1.2.2.4	Specification of AXMEDIS Editors and Viewers, first update of DE3.1.2 part B AXMEDIS-DE3-1-2-2-4-Spec-of-AX-Editors-and-Viewers-upB-v1-0.doc	DSI
DE3.1.2.2.5	Specification of External AXMEDIS Editors/Viewers and Players, first update of DE3.1.2 part B AXMEDIS-DE3-1-2-2-5-Spec-of-External-Editors-Viewers-Players-upB-v1-0.doc	EPFL
DE3.1.2.2.6	Specification of AXMEDIS Content Processing, first update of DE3.1.2 part C AXMEDIS-DE3-1-2-2-6-Spec-of-AX-Content-Processing-upC-v1-0.doc	DSI
DE3.1.2.2.7	Specification of AXMEDIS External Processing Algorithms AXMEDIS-DE3-1-2-2-7-Spec-of-AX-External-Processing-Algorithms-v1-0.doc	FHGIGD
DE3.1.2.2.8	Specification of AXMEDIS CMS Crawling Capabilities, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-8-Spec-of-AX-CMS-Crawling-Capab-v1-0.doc	DSI
DE3.1.2.2.9	Specification of AXMEDIS database and query support, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-9-Spec-of-AX-database-and-query-support-v1-0.doc	EXITEC H
DE3.1.2.2.10	Specification of AXMEDIS P2P tools, AXEPTool and AXMEDIS, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-10-Spec-of-AXEPTool-and-AXMEDIA-tools-v1-0.doc	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	EXITEC H

1.1 This document concerns (DSI)

AXMEDIS Object Manager and Protection Processor.

AXMEDIS Object Manager, so called AXOM, is the coordinator of all other modules used by or built in AXMEDIS Editor. Coordination activities it is important to permit development of others AXMEDIS Editor modules almost independently each other. AXMEDIS Object Manager guarantees DRM rules respect on AXMEDIS object manipulations.
AXMEDIS Object Manager works in respect of AXMEDIS Data Model Support and it fundamentally is composed by three modules:

- **AXMEDIS Command Manager** which is the real interface for processing content models as the AXMEDIS Data Model and protected according to the tools of the AXMEDIS Protection processor
- **AXMEDIS Data Model Support** to model the AXMEDIS objects according to MPEG-21 and additional requirements identified
- **AXMEDIS Protection Processor** to protect and processing registration, certification, and to protected and unprotect digital resources according to a dynamic mechanism similar to that of IPMP of MPEG21.

In particular, AXMEDIS Data Model Support can be decomposed in four modules and two formats:

- **MPEG-21 Object Model** allows to represent a MPEG-21 document as software object model
- **MPEG-21 Loader** allows to load an MPEG-21 document
- **MPEG-21 Saver** allows to save a software object model to a MPEG-21 document
- **AXMEDIS Object Model** allows to represent an AXMEDIS document as a software object model on the basis of the MPEG-21 Object Model
- **AXMEDIS Data Model** defines the structure and the information of an AXMEDIS Object
- **AXInfo** defines the structure of the AXMEDIS B2B information

Protection Processor has mainly four tasks:

1. To register and certify an AXMEDIS tool containing the AXOM, e.g. editor, player, engine, etc.
2. To control software which uses sensible content and does not contain AXOM, e.g. plug-ins for fingerprint
3. To reveal attacks during tool execution, e.g. code debugging
4. To protect and un-protect elements of AXMEDIS object

In the following those aspects will be described and solutions are proposed for them. After that, class implementation and interaction will be described.

1.2 List of Modules or Executable Tools Specified in this document (DSI)

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
MPEG-21 Object Model		MPEG-21 DIDL; MPEG-21 IPMP
MPEG-21 Loader		MPEG-21 DIDL; MPEG-21 IPMP
MPEG-21 Saver		MPEG-21 DIDL; MPEG-21 IPMP
AXMEDIS Object Model		Dublin Core
AXMEDIS Object Pre-processor and Post-processor		W3C XInclude, MPEG-21 Binary Format
Protection Processor		MPEG-21 IPMP; X.509 v3
Encryption/Decryption Support		

Compress/uncompress Support		
Scramble/Descramble Support		
MPEG-21 DIBO		ISO/IEC FDIS 21000-10:2005(E) - MPEG-21 DIP
MPEG-21 DIM		ISO/IEC FDIS 21000-10:2005(E) - MPEG-21 DIP
MPEG-21 DIA Processing		ISO/IEC FDIS 21000-7:2005(E) - MPEG-21 DIA

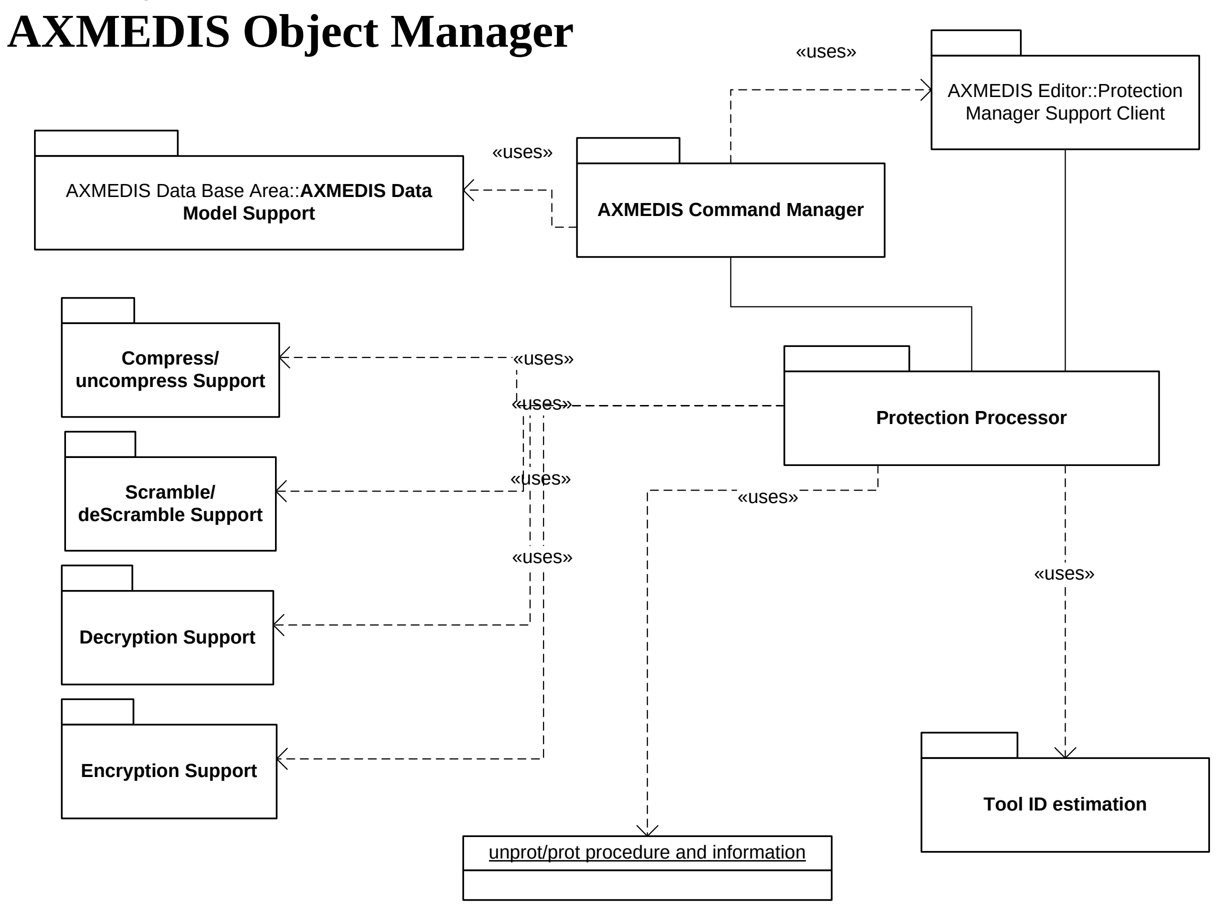
1.3 List of Formats Specified in this document (DSI)

A format can be (i) an XML content file for modelling 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
MPEG-21 DIA		MPEG-21 DIA
AXMEDIS Data Model		MPEG-21 DIDL; MPEG-21 IPMP
AXInfo		
AXMEDIS Tool Fingerprint		
AXMEDIS Protection Info		MPEG-21 IPMP
Protection Tool description		

2 General architecture and relationships among the modules produced (DSI, All)

The following figure sketch out the general architecture of the AXMEDIS Object Manager and Protection Processor togethers


AXMEDIS Object Manager
«uses»
AXMEDIS Editor::Protection Manager Support Client
«uses»
AXMEDIS Data Base Area::AXMEDIS Data Model Support
AXMEDIS Command Manager
Compress/ uncompress Support
«uses»
«uses»
Protection Processor
Scramble/ deScramble Support
«uses»
«uses»
«uses»
«uses»
Decryption Support
Encryption Support
Tool ID estimation
unprot/prot procedure and information

3 MPEG-21 Object Model (DSI)

Module/Tool Profile		
MPEG-21 Object Model		
Responsible Name	Andrea Vallotti, Davide Rogai	
Responsible Partner	DSI	
Status (proposed/approved)	Proposed	
Implemented/not implemented	Implemented	
Status of the implementation	In refinement	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows XP; Linux; Windows Mobile	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/project/axom/dimodel https://cvs.axmedis.org/repos/Framework/source/axom/dimodel https://cvs.axmedis.org/repos/Framework/include/axom/dimodel	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/repos/Framework/bin/axom/	
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)	Present	
Test cases location	https://cvs.axmedis.org/repos/Framework/doc/test/axom/	
Usage of the AXMEDIS configuration manager (yes/no)	No	
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.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

3.1 General Description of the Module

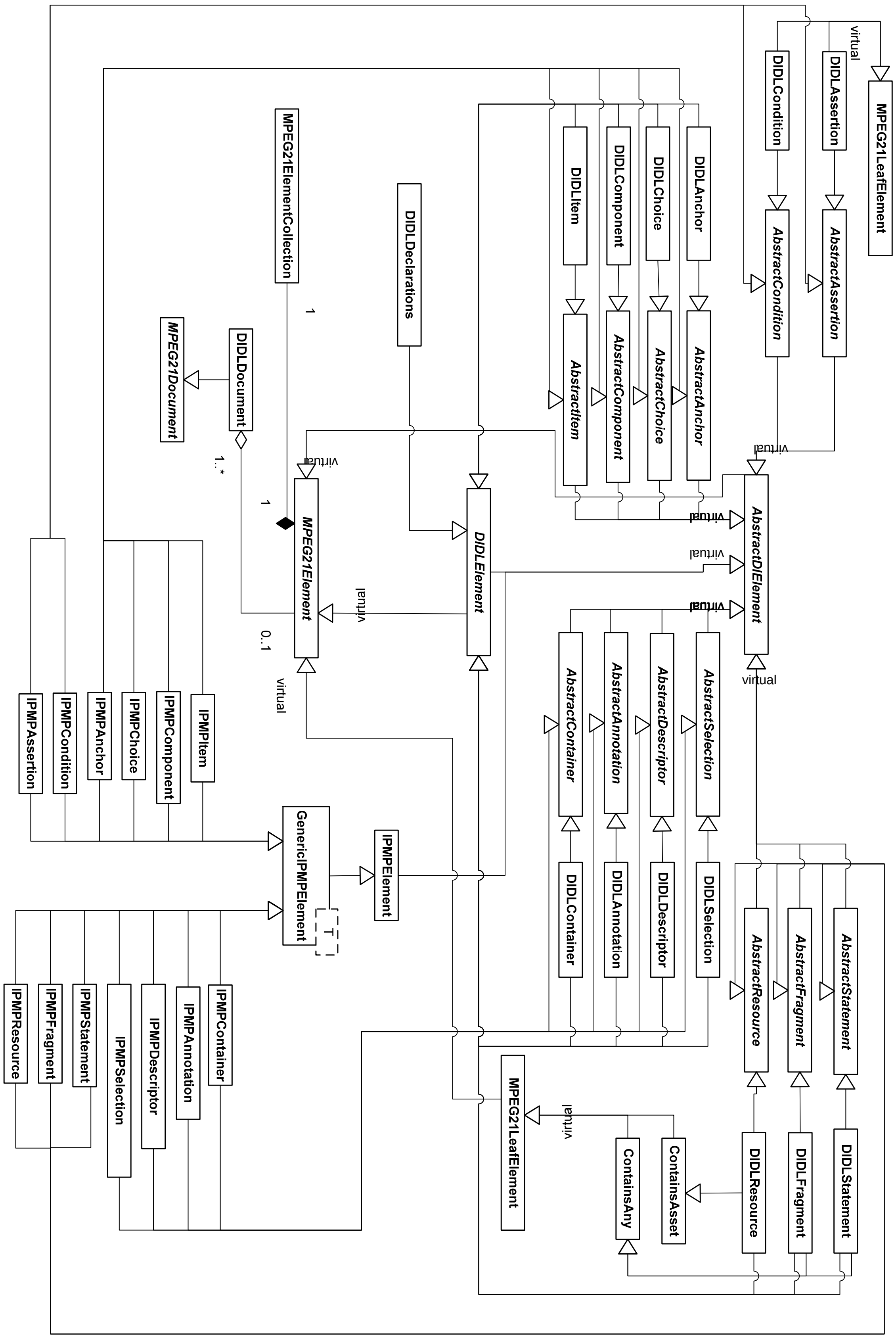
The MPEG-21 Object Model provides the means to represent an MPEG-21 Digital Item as a software object model. That is, a DI is represented as a tree of class instances. Each class represent one of the elements defined in the MPEG-21 standard. In particular, the elements of MPEG-21 DID and IPMP has been modelled since these parts of the standard are fundamental in order to represent protected and clear-text DIs.
This module provides all those features needed in order to completely handling and managing the DI model, including:

- Insertion, deletion and modification of model, model elements, and model element's attributes
- Modularity and expandability of model representation.
- Managing of inclusion functionalities and post parsing inclusions resolving and loading.
- Validation of model representation.

Items representation doesn't stop to DI hierarchy but includes even DII hierarchy modelling.

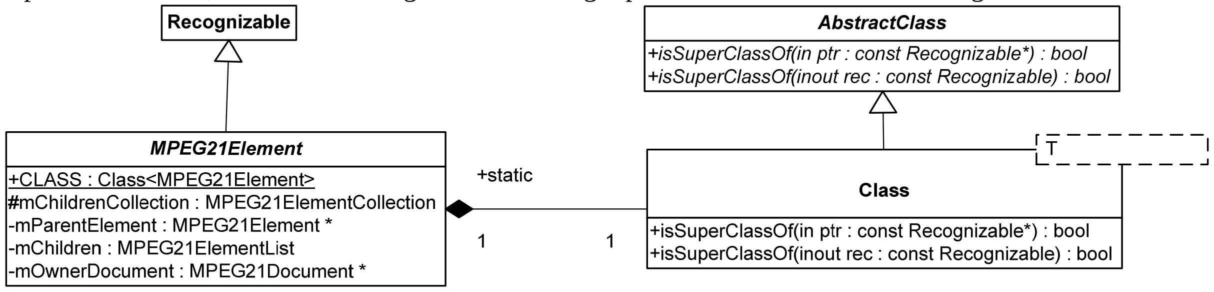
3.2 Module Design in terms of Classes

The class diagram below gives an overview of the MPEG-21 Object Model. IPMP elements are fully integrated in the representation tree of DI. This structure also allow insertion and managing of new hierarchies, like DII hierarchy.

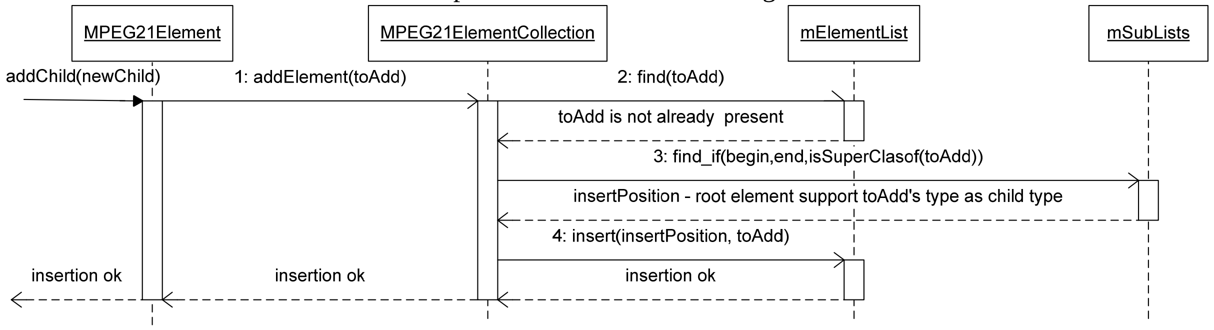

MPEG21LeafElement
DIDLAssertion
DIDLCondition
virtual
AbstractAssertion
AbstractCondition
DIDLAnchor
DIDLChoice
DIDLComponent
DIDLItem
AbstractAnchor
AbstractChoice
AbstractComponent
AbstractItem
DIDLDeclarations
MPEG21ElementCollection
1
DIDLDocument
MPEG21Document
1..*
1
virtual
MPEG21Element
DIDLElement
virtual
AbstractDIElement
virtual
virtual
virtual
virtual
0..1
virtual
AbstractSelection
AbstractDescriptor
AbstractAnnotation
AbstractContainer
DIDLSelection
DIDLDescriptor
DIDLAnnotation
DIDLContainer
IPMPItem
IPMPComponent
IPMPChoice
IPMPAnchor
IPMPCondition
IPMPAssertion
GenericIPMPElement
T
IPMPElement
AbstractStatement
AbstractFragment
AbstractResource
DIDLStatement
DIDLFragment
DIDLResource
ContainsAsset
ContainsAny
virtual
MPEG21LeafElement
IPMPContainer
IPMPAnnotation
IPMPDescriptor
IPMPSelection
IPMPStatement
IPMPFragment
IPMPResource

3.2.1 DI's Object Model Capabilities Overview

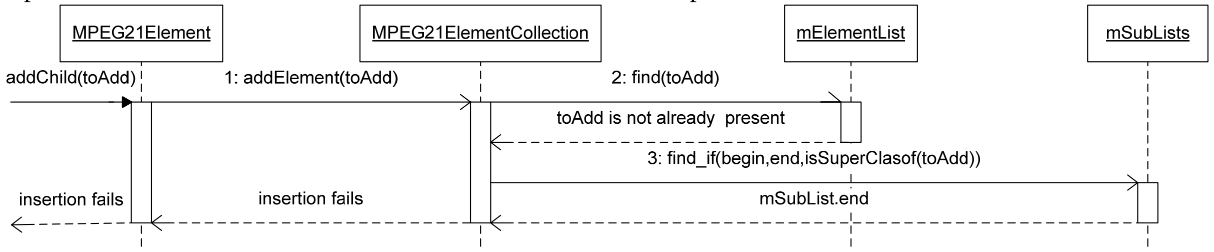
A fundamental role in module architecture is carried out by MPEG21ElementCollection. This class manages representation tree structure of DI, it implements type checking on elements, element ordered insertion, deletion, retrieving and representation visit.
Next diagram shows a deepest look to type checking implementation. All element classes in the model contains a static instance of Class element to specify their types. This instance provide type checking methods needed by MPEG21ElementCollection to allow or deny insertion in a particular position of representation tree, and then enabling it to find the right place in the tree for an incoming new element.


Recognizable
AbstractClass
+isSuperClassOf(in ptr : const Recognizable*) : bool
+isSuperClassOf(inout rec : const Recognizable) : bool
MPEG21Element
+CLASS : Class<MPEG21Element>
#mChildrenCollection : MPEG21ElementCollection
-mParentElement : MPEG21Element *
-mChildren : MPEG21ElementList
-mOwnerDocument : MPEG21Document *
+static
1
1
T
Class
+isSuperClassOf(in ptr : const Recognizable*) : bool
+isSuperClassOf(inout rec : const Recognizable) : bool

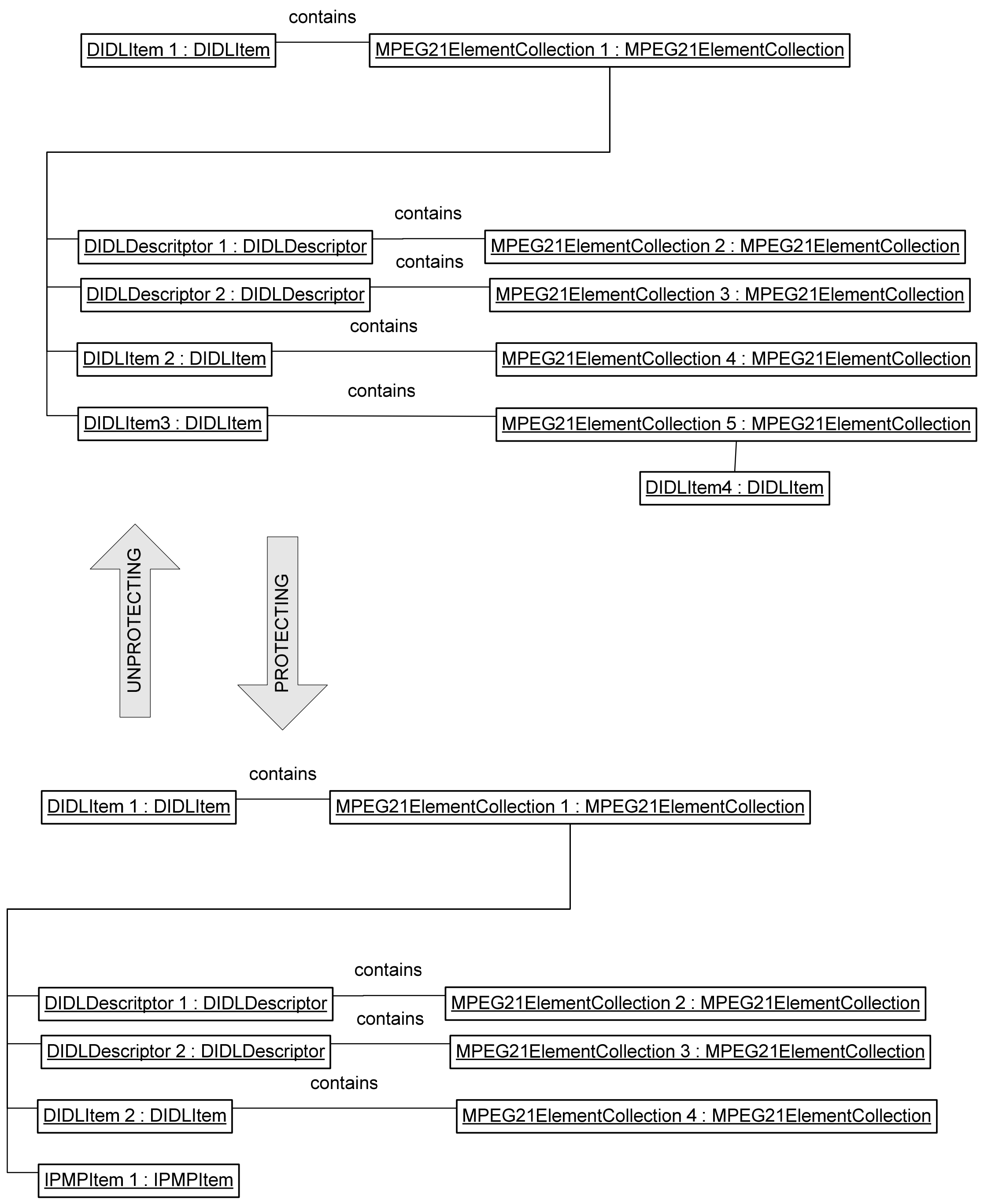
MPEG21ElementCollection could, using functionalities introduced by Class static element, determine if and where a new element is inserted in the representation of DI as next figure shows.

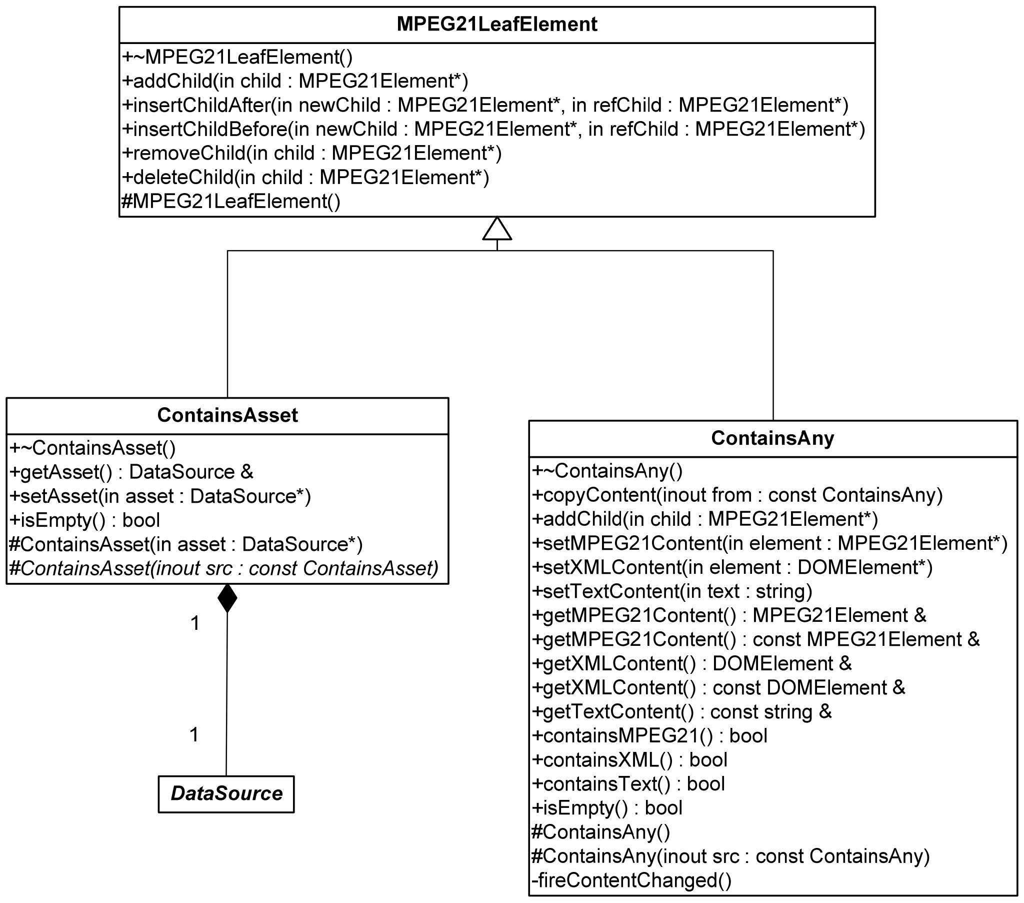

MPEG21Element
MPEG21ElementCollection
mElementList
mSubLists
addChild(newChild)
1: addElement(toAdd)
2: find(toAdd)
toAdd is not already present
3: find_if(begin,end,isSuperClasof(toAdd))
insertPosition - root element support toAdd's type as child type
4: insert(insertPosition, toAdd)
insertion ok
insertion ok
insertion ok

If insertion is not allowed, e.g. in case where the given element is not allowed to be inserted as a child of the target element, type checking allow rejection of insertion operation, granting consistency between representation of DI and XML Schema defined in MPEG-21 specification.



MPEG21Element collection provide a flexible management of representation tree. A IPMPItem could easily substitute a DIDLItem in a representation, e.g. by means of a protection operation. This is allowed since IPMPItem and DIDLItem has the same AbstractItem static member inside.
To manage those crititcal elements which carries the content for the representation a special behaviour is needed. We have to point out that those element are leaf elements for the representation, so we can easly group them with a common class MPEG21LeafElement. Another distinction we could do is about content these element carries: this may be text or metadata or may be some kind of multimedia content. Object model has to separate managing of these two cases .Classes ContainsAny for metadata and ContainsAsset for multimedia contents solve those problems and offers functionalities to maintain content information or references multimedia data. The latter indeed has not to be loaded in memory in many use cases.


contains
DIDLItem 1 : DIDLItem
MPEG21ElementCollection 1 : MPEG21ElementCollection
DIDLDescritptor 1 : DIDLDescriptor
MPEG21ElementCollection 2 : MPEG21ElementCollection
DIDLDescriptor 2 : DIDLDescriptor
MPEG21ElementCollection 3 : MPEG21ElementCollection
DIDLItem 2 : DIDLItem
MPEG21ElementCollection 4 : MPEG21ElementCollection
DIDLItem3 : DIDLItem
MPEG21ElementCollection 5 : MPEG21ElementCollection
DIDLItem4 : DIDLItem
UNPROTECTING
PROTECTING
IPMPItem 1 : IPMPItem


MPEG21LeafElement
+~MPEG21LeafElement()
+addChild(in child : MPEG21Element*)
+insertChildAfter(in newChild : MPEG21Element*, in refChild : MPEG21Element*)
+insertChildBefore(in newChild : MPEG21Element*, in refChild : MPEG21Element*)
+removeChild(in child : MPEG21Element*)
+deleteChild(in child : MPEG21Element*)
#MPEG21LeafElement()
ContainsAsset
+~ContainsAsset()
+getAsset() : DataSource &
+setAsset(in asset : DataSource*)
+isEmpty() : bool
#ContainsAsset(in asset : DataSource*)
#ContainsAsset(inout src : const ContainsAsset)
1
1
DataSource
ContainsAny
+~ContainsAny()
+copyContent(inout from : const ContainsAny)
+addChild(in child : MPEG21Element*)
+setMPEG21Content(in element : MPEG21Element*)
+setXMLContent(in element : DOMElement*)
+setTextContent(in text : string)
+getMPEG21Content() : MPEG21Element &
+getMPEG21Content() : const MPEG21Element &
+getXMLContent() : DOMElement &
+getXMLContent() : const DOMElement &
+getTextContent() : const string &
+containsMPEG21() : bool
+containsXML() : bool
+containsText() : bool
+isEmpty() : bool
#ContainsAny()
#ContainsAny(inout src : const ContainsAny)
-fireContentChanged()

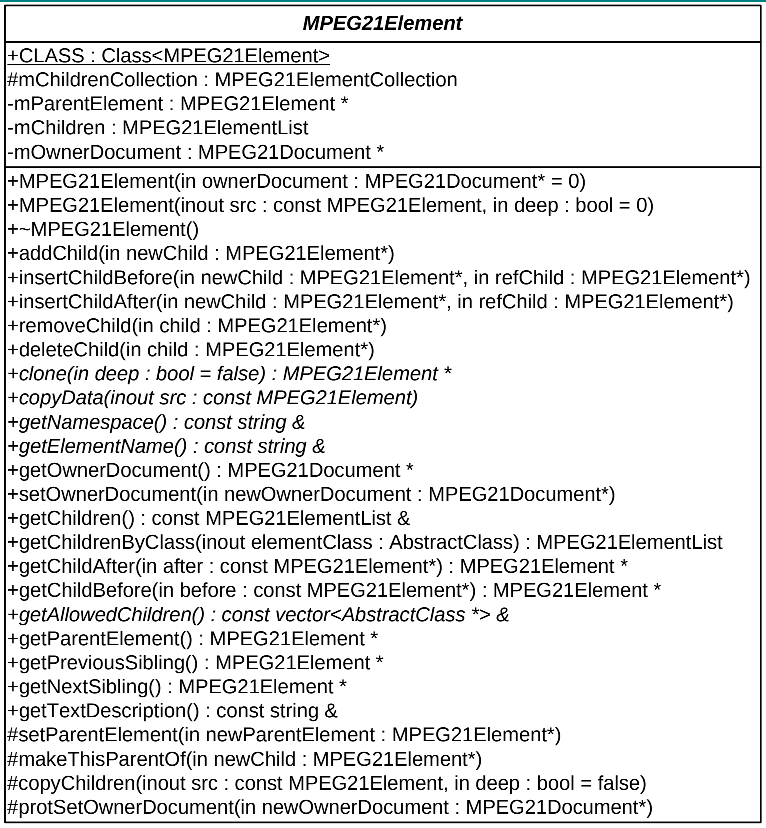
3.2.2 Classes and Methods Overview

MPEG21Element

This class acts as interface for all hierarchy elements. All methods defined and implemented by this class are virtual and could be overridden by inheriting elements.

MPEG21Element – class methods
addCild – insertChildBefore – insertChildAfter – removeChild – deleteChild
These methods are used to manage structure. All these methods match an appropriate method in MPEG21ElementCollection and acts as an interface for functionalities implemented by this class.
getChildren – get ChildrenByClass getChildAfter – getChildBefore
These methods are used to retrive children elements linked in MPEG21ElementCollection. All these methods uses MPEG21ElementCollection functionalities to retrive elements or groups of elements from representation tree
clone , copyData, copyChildren
These virtual methods are implemented to provide element copy functionalities. These methods are specialized by local elements implementation
MPEG21Element - ~MPEG21Element – MPEG21Element(MPEG21Element)
Constructor, destructor and copy constructor.
getNamespace
Retruns a string containing the element’s namespace.
getElementName
Returns a string containing the name of the element.
getParentElement setParentElement
Returns a pointer to element’s parent or set a parent for the elment
getNextSibiling – getPreviousSibiling
Returns a pointer to next or previous parent’s child element

getOwnerDocument – setOwnerDocument – protSetOwnerDocument
Returns or set a link to the document that contains the element.
getAllowedChildren
This method returns a vector containing the allowed children types for the element.
getTextDescription
Returns a textual description of the element
makeThisParentOf
Makes this element parent of the target element

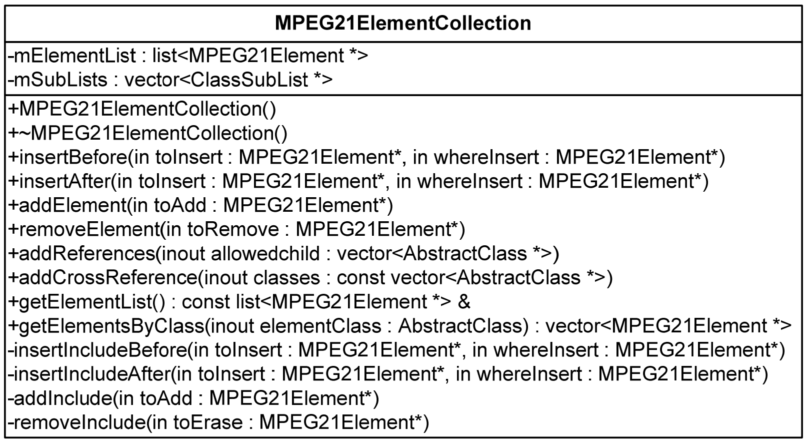

MPEG21Element

+CLASS : Class<MPEG21Element>
#mChildrenCollection : MPEG21ElementCollection
-mParentElement : MPEG21Element *
-mChildren : MPEG21ElementList
-mOwnerDocument : MPEG21Document *

+MPEG21Element(in ownerDocument : MPEG21Document* = 0)
+MPEG21Element(inout src : const MPEG21Element, in deep : bool = 0)
+~MPEG21Element()
+addChild(in newChild : MPEG21Element*)
+insertChildBefore(in newChild : MPEG21Element*, in refChild : MPEG21Element*)
+insertChildAfter(in newChild : MPEG21Element*, in refChild : MPEG21Element*)
+removeChild(in child : MPEG21Element*)
+deleteChild(in child : MPEG21Element*)
*+clone(in deep : bool = false) : MPEG21Element **
+copyData(inout src : const MPEG21Element)
+getNamespace() : const string &
+getElementName() : const string &
+getOwnerDocument() : MPEG21Document *
+setOwnerDocument(in newOwnerDocument : MPEG21Document*)
+getChildren() : const MPEG21ElementList &
+getChildrenByClass(inout elementClass : AbstractClass) : MPEG21ElementList
+getChildAfter(in after : const MPEG21Element*) : MPEG21Element *
+getChildBefore(in before : const MPEG21Element*) : MPEG21Element *
*+getAllowedChildren() : const vector<AbstractClass *> &*
+getParentElement() : MPEG21Element *
+getPreviousSibling() : MPEG21Element *
+getNextSibling() : MPEG21Element *
+getTextDescription() : const string &
#setParentElement(in newParentElement : MPEG21Element*)
#makeThisParentOf(in newChild : MPEG21Element*)
#copyChildren(inout src : const MPEG21Element, in deep : bool = false)
#protSetOwnerDocument(in newOwnerDocument : MPEG21Document*)

MPEG21ElementCollection

This class defines methods to manage Object Model representation tree
Any MPEG21Element include an instance of this class.


MPEG21ElementCollection

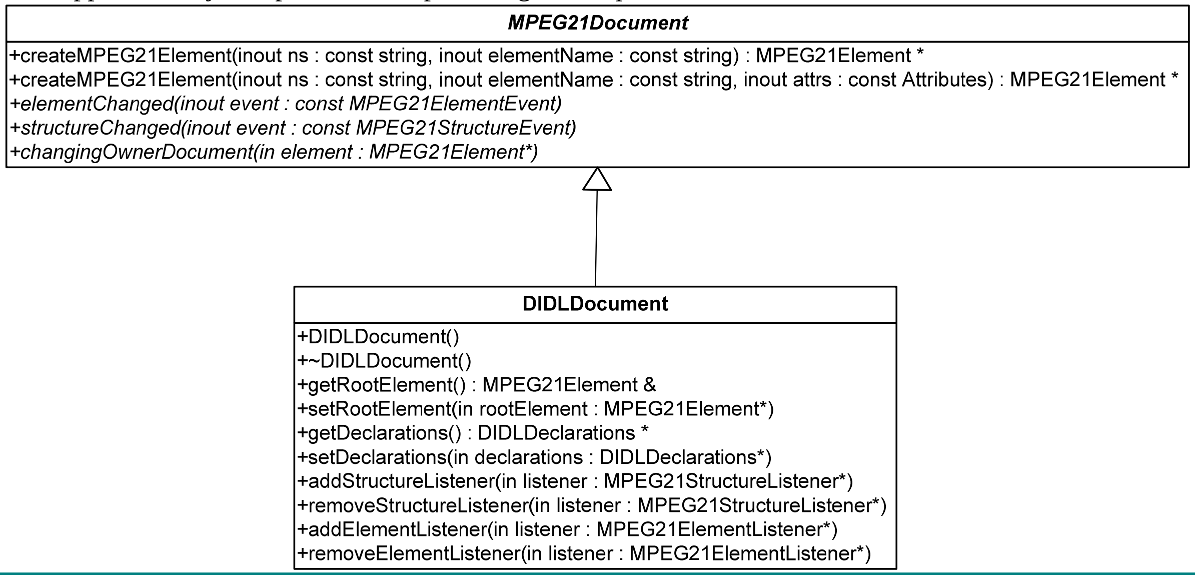
-mElementList : list<MPEG21Element *>
-mSubLists : vector<ClassSubList *>

+MPEG21ElementCollection()
+~MPEG21ElementCollection()
+insertBefore(in toInsert : MPEG21Element*, in whereInsert : MPEG21Element*)
+insertAfter(in toInsert : MPEG21Element*, in whereInsert : MPEG21Element*)
+addElement(in toAdd : MPEG21Element*)
+removeElement(in toRemove : MPEG21Element*)
+addReferences(inout allowedchild : vector<AbstractClass *>)
+addCrossReference(inout classes : const vector<AbstractClass *>)
+getElementList() : const list<MPEG21Element *> &
+getElementsByClass(inout elementClass : AbstractClass) : vector<MPEG21Element *>
-insertIncludeBefore(in toInsert : MPEG21Element*, in whereInsert : MPEG21Element*)
-insertIncludeAfter(in toInsert : MPEG21Element*, in whereInsert : MPEG21Element*)
-addInclude(in toAdd : MPEG21Element*)
-removeInclude(in toErase : MPEG21Element*)

MPEG21ElementCollection – class methods

MPEG21ElementCollection - ~MPEG21ElementCollection
Class constructor and destructor
insertBefore – insertAfter
Insert an element in the element list in a position specified by the whereInsert element.. Throws an exception if input position is not in accordance with inserted element or if input element is not allowed or already present in the list
addElement
Retrive the right insertion position for input element and insert it as last element of his own type in the elementList
removeElement
Removes target element from the elementList. Throws an exception if the element is not present in the collection
addReferences – addCrossReferences
Initialize the collection adding allowed childrens entries to mSubLists and setting list pointers to represent a children placement that satisfies Digital Item Schema definition of the element
getElementList – getElementListByClass
Returns entries contained in the elementList.
insertIncludeBefore – insertIncludeAfter – addInclude – removeInclude
Manages XInclude elements allowing insertion and removal from the collection

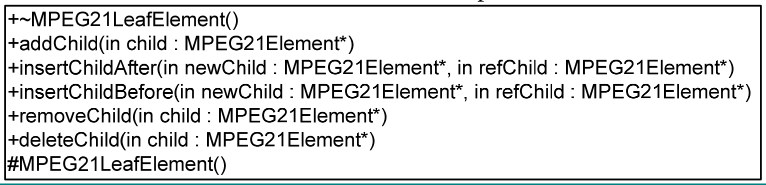
MPEG21Document and *DIDLDocument*
These classes acts as interface between object model and upper level applications. Methods implemented by these classes allow communication about document structure or element modifications. These class acts even as wrappers for object representation, providing access point to DI.


MPEG21Document
+createMPEG21Element(inout ns : const string, inout elementName : const string) : MPEG21Element *
+createMPEG21Element(inout ns : const string, inout elementName : const string, inout attrs : const Attributes) : MPEG21Element *
+elementChanged(inout event : const MPEG21ElementEvent)
+structureChanged(inout event : const MPEG21StructureEvent)
+changingOwnerDocument(in element : MPEG21Element)*
DIDLDocument
+DIDLDocument()
+~DIDLDocument()
+getRootElement() : MPEG21Element &
+setRootElement(in rootElement : MPEG21Element*)
+getDeclarations() : DIDLDeclarations *
+setDeclarations(in declarations : DIDLDeclarations*)
+addStructureListener(in listener : MPEG21StructureListener*)
+removeStructureListener(in listener : MPEG21StructureListener*)
+addElementListener(in listener : MPEG21ElementListener*)
+removeElementListener(in listener : MPEG21ElementListener*)

DIDLDocument – class methods
DIDLDocument - ~DIDLDocument
Class constructor and destructor
getRootElement – setRootElement
Get or set root representation element pointer in the document representation
addStructureListener – removeStructureListener
Set or remove the class Listener who has to manage structure changes events in the model for upper communication
addElementListener – removElementListener
Set or remove the class Listener who has to manage elements changes events in the model for upper communication
createMPEG21Element

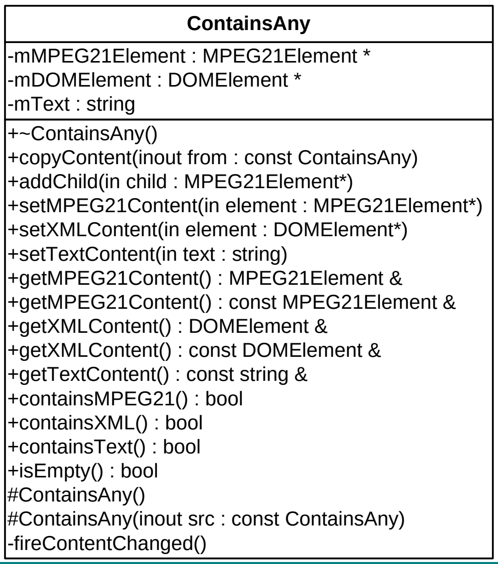
Provide interface about element creation for upper level application – See factory classes in MPEG-21 Loader module in this document for further information
elementChanged – structureChanged – changingOwnerDocument
Communicate changes from model to MPEG21Document

MPEG21LeafElement
Offer a different interface for those classes who represents representation tree leafs.
Children managing methods are overridden and throws an exception if invoked.



```
+~MPEG21LeafElement()
+addChild(in child : MPEG21Element*)
+insertChildAfter(in newChild : MPEG21Element*, in refChild : MPEG21Element*)
+insertChildBefore(in newChild : MPEG21Element*, in refChild : MPEG21Element*)
+removeChild(in child : MPEG21Element*)
+deleteChild(in child : MPEG21Element*)
#MPEG21LeafElement()
```

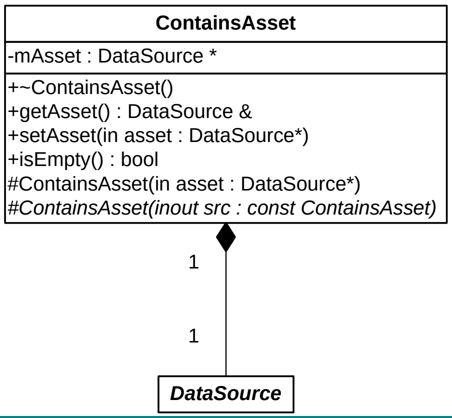
MPEG21LeafElement – class methods
MPEG21LeafElement - ~MPEG21LeafElement
Class constructor and destructor
addChild – insertChildAfter – insertChildBefore – removeChild – deleteChild
Overrides MPEG21ElementCollection methods. These methods throws an exception when a child operation is tried on a leaf element



```
ContainsAny
-mMPEG21Element : MPEG21Element *
-mDOMElement : DOMElement *
-mText : string
+~ContainsAny()
+copyContent(inout from : const ContainsAny)
+addChild(in child : MPEG21Element*)
+setMPEG21Content(in element : MPEG21Element*)
+setXMLContent(in element : DOMElement*)
+setTextContent(in text : string)
+getMPEG21Content() : MPEG21Element &
+getMPEG21Content() : const MPEG21Element &
+getXMLContent() : DOMElement &
+getXMLContent() : const DOMElement &
+getTextContent() : const string &
+containsMPEG21() : bool
+containsXML() : bool
+containsText() : bool
+isEmpty() : bool
#ContainsAny()
#ContainsAny(inout src : const ContainsAny)
-fireContentChanged()
```

ContainsAny – class methods
ContainsAny - ~ContainsAny
Class constructor copy constructor and destructor
addChild
Overrides MPEG21LeafElement::addChild.Check the content to be added. If is MPEG21Content and the element contains no element of such type it calls setMPEG21Content method
getMPEG21Content – setMPEG21Content
Set or get the MPEG-21 content of this element pointed by mMPEG21Element
getXMLContent – setXMLContent
Set or get the generic XML content of this element pointed by mDOMElement
getTextContent – setTextContent
Set or get the generic Text content of this element pointed by mText
containsMPEG21 – containsText – containsXML
Returns true if the specified element is present.
copyContent
Copy the content of this element.

isEmpty
Returns true if the element pointers are empty.
fireContentChanged
Communicate to owner document if the content of this element is changed


ContainsAsset
-mAsset : DataSource *
+~ContainsAsset()
+getAsset() : DataSource &
+setAsset(in asset : DataSource*)
+isEmpty() : bool
#ContainsAsset(in asset : DataSource*)
#ContainsAsset(inout src : const ContainsAsset)
1
1
DataSource

ContainsAsset – class methods
ContainsAsset - ~ContainsAsset
Class constructor, copy constructor, destructor
getAsset – setAsset
Get or set asset for this element. Asset will contain a pointer to a DataSource element
isEmpty
Returns true if the element has no Asset defined.

3.3 Examples of usage

Following code shows how the elements of the model are used. In the example an item is created and a descriptor child is added. Then the item is set as root MPEG-21 Element of the new created document.

```
DIDLDocument doc;
DIDLItem* item=new DIDLItem;
DIDLDescriptor* desc=new DIDLDescriptor;
item->setID("ax1");
item->addChild(desc);
item->getChildren()[0];    //return desc
doc.setRootElement(item);  //set item as root element of doc
```

3.4 Integration and compilation issues

MPEG-21 Module itself depends from Xerces C++ in some data structure definitions used in MPEG-21 elements
The following table summarizes the needed library in order to use MPEG-21 module

Name	OS/Platform		Library file	Description
	Windows/PC	Linux/PC		
Xerces C++	X	X	xerces-c_2.lib	Provides data support structures for XML parsing
Common	X	X	common.lib	

3.5 Errors reported and that may occur

Error signals through exceptions throwing.

Error code	Description and rationales
0	A requested element is not found
1	A referred element is not found
2	An inserting element is already present
3	The given element has wrong type thus cannot be inserted
4	No element can be inserted as a child of selected element
5	Input element is not an element of specified type to clone

4 MPEG-21 Loader (DSI)

Module/Tool Profile		
MPEG-21 Loader		
Responsible Name	Andrea Vallotti, Davide Rogai	
Responsible Partner	DSI	
Status (proposed/approved)		
Implemented/not implemented	implemented	
Status of the implementation		
Executable or Library/module (Support)	Static library	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows, Linux	
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)	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.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

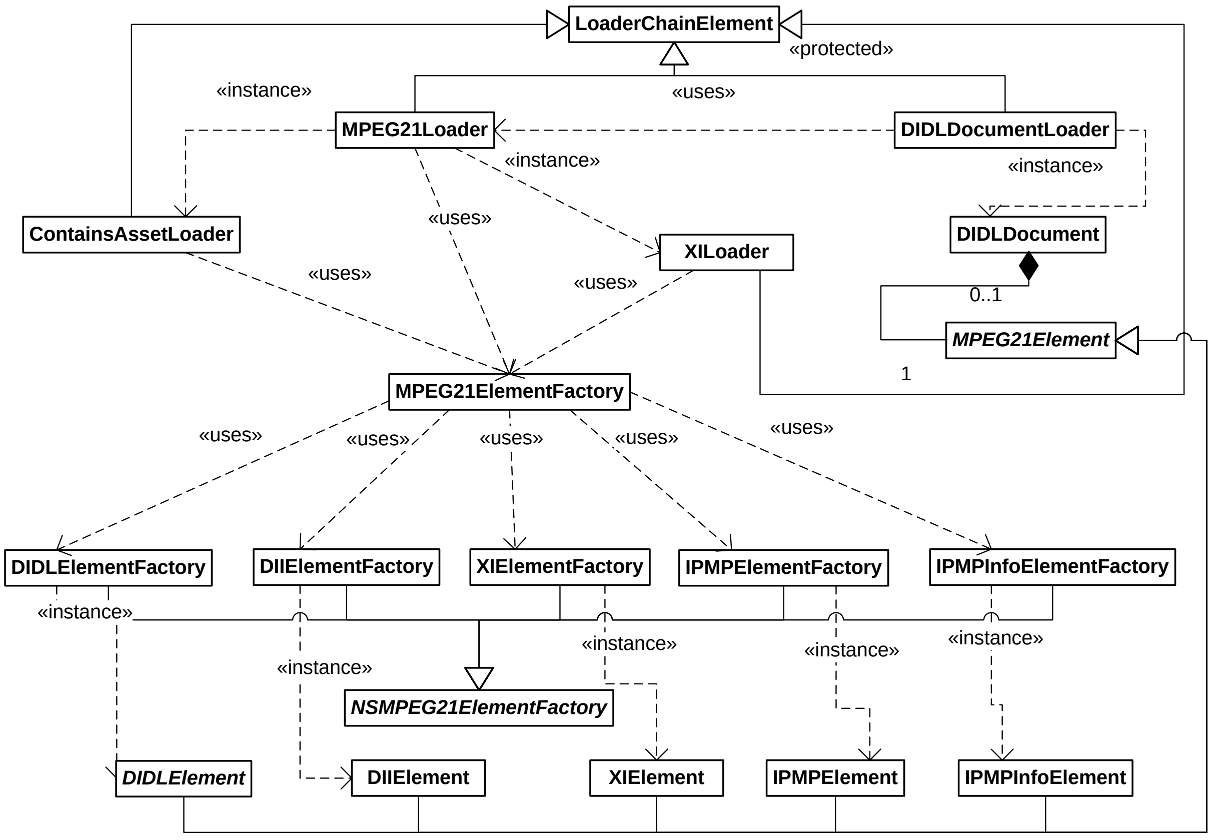
4.1 General Description of the Module

The MPEG-21 Loader allows to load in memory an MPEG-21 DI. In particular, the Loader reads data from a stream of bytes and it creates a software model of the DI (see MPEG-21 Object Model). In order to load a DI, the first step is processing the XML or BIN format. The resources of the object could be too big in order to be allocated as a whole in memory. A driven loading is the only useful mean in order to get the benefits of a memory representation of the DI, without the occupation of (potentially) giga bytes of data (e.g. a MPEG2 video resource of a movie).
The DOM implementation expose a node filtering feature that can reject a node after the DOM builder has loaded it; this is unacceptable for the loader purpose, because the resource could be only removed after it has occupied the whole memory and the loader has caused a fatal error on the system.
The solution is to intercept the "resource" tags occurrences before they are loaded in memory; something that the SAX2 interface can manage. In the object model the resource includes a way to retrieve the content stream still located on the file system. The new resource element must redirect access to the effective content location (stream to the file/storage position) during access functions (play, adapting, copying…).
It is even important to understand that, for the same reason of memory saving, an XInclude element, introduced in MPEG-21 specification as a hook to insert references in a DI XML Document, must not be loaded as a whole, but it has to be inserted in the object model structure as an equivalent entity. Since MPEG-21 awaits the use of a pre-processor to resolve XInclude references and expand target content in the document before parsing, MPEG-21 Loader has to skip these operations and provide a different behaviour to manage this case.
The IPMP Content of an IPMPItem (it is used to protect content inside DIs) is treated in the same way; it could be a big amount of data and it will be not loaded in the object model. The part in the object model that refer to a stream on the file will be integrated if requested by the user actions.

All these issues drives to a design that takes into account different implementations for different parsed elements. The concept of factory is useful to work out all troubles.

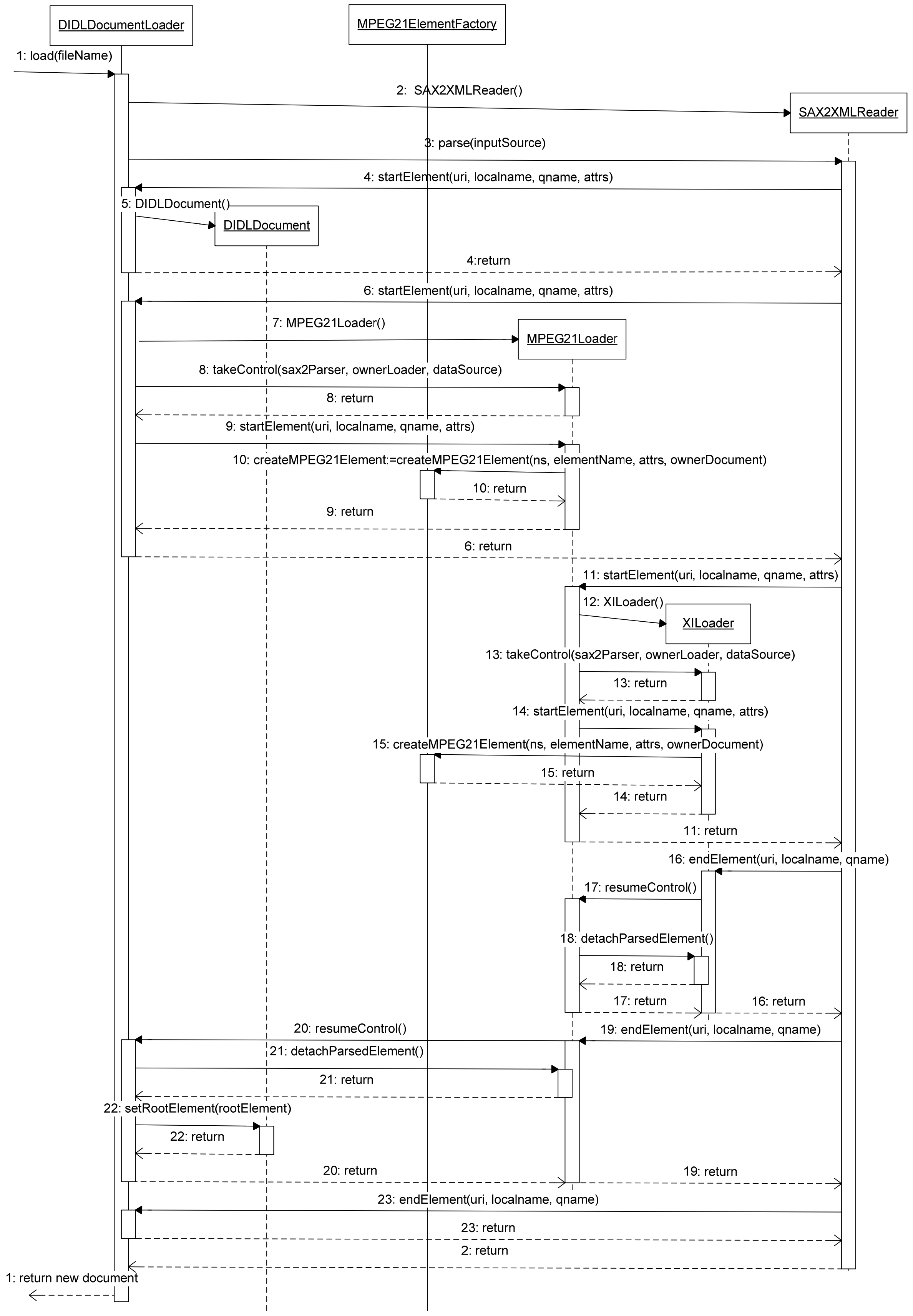
4.2 Module Design in terms of Classes

Next picture shows design of loader module. *DIDLDocumentLoader* represents the interface through which a user of the model (or a upper level application) may create an MPEG-21Object Model from an XML representation of the same. Whether *DIDLDocumentLoader* attends initialization and coordination in parsing activity and creation of a new *DIDLDocument* element to wrap information obtained, *MPEG21Loader* implements parsing methods. The latter has been implemented as a content handler for a SAX2 parser, as defined by the W3C. In order to create new elements, *MPEG21Loader* uses *MPEG21ElementFactory* architecture. This group of classes implements the pattern Factory for each hierarchy foreseen by Object Model, MPEG21ElementFactory coordinates creation calling the right factory instance on the base of information received from MPEG21Loader. New elements are added as components of the tree and wrapped in *DIDLDocument*..



4.2.1 MPEG-21 Loader Capabilities Overview

Next sequence diagram shows deeply loading chain operations and how the module will come to a complete object representation from an input XML document. file. *SAX2XMLReader* is a SAX2 parser complaining with W3C directives.

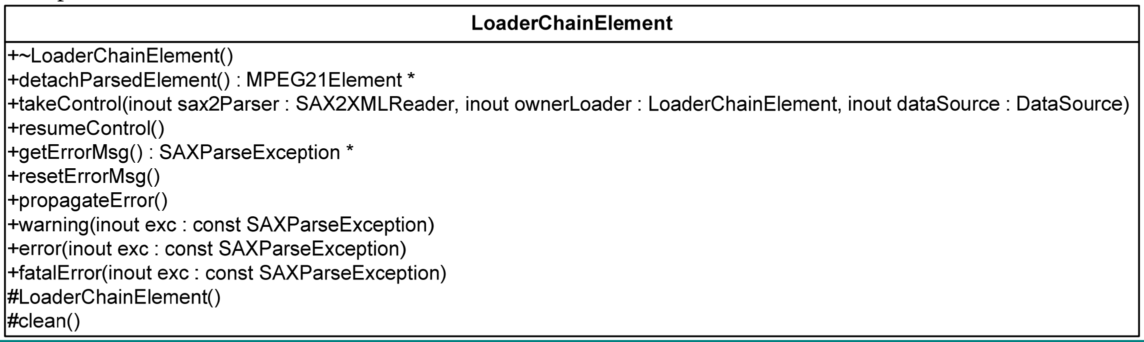

DIDLDocumentLoader
MPEG21ElementFactory
1: load(fileName)
2: SAX2XMLReader()
SAX2XMLReader
3: parse(inputSource)
4: startElement(uri, localname, qname, attrs)
5: DIDLDocument()
DIDLDocument
4:return
6: startElement(uri, localname, qname, attrs)
7: MPEG21Loader()
MPEG21Loader
8: takeControl(sax2Parser, ownerLoader, dataSource)
8: return
9: startElement(uri, localname, qname, attrs)
10: createMPEG21Element:=createMPEG21Element(ns, elementName, attrs, ownerDocument)
10: return
9: return
6: return
11: startElement(uri, localname, qname, attrs)
12: XILoader()
XILoader
13: takeControl(sax2Parser, ownerLoader, dataSource)
13: return
14: startElement(uri, localname, qname, attrs)
15: createMPEG21Element(ns, elementName, attrs, ownerDocument)
15: return
14: return
11: return
16: endElement(uri, localname, qname)
17: resumeControl()
18: detachParsedElement()
18: return
17: return
16: return
20: resumeControl()
19: endElement(uri, localname, qname)
21: detachParsedElement()
21: return
22: setRootElement(rootElement)
22: return
20: return
19: return
23: endElement(uri, localname, qname)
23: return
2: return
1: return new document

Loading operations follow a recursive pattern. All loading classes are supposed to be proficient in a specific namespace or element managing. Transitions through different loaders and factories are established by a root coordinating element (*MPEG21ElementFactory* for the factories and *MPEG21Loader* for the loaders) that implements capabilities to select the proper class on the base of the inputs that comes from XML parser. In the case of factories classes the architecture is composed by static elements, any loader will refer to static *MPEG21ElementFactory* class to create a new element, this implementation issue is possible because factories are stateless elements. Loader instead are not static, a proper loader instance is created when required by loading flows. When a loader end his task is deleted freeing memory. Loader indeed needs a inner state to work.

4.2.2 Class Methods Overwiew

LoaderChainElement:

This class represents loaders common interface. Methods defined here are virtual and could be overridden by local implementations of the loader classes.



LoaderChainElement

```
+~LoaderChainElement()
+detachParsedElement() : MPEG21Element *
+takeControl(inout sax2Parser : SAX2XMLReader, inout ownerLoader : LoaderChainElement, inout dataSource : DataSource)
+resumeControl()
+getErrorMsg() : SAXParseException *
+resetErrorMsg()
+propagateError()
+warning(inout exc : const SAXParseException)
+error(inout exc : const SAXParseException)
+fatalError(inout exc : const SAXParseException)
#LoaderChainElement()
#clean()
```

LoaderChainElement – class methods
LoaderChaiElement – ~LoaderChainElement
Class constructor and destructor
detachParsedElement
Returns root element of the representation tree parsed by the loader
takeControl
take the control of the parsing flow from another loader instance.
resumeControl
Resume the control of parsing flow from another loader instance. The loader adds as a child element of his representation tree the root element contained by the other loader
getErrorMsg
Retrivie information about an occurred error from a bottom loader instance
propagateError
Send a message to a upper loader of an error occurrence.
warning – error
Rise a warning or an error during the parsing operations
Clean
Reset the loader

DIDLDocumentLoader

This class acts as access point and root element for all loading activities. It is responsible to create a new *DIDLDocument* representation in memory and to instance a new *MPEG21Loader* that will be the real engine of loading. At the end of his work *DIDLDocumentLoader* will collect parsed DI representation and will be able to add the representation to the newly created *DIDLDocument*

DIDLDocumentLoader
-mDocument : DIDLDocument * -mState : DIDLDocumentLoaderState *

+detachParsedElement() : MPEG21Element *
+takeControl(inout sax2Parser : SAX2XMLReader, inout ownerLoader : LoaderChainElement, inout dataSource : DataSource)
+resumeControl()
+propagateError()
#clean()
+DIDLDocumentLoader()
+~DIDLDocumentLoader()
+load(inout source : DataSource) : DIDLDocument *
+load(inout fileName : const string) : DIDLDocument *
+characters(in chars : const XMLCh*, in length : const unsigned int)
+endElement(in uri : const XMLCh*, in localname : const XMLCh*, in qname : const XMLCh*)
+startElement(in uri : const XMLCh*, in localname : const XMLCh*, in qname : const XMLCh*, inout attrs : const Attributes)
-setState(in newState : DIDLDocumentLoaderState*)

DIDLDocumentLoader derives from the abstract class *LoaderChainElement*, therefore the former implements all the functions defined in the latter. These functions are not replicated in the table below.

DIDLDocumentLoader – class methods
DIDLDocumentLoader – ~DIDLDocumentLoader
Class constructor and destructor
detachParsedElement
Returns root element of the representation tree parsed by the loader
load
Load a representation from a data source or a file entry
characters
Skip useless characters in document
startElement
Called by the XML parser when a start element event occurs. Changes from state to state. It could lead to state change, MPEG21Loader instantiation or exception rising. Depending from the state is even the creation of a new document or a new element to be added to representation tree
endElement
Called by the XML parser when an end element event occurs. Add current parsed element to the representation tree

MPEG21Loader

The real engine of loading, through an instance of this class the loading of target document is accomplished. Loading operations are worked out in a recursive way, going down in pre-order visit . When the loader finds an unmanageable element it instances a proper loader chosen between available loader factories and gives it the loading control. Control will be returned ,along with the loading output, when unmanageable namespace or elements ends.

MPEG21Loader

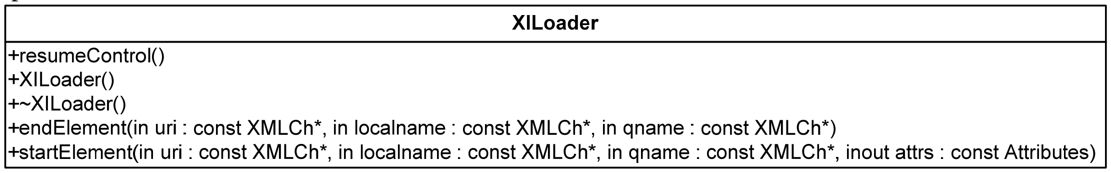
+resumeControl()
#clean()
+MPEG21Loader()
+~MPEG21Loader()
+addLoaderFactory(inout ns : const string, inout elemName : const string, in loaderFactory : LoaderFactoryType)
+addDefaultNsLoaderFactory(inout ns : const string, in loaderFactory : LoaderFactoryType)
+getLoaderFactory(inout ns : const string, inout elemName : const string = "") : LoaderFactoryType
+characters(in chars : const XMLCh*, in length : const unsigned int)
+ignorableWhitespace(in chars : const XMLCh*, in length : const unsigned int)
+endElement(in uri : const XMLCh*, in localname : const XMLCh*, in qname : const XMLCh*)
+startElement(in uri : const XMLCh*, in localname : const XMLCh*, in qname : const XMLCh*, inout attrs : const Attributes)
+load(inout dataSource : DataSource) : MPEG21Element *

MPEG21Loader derives from the abstract class *LoaderChainElement*, therefore the former implements all the functions defined in the latter. These functions are not replicated in the table below.

MPEG21Loader – class methods
MPEG21Loader – ~ MPEG21Loader
Class constructor and destructor

load
Load a representation from a data source
addLoaderFactory
Adds to the loader a Factory for a specified element in a specified namespace
addDefaultNsLoaderFactory
Adds a Factory to manage a specified namespace's elements creation
getLoaderFactory
Get the factory associated with input namespace
characters
Skip useless characters in document
ignorableWhitespace
Skip useless whitespace in document
startElement
Called by an XML parser when a start element event occurs. Changes from state to state. It could lead to state change, MPEG21Loader instantiation or exception rising. Depending from the state is even the creation of a new document or a new element to be added to representation tree
endElement
Called by an XML parser when an end element event occurs. Add current parsed element to the representation tree

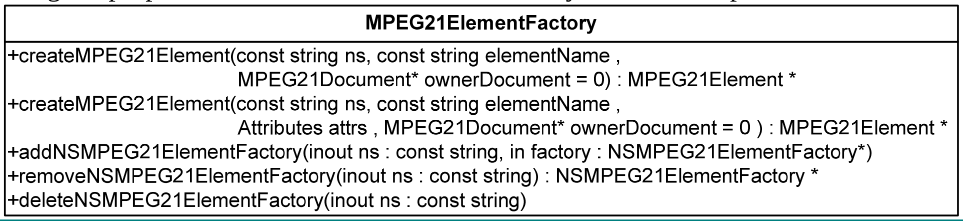
XILoader
This is a class for XI hierarchy elements loading . This class provides the needed functionalities to manage the difference between XInclude and other elements. However the implementation is transparent and the class provides the same methods of a common loader



XILoader

+resumeControl()
+XILoader()
+~XILoader()
+endElement(in uri : const XMLCh*, in localname : const XMLCh*, in qname : const XMLCh*)
+startElement(in uri : const XMLCh*, in localname : const XMLCh*, in qname : const XMLCh*, inout attrs : const Attributes)

MPEG21ElementFactory
This class coordinate automatic creation of elements on the base of element namespace and element name inputs. Selecting the proper factories and maintains a ns/factory reference map are first tasks of this class.



MPEG21ElementFactory
+createMPEG21Element(const string ns, const string elementName , MPEG21Document* ownerDocument = 0) : MPEG21Element * +createMPEG21Element(const string ns, const string elementName , Attributes attrs , MPEG21Document* ownerDocument = 0) : MPEG21Element * +addNSMPEG21ElementFactory(inout ns : const string, in factory : NSMPEG21ElementFactory*) +removeNSMPEG21ElementFactory(inout ns : const string) : NSMPEG21ElementFactory * +deleteNSMPEG21ElementFactory(inout ns : const string)

MPEG21ElementFactory – class methods
createMPEG21Element
Call and delegate the proper namespace factory to instance a new element of the input element type . Alternative method also pass attrs content to fills element's proper attributes fields.
addNSMPEG12ElementFactory – removeNSMPEG21ElementFactory – deleteNSMPEG21ElementFactory
Adds removes or deletes named factory from the collection available in MPEG21ElementFactory

DIDLElementFactory
This class and the classes below are examples of specific NS factory classes. Main task of these classes is selection of the right element to instance on the base of element name. A secondary task of this classes is filling the new element with provided attributes value.

DIDLElementFactory
+createMPEG21Element(const string elementName , MPEG21Document* ownerDocument = 0) : MPEG21Element *
+createMPEG21Element(const string elementName , const Attributes attrs, MPEG21Document* ownerDocument = 0) : MPEG21Element *
+getNamespace() : const string &

DIDLElementFactory – class methods
createMPEG21Element
Create a new IPMPElement of the input element type . Alternative method also fills element's proper attributes fields in the created element
getNamespace
Returns the factory target namespace

IPMPElementFactory
+createMPEG21Element(const string elementName , MPEG21Document* ownerDocument = 0) : MPEG21Element *
+createMPEG21Element(const string elementName , const Attributes attrs, MPEG21Document* ownerDocument = 0) : MPEG21Element *
+getNamespace() : const string &

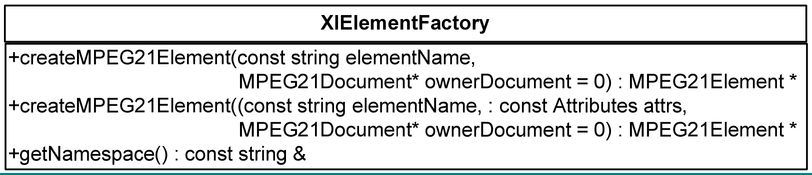
IPMPElementFactory – class methods
createMPEG21Element
Create a new IPMPElement of the input element type . Alternative method also fills element's proper attributes fields in the created element
getNamespace
Returns the factory target namespace

DIIElementFactory
+createMPEG21Element(const string elementName , MPEG21Document* ownerDocument = 0) : MPEG21Element *
+createMPEG21Element(const string elementName , const Attributes attrs, MPEG21Document* ownerDocument = 0) : MPEG21Element *
+getNamespace() : const string &

DIIElementFactory – class methods
createMPEG21Element
Create a new IPMPElement of the input element type . Alternative method also fills element's proper attributes fields in the created element
getNamespace
Returns the factory target namespace

IPMPInfoElementFactory
+createMPEG21Element(const string elementName , MPEG21Document* ownerDocument = 0) : MPEG21Element *
+createMPEG21Element(const string elementName , const Attributes attrs, MPEG21Document* ownerDocument = 0) : MPEG21Element *
+getNamespace() : const string &

IPMPInfoElementFactory – class methods
createMPEG21Element
Create a new IPMPElement of the input element type . Alternative method also fills element's proper attributes fields in the created element
getNamespace
Returns the factory target namespace



```
XIElementFactory
+createMPEG21Element(const string elementName,
                     MPEG21Document* ownerDocument = 0) : MPEG21Element *
+createMPEG21Element((const string elementName, : const Attributes attrs,
                     MPEG21Document* ownerDocument = 0) : MPEG21Element *
+getNamespace() : const string &
```

XIElementFactory – class methods
createMPEG21Element
Create a new IPMPElement of the input element type . Alternative method also fills element's proper attributes fields in the created element
getNamespace
Returns the factory target namespace

4.3 Examples of usage

In the following code example use of DIDLDocumentLoader interface is showed. To point out the ease of use of this interface. Only target input xml file is needed to completely parse an XML MPEG-21 Document

```
DIDLDocumentLoader loader;
DIDLDocumentWriter documentwri;
DIDLDocument*doc=loader.load("MPEG21XMLREPRESENTATION.xml");
std::ofstream filestr("MPEG21XMLOUTPUT.xml");
documentwri.writeDocument(*doc,&filestr);
filestr.close();
```

4.4 Integration and compilation issues

The following table summarizes the needed library in order to use MPEG-21 Loader.

Name	OS/Platform		Library file	Description
	Windows/PC	Linux/PC		
Xerces C++	X	X	xerces-c_2.lib	Provide parsing functionalities for the module
Common	X	X	common.lib	

`

4.5 Configuration Parameters

Config parameter	Possible values
AXOM_CONF_MODULE-SCHEMA_LOCATION_PARAM	Any valid URL which points to schema file location for MPEG-21 Schema

4.6 Errors reported and that may occur

Error code	Description and rationales
Class	**Loaders**
0	This loader has already an Owner Loader
1	This loader has already a SAX2Parser instance
2	resumeControl not supported
3	No available Active loader. Propagate error call improper
4	Improper method call to this loader
5	Specified element expected
6	Element unexpected
Class	**Factories**
7	Invalid input elment name is given to element factory
8	Namespace Factory already added in MPEG21ElementFactory

Class	Documents
9	`Document has not a root element`
10	`Invalid input element type`

5 MPEG-21 Saver (DSI)

Module/Tool Profile		
MPEG-21 Saver		
Responsible Name	Andrea Vallotti, Davide Rogai	
Responsible Partner	DSI	
Status (proposed/approved)		
Implemented/not implemented	implemented	
Status of the implementation		
Executable or Library/module (Support)	static library	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows, Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/.....................	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any		
Test cases (present/absent)		
Test cases location	http://////////////////	
Usage of the AXMEDIS configuration manager (yes/no)		
Usage of the AXMEDIS Error Manager (yes/no)		
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section

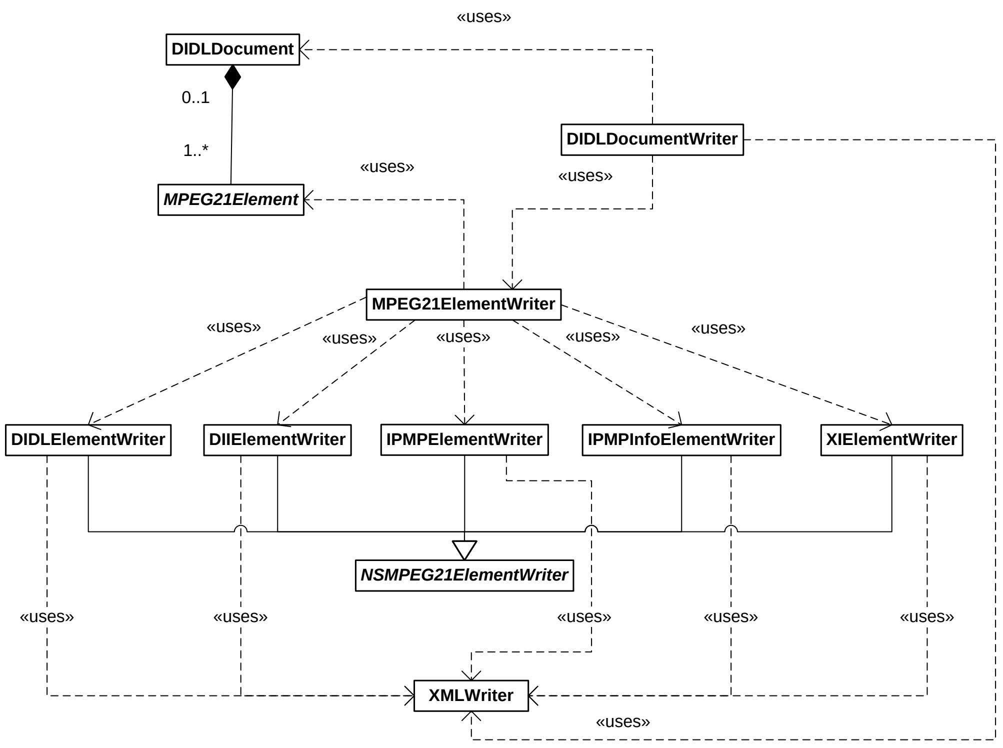
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

5.1 General Description of the Module

The MPEG-21 Saver allows to obtain the XML representation of a software representation of a DI. In particular, the Saver browses the object model of DI and writes the XML representation of the latter to a stream of byte. In order to exploit this task, the Saver makes use of an XML writer which is in charge of managing the issues related to the XML syntax.

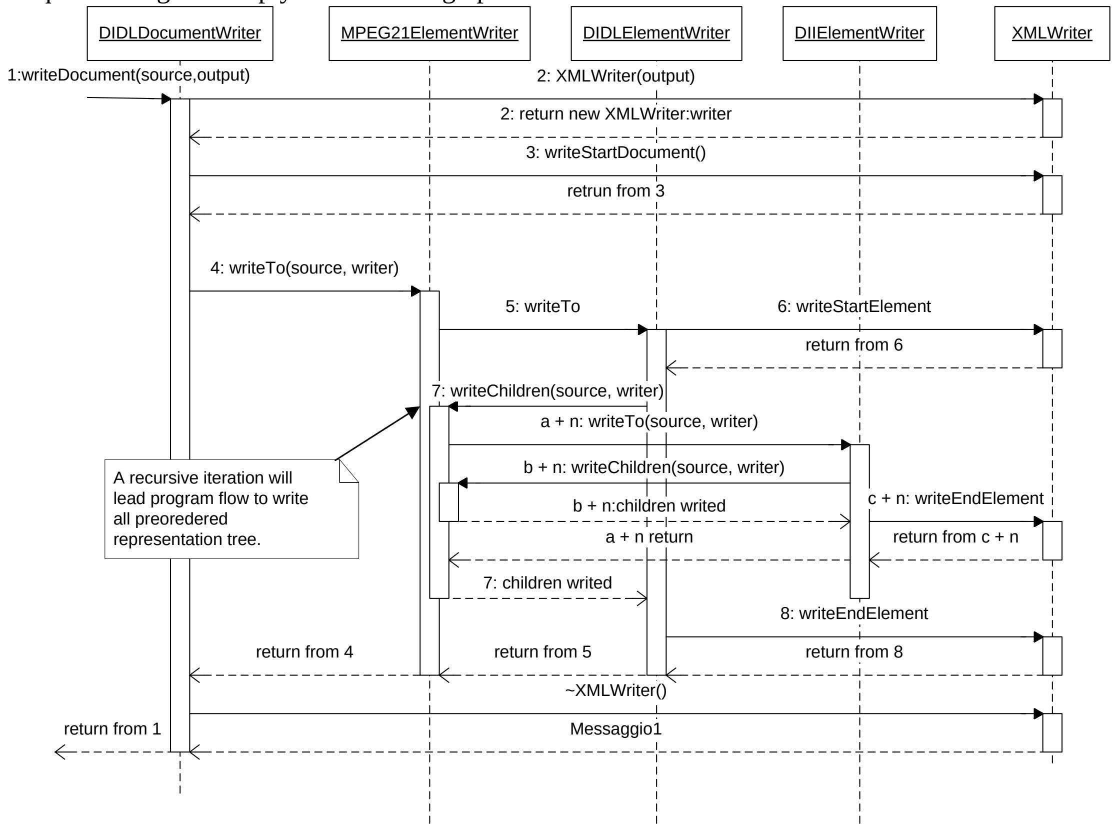
5.2 Module Design in terms of Classes

As shown in next picture MPEG-21 writer top element is *DIDLDocumentWriter*. Through this element an application or user could use module to write a *DIDLDocument* as DI XML representation. All elements in the module uses an instance of *XMLWriter* class. This class is responsible for XML syntax of the produced document. All issues that derives from data writing, data format, and data syntax from an XML point of view (including namespaces, prefixes rules and XML Schema compliance) are resolved by means of this class. Last elements included in this module *MPEG21ElementWriter* and related namespace writers (i.e. NSWriters) navigates representation tree providing input data for XMLWriter, in the right order to allow a correct output XML document.



5.2.1 MPEG-21Saver Capabilities Overview

Next sequence diagram deeply shows writing operation. *DIDLDocumenWriter* instances a new *XMLWriter*.



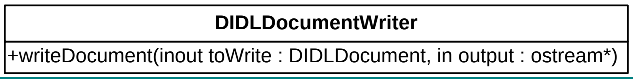
Once the new *XMLWriter* is available *DIDLDocumentWriter* starts writing operations, asking *XMLWriter* to write XML document header. Writing operations flow is given to *MPEG21ElementWriter*, this class will decide the writer which will take control of writing for next element of representation tree. The chosen writer

(*DIDLElementWriter* in the picture) will send a startElement command to *XMLWriter* in order to write element start tag and then, depending on the current element, ask *MPEG21ElementWriter* to check and write children of current element. Here begins a recursive iteration that will lead to a complete pre-ordered visit of representation tree, performing writing of any single element. In order to point out the role of *XMLWriter*, this element maintains the state information useful for writing operations (open element tags, current namespace domains and namespace/prefix parallelism).

5.2.2 Class and Methods Overview

DIDLDocumentWriter

Interface for writing operations.

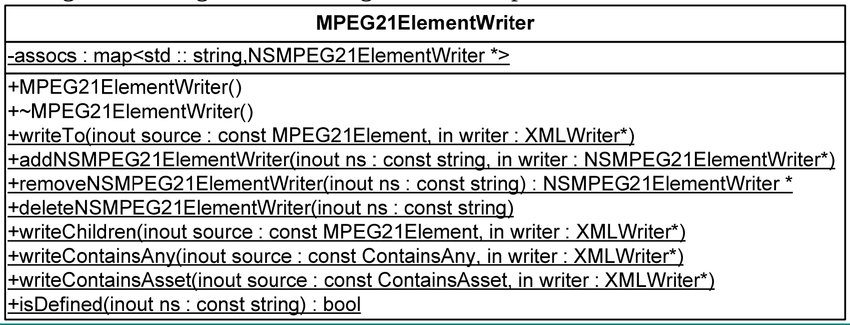


```
DIDLDocumentWriter
+writeDocument(inout toWrite : DIDLDocument, in output : ostream*)
```

DIDLDocumentWriter – class methods
writeDocument
This method starts starts writing operations . In this method a new XMLWriter is created and is used to write input document in output stream

MPEG21ElementWriter

This static class manage all writing flow choosing wich namespace writer is able to write current element

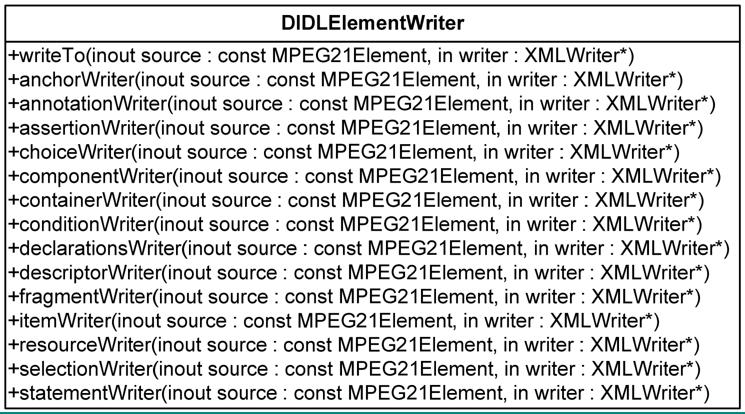


```
MPEG21ElementWriter
-assocs : map<std :: string,NSMPEG21ElementWriter *>
+MPEG21ElementWriter()
+~MPEG21ElementWriter()
+writeTo(inout source : const MPEG21Element, in writer : XMLWriter*)
+addNSMPEG21ElementWriter(inout ns : const string, in writer : NSMPEG21ElementWriter*)
+removeNSMPEG21ElementWriter(inout ns : const string) : NSMPEG21ElementWriter *
+deleteNSMPEG21ElementWriter(inout ns : const string)
+writeChildren(inout source : const MPEG21Element, in writer : XMLWriter*)
+writeContainsAny(inout source : const ContainsAny, in writer : XMLWriter*)
+writeContainsAsset(inout source : const ContainsAsset, in writer : XMLWriter*)
+isDefined(inout ns : const string) : bool
```

MPEG21ElementWriter – class methods
MPEG21ElementWriter - ~MPEG21ElementWriter
Contstructor and destructor
writeTo
Choose wich NSwriter will save current element. Namespace is taken from input element.
addNSMPEG21ElementWriter – removeNSMPEG21ElementWriter – deleteNSMPEG21ElementWriter
Add remove or delete from assoc a pair ns/writer
writeChildren
Write children of input element. Children are selected in preorder visit and for each child method writeTo is called
writeContainsAny – writeContainsAsset
Special implementation for two element types of representation tree.
isDefined
Check if a pair ns/writer is defined in MPEG21ElementWriter

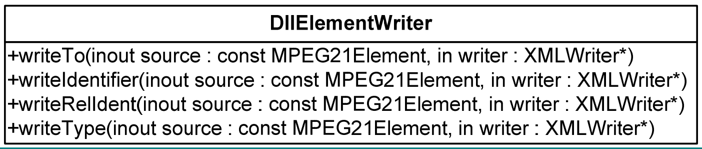
Namespace Writers

These classes implements methods to manage writng of elements related to a specific namespace.



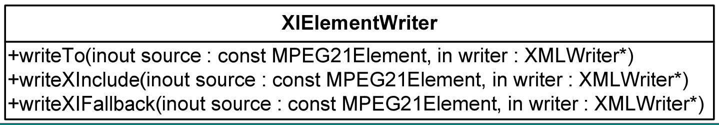
```
DIDLElementWriter
+writeTo(inout source : const MPEG21Element, in writer : XMLWriter*)
+anchorWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+annotationWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+assertionWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+choiceWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+componentWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+containerWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+conditionWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+declarationsWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+descriptorWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+fragmentWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+itemWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+resourceWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+selectionWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
+statementWriter(inout source : const MPEG21Element, in writer : XMLWriter*)
```

DIDLElementWriter – class methods
writeTo
Choose wich element writer will save current element. Element name is taken from input element.
anchorWriter – annotationWriter – assertionWriter – choiceWriter – componentWriter – containerWriter – conditionWriter – declarationsWriter – descriptorWriter – fragmentWriter – itemWriter – resourceWriter – selectionWriter – statementWriter
These methods manage writing of specific DIDL elements



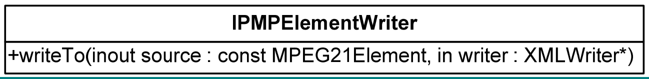
```
DIIElementWriter
+writeTo(inout source : const MPEG21Element, in writer : XMLWriter*)
+writeIdentifier(inout source : const MPEG21Element, in writer : XMLWriter*)
+writeRelIdent(inout source : const MPEG21Element, in writer : XMLWriter*)
+writeType(inout source : const MPEG21Element, in writer : XMLWriter*)
```

DIIElementWriter – class methods
writeTo
Choose wich element writer will save current element. Element name is taken from input element.
writeIdentifier – writeRelIdentifier – writeType
These methods manage writing of specific DII elements



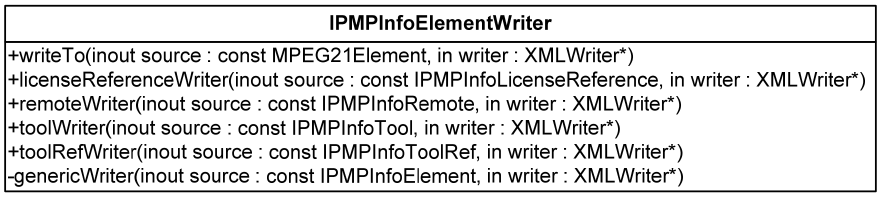
```
XIElementWriter
+writeTo(inout source : const MPEG21Element, in writer : XMLWriter*)
+writeXInclude(inout source : const MPEG21Element, in writer : XMLWriter*)
+writeXIFallback(inout source : const MPEG21Element, in writer : XMLWriter*)
```

XIElementWriter – class methods
writeTo
Choose wich element writer will save current element. Element name is taken from input element.
writeXInclude – writeXIFallback
These methods manage writing of specific DII elements



```
IPMPElementWriter
+writeTo(inout source : const MPEG21Element, in writer : XMLWriter*)
```

IPMPElementWriter – class methods
writeTo
Writes IPMP element to output stream



```
IPMPInfoElementWriter
+writeTo(inout source : const MPEG21Element, in writer : XMLWriter*)
+licenseReferenceWriter(inout source : const IPMPInfoLicenseReference, in writer : XMLWriter*)
+remoteWriter(inout source : const IPMPInfoRemote, in writer : XMLWriter*)
+toolWriter(inout source : const IPMPInfoTool, in writer : XMLWriter*)
+toolRefWriter(inout source : const IPMPInfoToolRef, in writer : XMLWriter*)
-genericWriter(inout source : const IPMPInfoElement, in writer : XMLWriter*)
```

IPMPInfoElementWriter – class methods
writeTo
Choose wich element writer will save current element. Element name is taken from input element.
licenseReferenceWriter – toolWriter – remotWriter – toolRefWriter -
These methods manage writing of specific DII elements
genericWriter
Writes a generic IPMPInfoElement

5.3 Examples of usage

The following example code shows how is used DIDLDocumentWriter interface to write a DIDLDocument formerly loaded by DIDLDocumentLoader. To point out the use of a std::ofstream as target output stream. All output streams are valid target of writeDocument method.

```
DIDLDocumentLoader loader;
DIDLDocumentWriter documentwri;
DIDLDocument*doc=loader.load("MPEG21XMLREPRESENTATION.xml");
std::ofstream filestr("MPEG21XMLOUTPUT.xml");
documentwri.writeDocument(*doc,&filestr);
filestr.close();
```

5.4 Integration and compilation issues

The following table summarizes the needed library in order to use MPEG-21 Loader.

Name	**OS/Platform**		**Library file**	**Description**
	Windows/PC	**Linux/PC**		
Common	X	X	common.lib	Provide access to xmlwriter and support for off-schema elements

5.5 Errors reported and that may occur

Error code	**Description and rationales**
0	Namespace Factory already added in MPEG21ElementFactory
1	`Invalid input element type`

6 AXMEDIS Object Model (DSI)

Module/Tool Profile		
AXMEDIS Object Model		
Responsible Name	Davide Rogai, Andrea Vallotti	
Responsible Partner	DSI	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)	Static librsy	
Single Thread or 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/axom/axmodel/ https://cvs.axmedis.org/repos/Framework/include/axom/axmodel/	
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

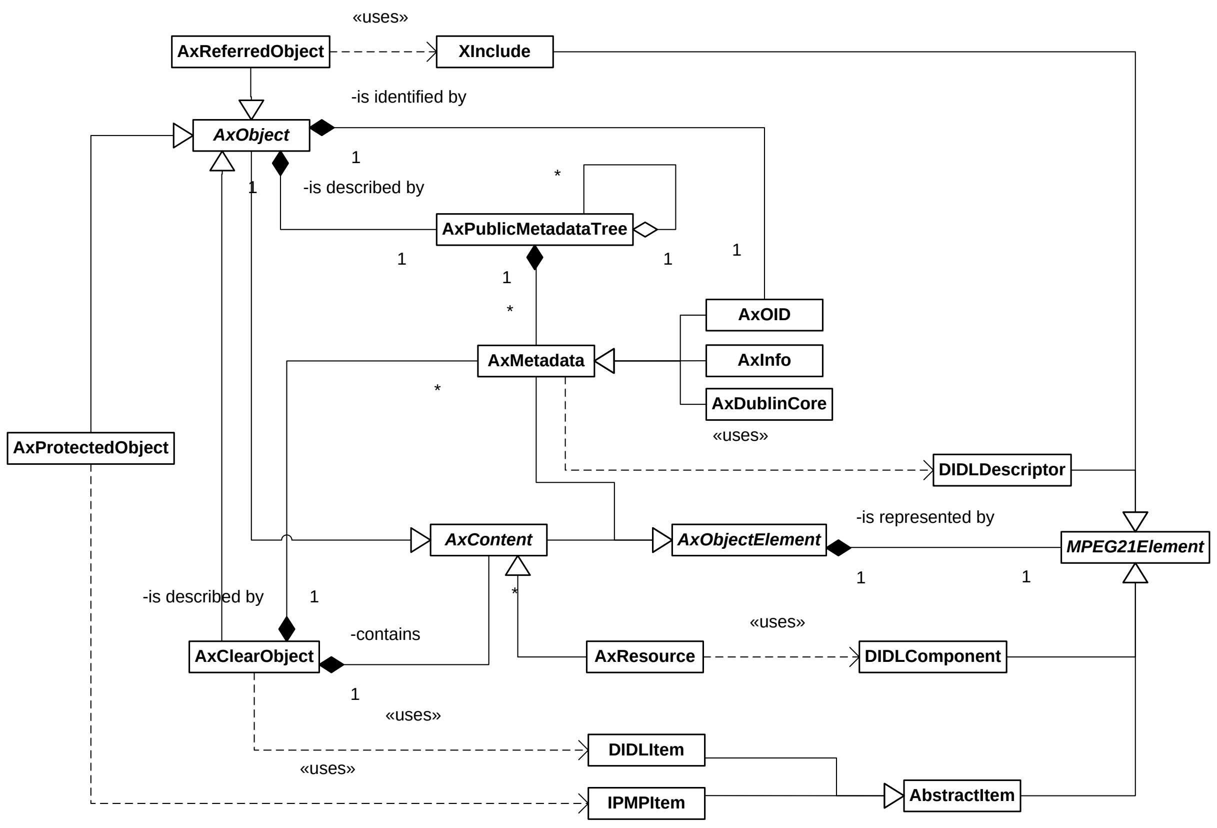
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

6.1 General Description of the Module

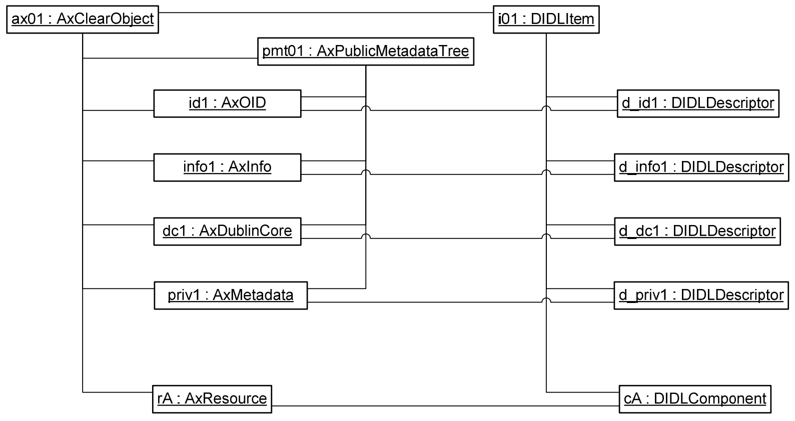
The AXMEDIS Object Model provides the means to represent an AXMEDIS Object as a software object model on the basis of the MPEG-21 Object Model. In fact, an AXMEDIS Object is a particular kind of DI. That is, an AXMEDIS Object is a DI with a given structure and providing mandatory information such as the AXOID and AxInfo as explained in the following sections. Therefore, the object model of an AXMEDIS object is a tree of class instances each of which refers to the corresponding class instance in the MPEG-21 Object Model.
Moreover, the AXMEDIS Object Model provides the means to create an object model on the basis of a MPEG-21 object model and to synchronize the two models.

6.2 Module Design in terms of Classes

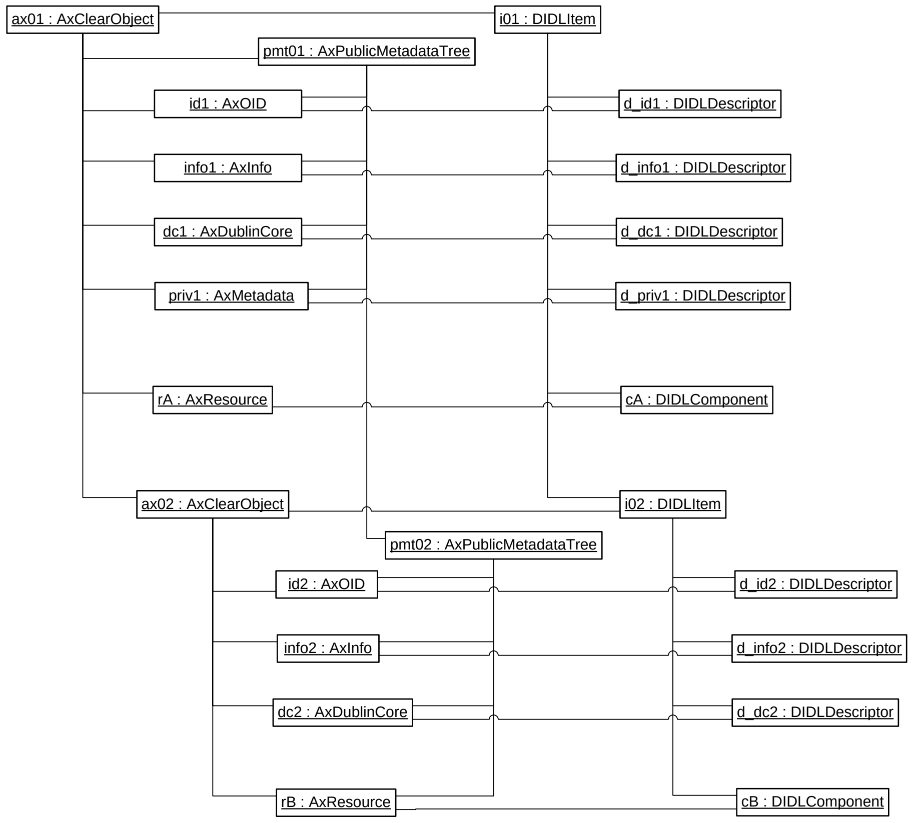
In this section the classes modelling AXMEDIS objects are reported.
Abstract class *AxObjectElement* represents any element that can be stored in an AXMEDIS Object. It refers to an *MPEG21Element* that represents it in the corresponding MPEG21 Digital Item.
Class *AxMetadata* represents any XML metadata associated with content, it is further specialised in *AxInfo*, *xDublinCore* and *AxOID*. The mPublic attribute indicates if the metadata has to visible even if the object is protected.
Abstract class *AxContent* represents content to be stored in AXMEDIS objects, it can be *AxObject* or *AxResource.* An *AxResource* represents any digital resource identified with a mime type, it can be an image, a document, an audio. An *AxObject* can contain any number of metadata and of content.



The following is the object diagram of a basic object:

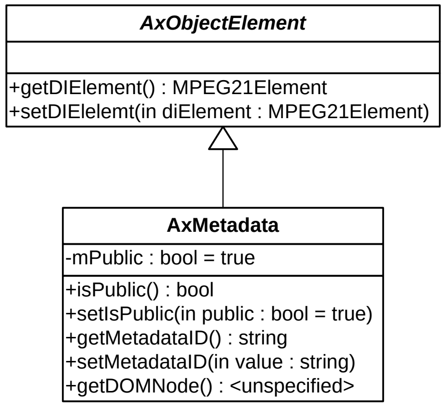


While the following is the object diagram of a composite AXMEDIS object:



6.2.1 AxMetadata

Class *AxMetadata* is a class to store any XML metadata.
The *MPEG21Element* it refers to should be a *DIDLDescriptor*, containing a *DIDLStatement* with the XML content. XML content can be accessed trought a *DOMNode* object.


AxObjectElement
+getDIElement() : MPEG21Element
+setDIElelemt(in diElement : MPEG21Element)
AxMetadata
-mPublic : bool = true
+isPublic() : bool
+setIsPublic(in public : bool = true)
+getMetadataID() : string
+setMetadataID(in value : string)
+getDOMNode() : <unspecified>

6.2.2 AxInfo

Class *AxInfo*, derived from *AxMetadata*, provides access to the information related to the AXMEDIS object.
Methods available for this class are:

ObjectCreator Management

+getObjectCreatorAXCID() : string +setObjectCreatorAXCID(in value : string)
allow to get and set the AXCID value for the ObjectCreator
+getObjectCreatorName() : string +setObjectCreatorName(in value : string)

allow to get and set the Name value for the ObjectCreator
+getObjectCreatorURL() : string +setObjectCreatorURL(in value : string)
allow to get and set the URL value for the ObjectCreator
+getObjectCreatorCompany() : string +setObjectCreatorCompany(in value : string)
allow to get and set the Company value for the ObjectCreator
+getObjectCreatorCompanyURL() : string +setObjectCreatorCompanyURL(in value : string)
allow to get and set the CompanyURL value for the ObjectCreator
+getObjectCreatorNationality() : string +setObjectCreatorNationality(in value : string)
allow to get and set the Nationality value for the ObjectCreator

ObjectContributor Management

+addObjectContributor(in position : int = -1) : int
adds a new ObjectContributor in the position given (starting from 0), position -1 means to add at the end. the return value indicates the position in which it is added.
+removeObjectContributor (in position : int)
removes an ObjectContributorfrom the position specified
+getObjectContributorCount() : int
returns the number of ObjectContributor present
+findObjectContributorByAXCID(in axcid : string) : int
returns the position of an ObjectContributor with a specific AXCID. It returns -1 if not found.
+findObjectContributorByName(in name : string) : int
returns the position of an ObjectContributor with a specific Name. It returns -1 if not found.
+getObjectContributorAXCID(in refNum : int = 0) : string +setObjectContributorAXCID(in refNum : int, in value : string)
allow to get and set the AXCID value for an ObjectContributor identified by position
+getObjectContributorName(in refNum : int = 0) : string +setObjectContributorName(in refNum : int, in value : string)
allow to get and set the Name value for an ObjectContributor identified by position
+getObjectContributorURL(in refNum : int = 0) : string +setObjectContributorURL(in refNum : int, in value : string)
allow to get and set the URL value for an ObjectContributor identified by position
+getObjectContributorCompany(in refNum : int = 0) : string +setObjectContributorCompany(in refNum : int, in value : string)
allow to get and set the Company value for an ObjectContributor identified by position
+getObjectContributorCompanyURL(in refNum : int = 0) : string +setObjectContributorCompanyURL(in refNum : int, in value : string)
allow to get and set the CompanyURL value for an ObjectContributor identified by position
+getObjectContributorNationality(in refNum : int = 0) : string +setObjectContributorNationality(in refNum : int, in value : string)
allow to get and set the Nationality value for an ObjectContributor identified by position

Owner Management

+getOwnerID() : string +setOwnerID(in value: string)
allow to get and set the code identifying the owner
+getOwnerIDCoding() : string +setOwnerIDCoding(in value: string)
allow to get and set the coding used to identify the owner
+getOwnerName() : string +setOwnerName(in value: string)

allow to get and set the name of the owner
+getOwnerURL() : string +setOwnerURL(in value: string)
allow to get and set the URL of the owner
+getOwnerCompany() : string +setOwnerCompany(in value: string)
allow to get and set the company of the owner
+getOwnerCompanyURL() : string +setOwnerCompanyURL(in value: string)
allow to get and set the company URL of the owner
+getOwnerNationality() : string +setOwnerNationality(in value: string)
allow to get and set the nationality of the owner
+addOwnerDescription(in position:int = -1) : int
adds a new description of the owner at the position specified or at the end if position is -1. The return value indicates the position where it is added.
+removeOwnerDescription(in position:int)
removes the description specified
+getOwnerDescription(in position:int = 0) : string +setOwnerDescription(in position:int, in value:string)
allow to get and set the value of the description
+getOwnerDescriptionLanguage(in position:int = 0) : string +setOwnerDescriptionLanguage(in position:int, in value:string)
allow to get and set the value of the description language

Distributor Management

+addDistributor()
adds a Distributor if not present.
+removeDistributor()
removes the Distributor
+getDistributorCount() : int
returns the number of Distributors present
+getDistributorAXDID() : string +setDistributorAXDID(in value : string)
allow to get and set the AXDID value for the Distributor
+getDistributorName() : string +setDistributorName(in value : string)
allow to get and set the Name value for the Distributor
+getDistributorURL() : string +setDistributorURL(in value : string)
allow to get and set the URL value for the Distributor
+getDistributorNationality() : string +setDistributorNationality(in value : string)
allow to get and set the Nationality value for the Distributor

Object Status

+getAccessMode() : string +setAccessMode(in value : string)
allow to get and set the Access the the object “READ_ONLY” or “READ_WRITE”. These strings have been used instead of C++ *enum* in order to simply the interface with JavaScript applications (e.g. AXMEDIS Content Processing)
+getCreationDate() : DateTime
get the local date and time of object creation
+getLastModificationDate() : DateTime
get the local date and time of object modification

+getVersion() : int
get the version of the object
+getRevision() : int
get the revision of the object
+getObjectStatus() : string +setObjectStatus(in value : string)
allow to get and set the current status of the object, the status values are factory dependent and set by the workflow therefore cannot be defined a priori.
+getObjectType() : string
allow to get object type ("BASIC" or "COMPOSITE")
+getObjectIsProtected() : bool +setObjectIsProtected(in value:bool)
allows to get and set if the object is protected or not
+getProtectionStamp() : string +setProtectionStamp(in value:string)
allows to get and set the protection stamp
+getObjectIsGoverned() : bool +setObjectIsGoverned(in value:bool)
allow to get and set if the object contains a licence or not. The license is not stored in the axinfo, the setter should be used to update the axinfo when the licence is added/removed from the object

PromoOf Management

+addPromoOfAXOID(in axoid:string, in position:int=-1)
adds a new AXOID in the PromoOf section, the position indicates where to put the AXOID, -1 means at the end
+removePromoOfAXOID(in position:int)
removes the AXOID in the position specified
+getPromoOfAXOIDCount() : int
get the count of AXOID in the PromoOf section
+getPromoOfAXOID(in position:int) : string +setPromoOfAXOID(in position:int, in value:string)
allow to get and set the AXOID in a specified position

Workflow Status

+getWorkflowWorkItemID() : string +setWorkflowWorkItemID (in value : string)
allow to get and set the WorkflowWorkItemID
+getWorkflowWorkspaceInstanceID() : string +setWorkflowWorkspaceInstanceID (in value : string)
allow to get and set the WorkflowWorkspaceInstanceID

Internal Potential Available Rights Management

+addInternalPotentialAvailableRights()
adds a new Internal PAR section
+removeInternalPotentialAvailableRights()
removes the Internal PAR section
+getInternalPotentialAvailableRightsCount()
gets how many Internal PAR sections are present (0 or 1)
+getInternalPotentialAvailableRightsStatus() : string +setInternalPotentialAvailableRightsStatus(in value:string)
allow to get and set the internal PAR status
+getInternalPotentialAvailableRightsLicense() : DOMNode
gets the DOM node of the license

Potential Available Rights Management

+addPotentialAvailableRights()
adds a new PAR section if not present

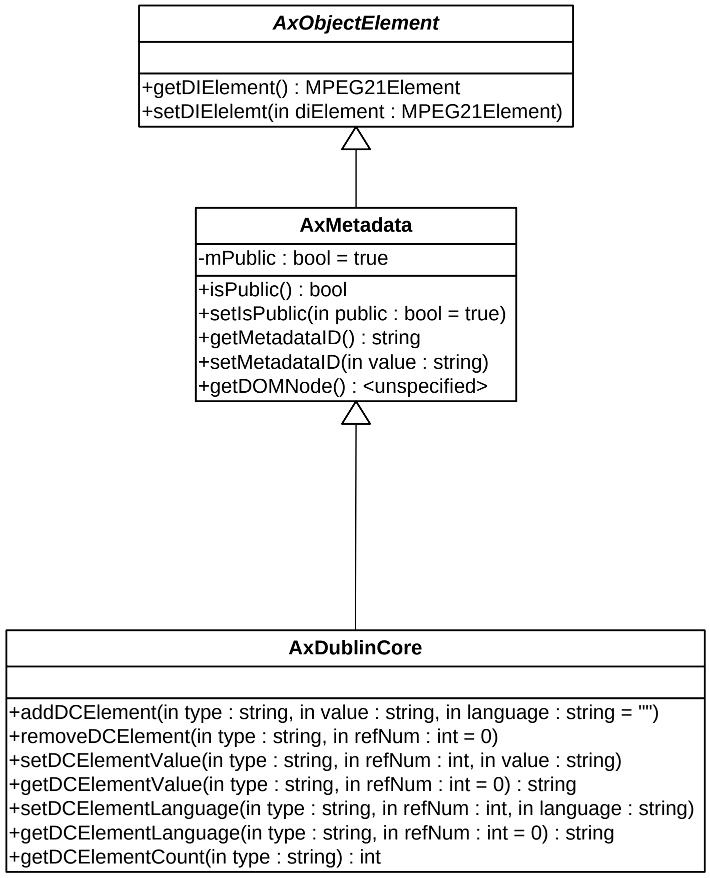
+removePotentialAvailableRights()
removes the PAR section
+getPotentialAvailableRightsCount()
gets how many PAR sections are present (0 or 1)
+getPotentialAvailableRightsLicensingURL() : string +setPotentialAvailableRightsLicensingURL (in value:string)
allow to get and set the licensing URL
+getPotentialAvailableRightsStatus() : string +setPotentialAvailableRightsStatus(in value:string)
allow to get and set the PAR status
+getPotentialAvailableRightsLicense() : DOMNode
gets the DOM node of the license

Object History Management

+getHistoryOfVersion(in version:int) : DOMNode
gets the history of a version as a DOM Node

6.2.3 AxDublinCore

Class AxDublinCore allows to manage a Dublin Core descriptor:


AxObjectElement
+getDIElement() : MPEG21Element
+setDIElelemt(in diElement : MPEG21Element)
AxMetadata
-mPublic : bool = true
+isPublic() : bool
+setIsPublic(in public : bool = true)
+getMetadataID() : string
+setMetadataID(in value : string)
+getDOMNode() : <unspecified>
AxDublinCore
+addDCElement(in type : string, in value : string, in language : string = "")
+removeDCElement(in type : string, in refNum : int = 0)
+setDCElementValue(in type : string, in refNum : int, in value : string)
+getDCElementValue(in type : string, in refNum : int = 0) : string
+setDCElementLanguage(in type : string, in refNum : int, in language : string)
+getDCElementLanguage(in type : string, in refNum : int = 0) : string
+getDCElementCount(in type : string) : int

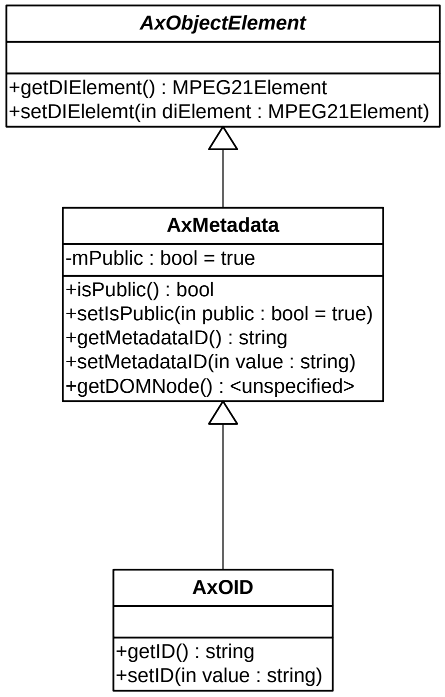
Example of use:

```
AxDublinCore aDC;

if(aDC.getDCElementCount(“creator”)==0)
        aDC.addDCElement(“creator”, “Mozart”);
```

```
else
        aDC.setDCElement(“creator”, 0, “Mozart”);

string creator=aDC.getDCElementValue(“creator”);
```

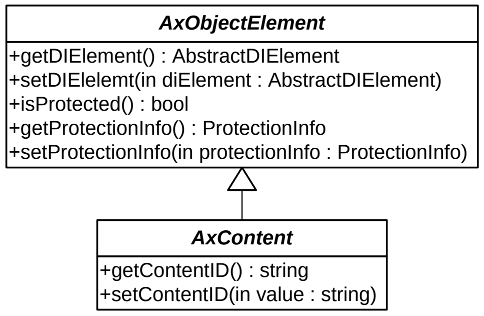
6.2.4 AxOID

Class AxOID embeds the AXOID identifier.


AxObjectElement
+getDIElement() : MPEG21Element
+setDIElelemt(in diElement : MPEG21Element)
AxMetadata
-mPublic : bool = true
+isPublic() : bool
+setIsPublic(in public : bool = true)
+getMetadataID() : string
+setMetadataID(in value : string)
+getDOMNode() : <unspecified>
AxOID
+getID() : string
+setID(in value : string)

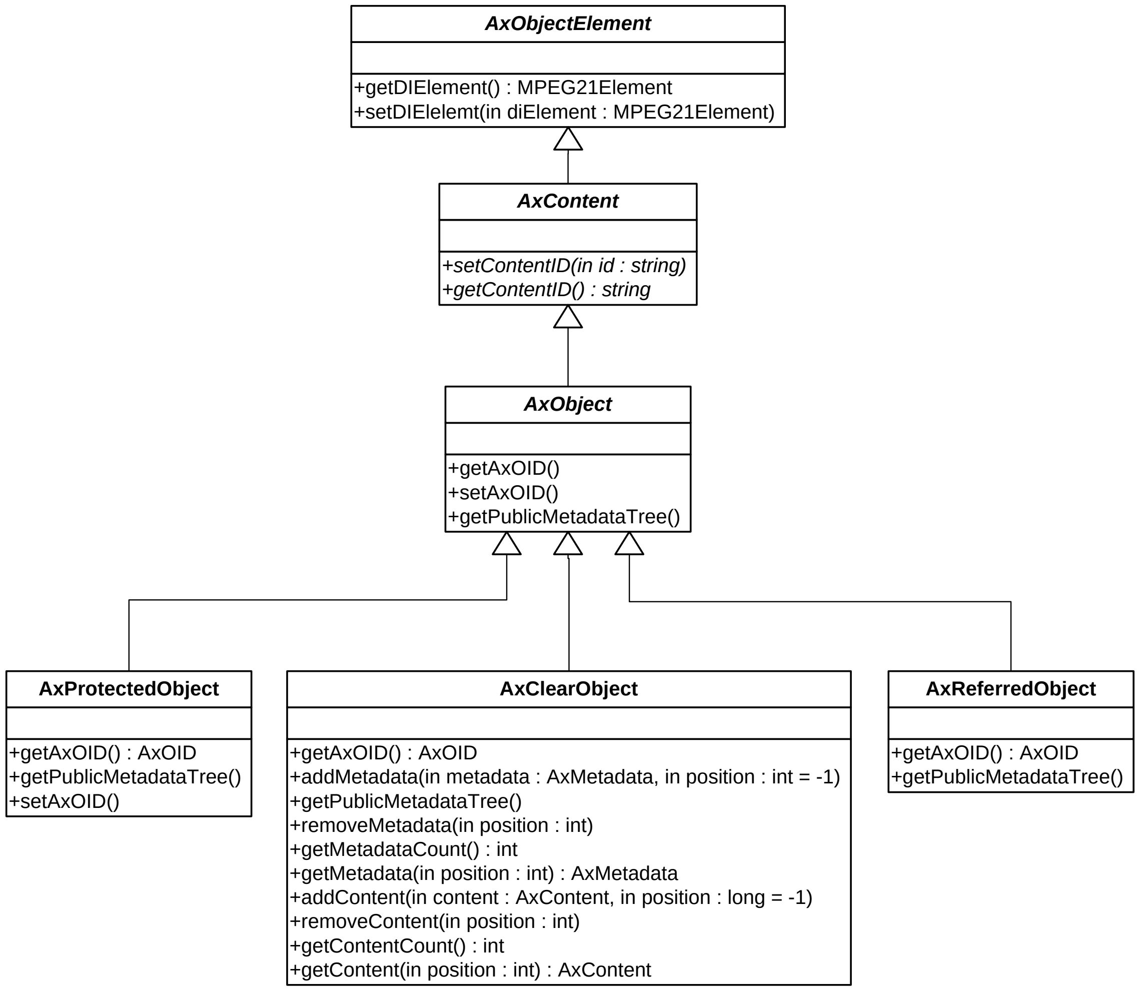
6.2.5 AxContent

Abstract class *AxContent* represents content to be stored in AXMEDIS objects, it is specialized in *AxObject* or *AxResource*.


AxObjectElement
+getDIElement() : AbstractDIElement
+setDIElelemt(in diElement : AbstractDIElement)
+isProtected() : bool
+getProtectionInfo() : ProtectionInfo
+setProtectionInfo(in protectionInfo : ProtectionInfo)
AxContent
+getContentID() : string
+setContentID(in value : string)

6.2.6 AxObject

Class *AxObject* represents an AXMEDIS Object, in its different forms. Generally, it models the basic relations of an AXMEDIS object like identification (refers to an AXOID) and classification with metadata that has to be always accessible (in any form) the so called “Public Metadata”.


AxObjectElement
+getDIElement() : MPEG21Element
+setDIElelemt(in diElement : MPEG21Element)
AxContent
+setContentID(in id : string)
+getContentID() : string
AxObject
+getAxOID()
+setAxOID()
+getPublicMetadataTree()
AxProtectedObject
+getAxOID() : AxOID
+getPublicMetadataTree()
+setAxOID()
AxClearObject
+getAxOID() : AxOID
+addMetadata(in metadata : AxMetadata, in position : int = -1)
+getPublicMetadataTree()
+removeMetadata(in position : int)
+getMetadataCount() : int
+getMetadata(in position : int) : AxMetadata
+addContent(in content : AxContent, in position : long = -1)
+removeContent(in position : int)
+getContentCount() : int
+getContent(in position : int) : AxContent
AxReferredObject
+getAxOID() : AxOID
+getPublicMetadataTree()

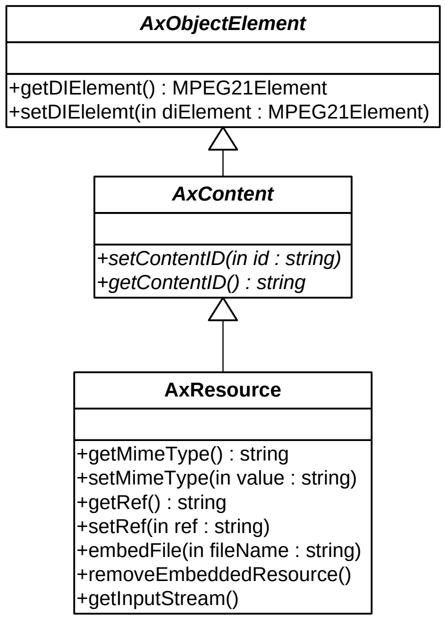
Class *AxClearObject* represent a "clear-text" aggregation of content and metadata. It can contain any number of metadata and any number of content. It exposes all the needed methods in order to manipulate the content structure it represents.

Class *AxProtectedObject* represent a "protected" (i.e. not accessible without permission) AXMEDIS Object. Since the classification have to be always possible methods to access Public metadata of the pretected object are available. The AXOM it is in charge of obtaining the corresponding AxClearObject when an authorized action has to be performed.

Class *AxReferredObject* models reference to external AXMEDIS Objects. In order to immediately perceive which kind of object is referred public metadata are retrievable.

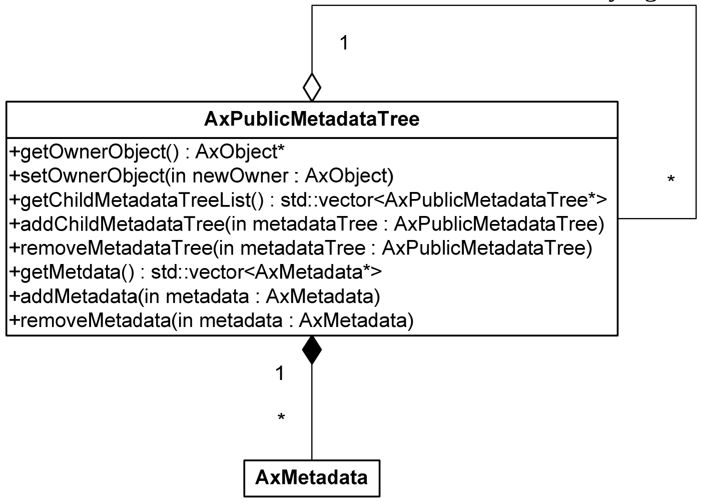
6.2.7 AxResource

Class *AxResource* represents any digital resource identified with a mime type, it can be an image, a document, an audio etc.


AxObjectElement
+getDIElement() : MPEG21Element
+setDIElelemt(in diElement : MPEG21Element)
AxContent
+setContentID(in id : string)
+getContentID() : string
AxResource
+getMimeType() : string
+setMimeType(in value : string)
+getRef() : string
+setRef(in ref : string)
+embedFile(in fileName : string)
+removeEmbeddedResource()
+getInputStream()

6.2.8 AxPublicMetadataTree

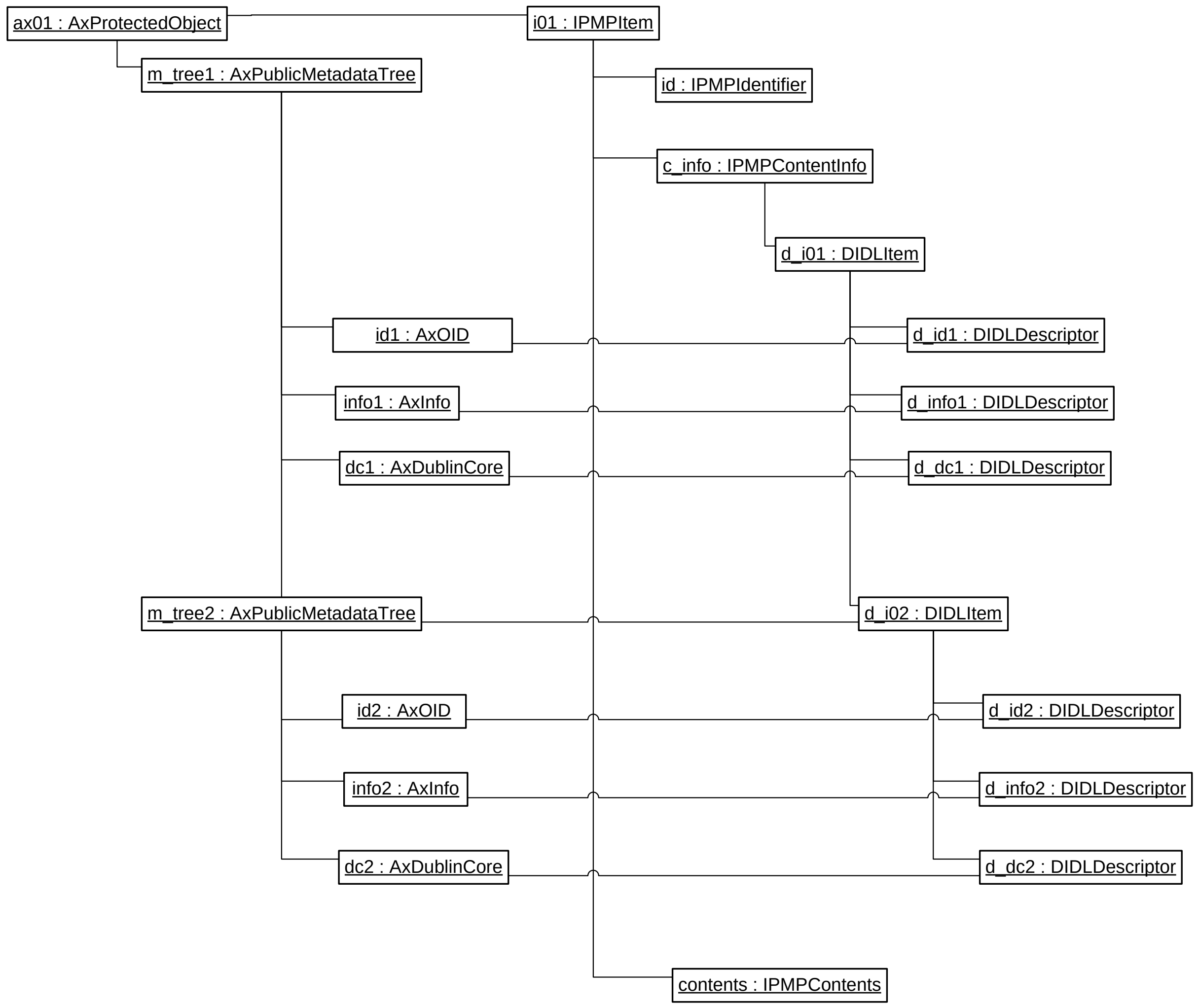
Class *AxPublicMetadataTree* represents a hierarchy of metadata associated to an AXMEDIS Object. This class allows accessing public metadata of the owner AXMEDIS object and the public metadata tree of the child objects. The *AxPublicMetadataTree* is designed to be a read-only information for the AxObject consumer, since only AxClearObject, AxProtectedObject and AxReferredObject are entitled to update a consistent metadata structure on the basis of the underlying content.


1
AxPublicMetadataTree
+getOwnerObject() : AxObject*
+setOwnerObject(in newOwner : AxObject)
+getChildMetadataTreeList() : std::vector<AxPublicMetadataTree*>
+addChildMetadataTree(in metadataTree : AxPublicMetadataTree)
+removeMetadataTree(in metadataTree : AxPublicMetadataTree)
+getMetdata() : std::vector<AxMetadata*>
+addMetadata(in metadata : AxMetadata)
+removeMetadata(in metadata : AxMetadata)
*
1
*
AxMetadata

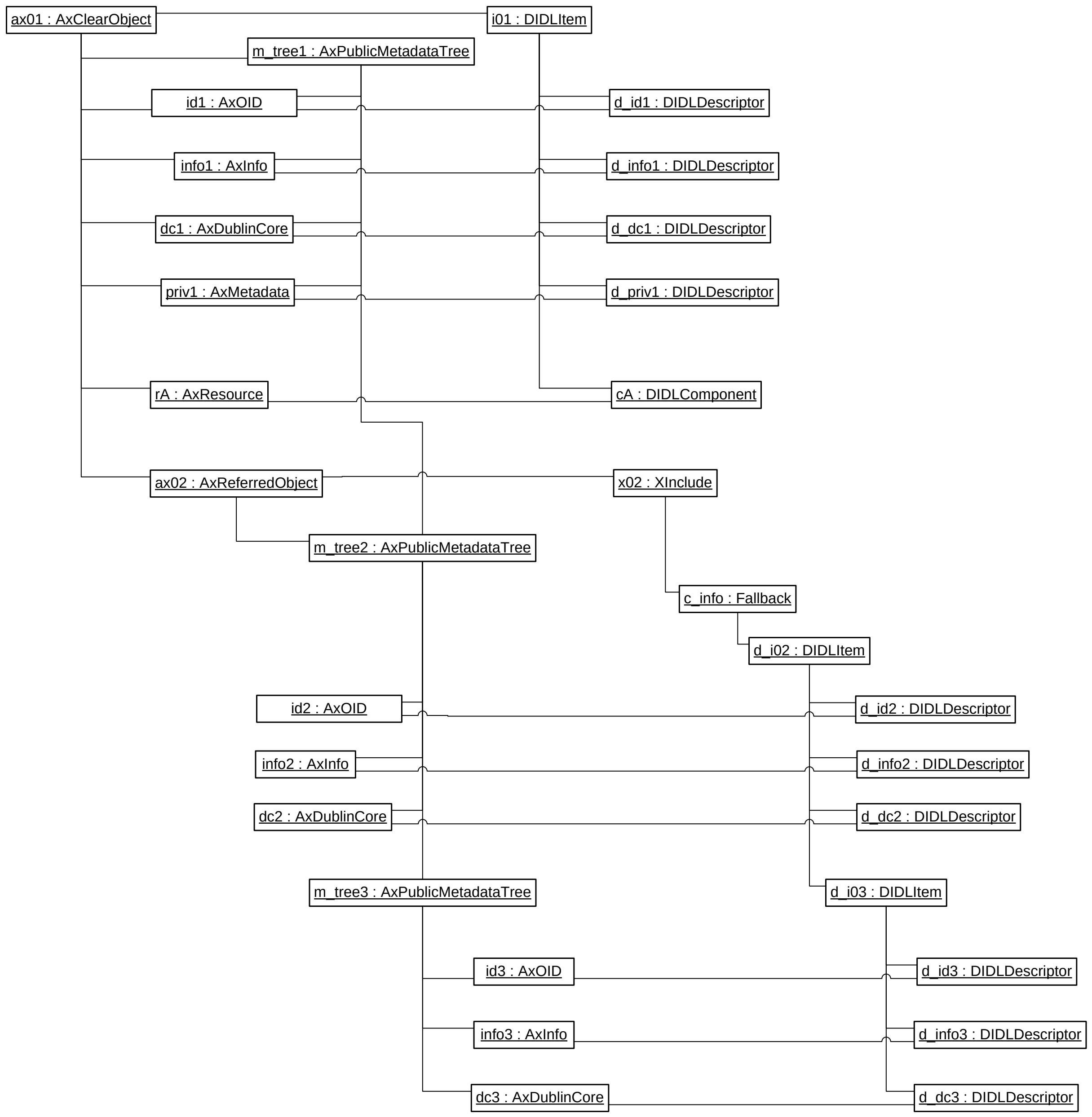
Please note that such a hierarchy does not duplicate the existing metadata elements, while it is simply a shortcut to reach them in an uniform manner with respect to object forms (e.g. protected).

In the following diagram an object configuration has been depicted in order to understand the direct link in order to access the metadata structure that describe the protected content.

Please note that the corresponding MPEG-21 element configuration has been depicted.

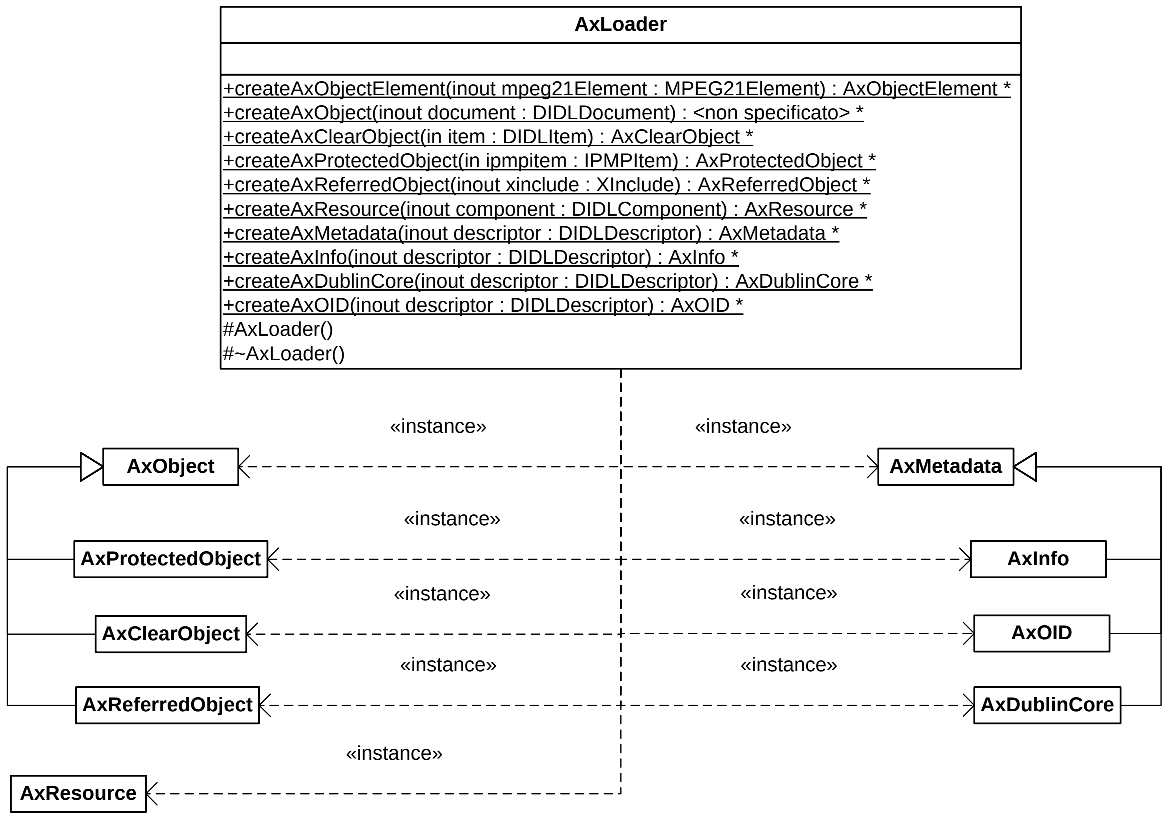


In the following diagram a clear-text object which contains a resource and referred object is presented. Please note how transparently the PublicMetadataTree is able to access to the whole metadata hierarchy without doubling the existing metadata elements of the clear-text object.



6.2.9 AxLoader

Class *AxLoader* has been designed in order to obtain an AXMEDIS Object Model once the MPEG-21 Model has been loaded by the suitable loader. In fact it is able to build AxObjectElement instances against corresponding MPEG21Element instances.


AxLoader
+createAxObjectElement(inout mpeg21Element : MPEG21Element) : AxObjectElement *
+createAxObject(inout document : DIDLDocument) : <non specificato> *
+createAxClearObject(in item : DIDLItem) : AxClearObject *
+createAxProtectedObject(in ipmpitem : IPMPItem) : AxProtectedObject *
+createAxReferredObject(inout xinclude : XInclude) : AxReferredObject *
+createAxResource(inout component : DIDLComponent) : AxResource *
+createAxMetadata(inout descriptor : DIDLDescriptor) : AxMetadata *
+createAxInfo(inout descriptor : DIDLDescriptor) : AxInfo *
+createAxDublinCore(inout descriptor : DIDLDescriptor) : AxDublinCore *
+createAxOID(inout descriptor : DIDLDescriptor) : AxOID *
#AxLoader()
#~AxLoader()
«instance»
AxObject
AxMetadata
AxProtectedObject
AxInfo
AxClearObject
AxOID
AxReferredObject
AxDublinCore
AxResource

It is a full static clas exposing all static functions to build any type of AxObjectElement. Any function is responsible of containing the knowledge about how the AXMEDIS Model is mapped onto the MPEG-21 DI (e.g. a AxClearTextObject expects an Item with a certain number of mandatory descriptors and some components or items).

Please note that the main entry point for loading AXMEDIS Object is *createAxObject* that is able to examine a whole DIDL Document representing the object. This function will use all the specific loading functions for clear-text, protected or referred objects and for loading metadata and resources.

7 AXMEDIS Object Preprocessor and Postprocessor (EPFL)

Module/Tool Profile		
MPEG-21 Object Model		
Responsible Name	Beilu Shao	
Responsible Partner	EPFL	
Status (proposed/approved)	Proposed	
Implemented/not implemented	Implemented	
Status of the implementation		
Executable or Library/module (Support)	Library	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows, Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Applications/PreProcessor/xinclude/ [DSI] Why is this located in Application directory while, as stated above, it should be a library?	
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)	Present	
Test cases location	https://cvs.axmedis.org/repos/Applications/PreProcessor/testXinclude/	
Usage of the AXMEDIS configuration manager (yes/no)	Yes	
Usage of the AXMEDIS Error Manager (yes/no)	Yes	
Major Problems not solved	-- std::istream* resolveXInclude(const XInclude& xi) is needed -- integration with loader--	
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
LibXML 2.0	LibXML 2.0	MIT License

7.1 General Description of the Module

MISSING

7.2 Module Design in terms of Classes

7.2.1 Reference Solver

The reference resolver works in the following situation: when an application wants to obtain a part of the object which is referred but not present in the object, reference resolver will provide the resolved part as an input. The function of the resolver like std::istream* resolveXInclude(const XInclude& xi) will be called to do this.
Reference integrator is a module for the automatic integration of the XInclude elements at loading time. It works in some way that the user never knows whether some parts of the object have been retrieved outside the object or not. The processes of resolving and integration are performed together in the loading phase. For instance, when the player loads an axmedis object, the loader will call the resolver to find referred elements which are not present in this object but exist in some other objects. After obtaining these referred elements, the integrator integrates these elements and transfers the original object with the new composite one.
The following diagram demonstrates an instance, e.g. a player, asks to resolve a reference. We assume that the object referenced is a resource contained inside an AXMEDIS object located in the AXMEDIS database. The resolver uses the Database Web Service to find and download the AXMEDIS object inside of which there is the target resource. Once the AXMEDIS object is downloaded, the reference resolver uses the Object Manager to extract the resource from the AXMEDIS Object. Finally, the resource is returned to the calling instance by means of a pointer or reference to an outputStream interface. If the referenced object is not in the

local file system, the reference resolver uses the Axmedis Database Web Service to localize and download the object.

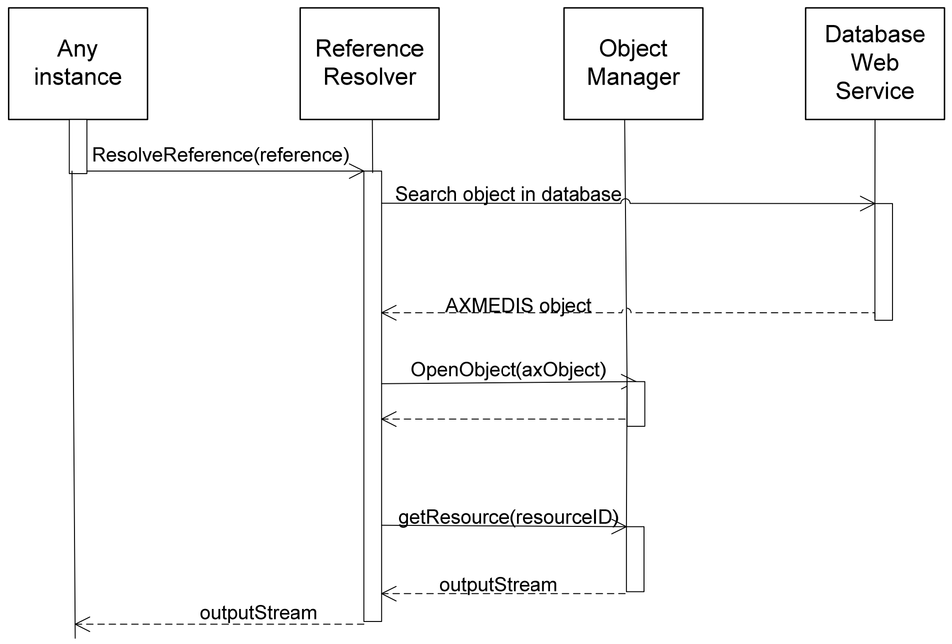


Fig. the working process of reference resolver

[DSI] As we told you, Reference Resolver should be able to resolve several kind of protocols not only AXMEDIS protocol. Did you have considered this? For example, http, ftp, etc…
What is the “outputStream” shown in the figure? Where does this stream write? Who uses it in order to write? I do not understand. I supposed reference resolver should return an “inputStream”? Am I wrong?

The following diagram demonstrates the process of integration. We assume that some instance wants to insert a resource which is located inside Object1 into Object2. The integrator first resolves Object1 with the resolver and extracts the target resource from Object1 and then it adds the referred part into Object2.

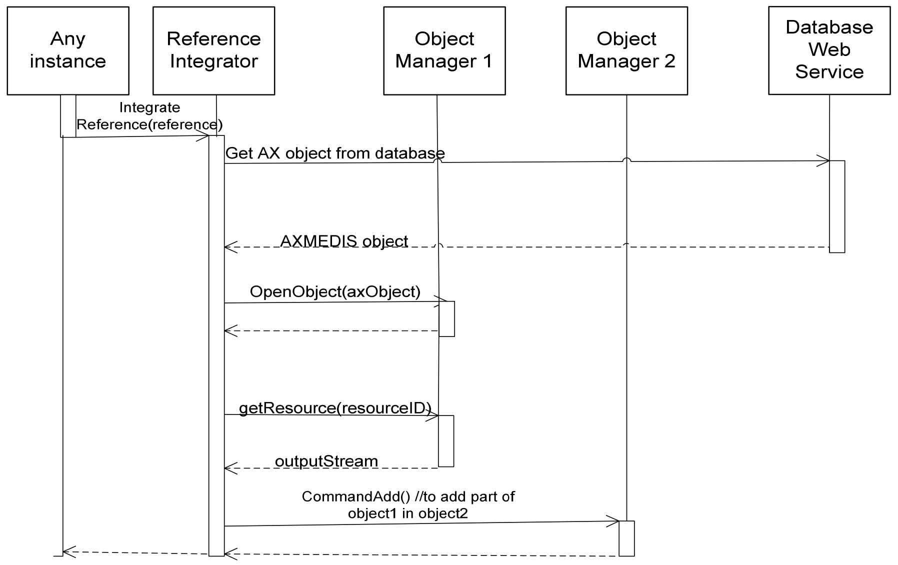


Fig. the working process of reference integrator
[DSI] It is not really understandable where Object Manager 1 and 2 comes from?

7.2.2 MPEG-21 Binarization

MISSING

7.3 Technical and Installation information

References to other major components needed	Axmedis loader
Problems not solved	• Integration with the loader • std::istream* resolveXInclude(const XInclude& xi) is needed
Configuration and execution context	

7.4 Draft User Manual

MISSING

7.5 Examples of usage

MISSING

7.6 Integration and compilation issues

MISSING

7.7 Errors reported and that may occur

MISSING

Error code	Description and rationales

7.8 Formal description of algorithm <…………..>

MISSING

name	
Method	
Description	
Input parameters	
Output parameters	

name

Method	
Description	
Input parameters	
Output parameters	

8 Protection Processor (DSI)

Module/Tool Profile		
Protection Processor		
Responsible Name	Andrea Vallotti, Leonardo Ortimini	
Responsible Partner	DSI	
Status (proposed/approved)		
Implemented/not implemented		
Status of the implementation		
Executable or Library/module (Support)		
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows/Linux	
Reference to the AXFW location of the source code demonstrator		
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)		
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

8.1 General Description of the Module

Protection Processor has mainly four tasks:

1. To register and certify an AXMEDIS tool containing the AXOM, e.g. editor, player, engine, etc…
2. To control software which uses sensible content and does not contain AXOM, e.g. plug-ins for fingerprint
3. To reveal attacks during tool execution, e.g. code debugging
4. To protect and un-protect elements of AXMEDIS object

In the following those aspects will be described and solutions are proposed for them. After that, class implementation and interaction will be described.

8.1.1 AXMEDIS tool registration and certification

In the following the needed information to reach an adequate level of protection will be identified and described. For each identified information, the responsible component is found out and the relationship to the other is depicted. At the end, the interactions and operations among all involved components to guarantee trustiness of a tool are described.

8.1.1.1 Software and hardware fingerprint

Given a device and an installation of an AXMEDIS compliant software on it, a fingerprint estimation of the whole tool (software/installation and hardware/device together) is possible. In particular, this section refers to software which embed an AXOM (and thus a Protection Processor), e.g. AXMEDIS editor, compositional/formatting engine, plug-ins for external viewer/editor, etc. The certification of plug-ins for AXOM (i.e. which do not contain an AXOM) is discussed later. By fingerprint estimation is intended an extraction of relevant information regarding the device and the most important files of the application (i.e. those files which are fundamental for the trustiness of the environment). The proposed fingerprint is composed by the following data:

1. for each hard-disk in the device: serial, controller revision number
2. for each processor in the device: name (i.e. the standard description of its features), serial (if reachable)
3. BIOS: name (comprehensive of the version), serial
4. optionally, for each network device: MAC address
5. operative system: name, version, installed upgrade (e.g. SP1), serial (e.g. product id)
6. manufacturer-defined name of all available components e.g. video device, audio device, motherboard, etc…
7. For each of the following files:
    a. executable file or library containing the AXOM
    b. main file of each plug-ins
    c. configuration files
    d. secure cache files

   the following features have to be collected:
    i. full name (path and file name)
    ii. physical position (e.g. if mass storage is an hard-disk it is the cluster index)
    iii. digest (e.g. MD5)
    iv. creation date and time
    v. last modification date and time
    vi. size
8. AXMEDIS Tool Type ID (AXTTID)
9. AXMEDIS Registration Tool ID (AXRTID) which is the digest of the main program executable file

Please, see “AXMEDIS Tool Fingerprint” section for fingerprint XML format details. All this information is estimated by the Protection Processor. It is responsible of their estimation, they are stored on the AXMEDIS Certifier and Supervisor and transmitted to it by means of the PMS Client. It is to point out that last modification date and time (previous point 7-v) and size (previous point 7-vi) should not be considered as parts of the fingerprint for those file which change during the lifetime of a tool, e.g. configuration file and secure cache, otherwise information stored on the AXCS and those estimated at runtime will hardly correspond.
The fingerprint is estimated and used for the following operation:
1. Tool certification – the tool has to transmit its complete fingerprint to the AXCS.
2. Tool re-certification – the tool has to transmit complete fingerprint to the AXCS in order to allow a deep fingerprint verification and support decision about tool re-certification of a disabled tool.
3. Tool re-verification – the tool has to transmit complete fingerprint to the AXCS in order to allow a deep fingerprint verification.

A digest of the fingerprint is used in other operations. Digests are generated by different algorithms in different cases
1. Tool Verification – the tool has to re-estimate the fingerprint and to transmit a digest of it to the AXCS. Calculation method: SHA1
2. Grant Authorisation – tool fingerprints digest is included in the new generated action log describing the requesting action. Calculation method: SHA1
3. Offline Tool Verification – tool fingerprints digest is created to compare with the enabling code received as a return of Tool Certification operation. The digest is created by means of SHA512 algorithm

8.1.1.2 Tool certificate

Tool certificate is issued by an AXCS to the tool itself when the latter certifies itself at the first activation. The certificate is formatted in the X.509v3 format. It contains the following information:

Version	2 (that is X.509v3 is identified by this value)
SerialNumber	The serial number of the certificate. It is defined by the issuer which is the AXCS
Signature	The encryption algorithm used to encrypt the signature. In this case, the signature algorithm is the RSA with SHA-1 which is identified by the following ASN.1 object identifier: sha-1WithRSAEncryption OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 5 }

Issuer	The AXCS which issued the certificate expressed in the X.500 format
Validity	The validity period of the certificate which for the AXMEDIS purposes could be one/five year since the issuing time
Subject	Identified who or what receives the certificate. In this case it is the AXTID which identifies the tool
SubjectPublicKeyInfo	The encryption algorithm used to generate the public key and the public key itself. In this case, the used encryption algorithm is the RSA which is identified by the following ASN.1 object identifier: rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1 }
Encrypted	The encrypted digital signature of the certificate
Extensions	Extension to the certificate. This field can contain the activation code for the tool and other information. See DE3.1.2.2.13 regarding OIDs tree assigned by IANA to AXMEDIS

Before generating the certificate, the AXCS requests to the Key Generator (component of the PMS Server) to generate a key pair (public/private keys). As said above, the AXCS inserts the public key into the tool certificate and sends certificate and private key to the tool. They are packed together as stated in PKCS #12. The integrity of transmitted information is guaranteed by a signature using the private key of the AXCS (see PKCS #12 – Public-key integrity mode). The privacy of that information is guaranteed because it is encrypted using the public key of the user who is registering the tool (see PKCS #12 – Public-key privacy mode).
The certificate and the private key are stored in the specific device certificate repository and in the PMS Client secure cache. In that way, their consistence can be tested every time the tool is used, the Protection Processor is in charge of doing that check.
The private key corresponding to the tool certificate is marked as un-extractable (see PKCS #11), i.e. it can be used on the device where it have been stored on the first time but it cannot be exported on other devices, not even by the device administrator. The tool certificate has to be accessible from all the user of a device to avoid multiple registration of the same tool by different users.

8.1.1.3 Tool Registration Certificate

Tool Registration Certificate is issued by Tool Creator and included in tool installation. This certificate provide evidence of tool authenticity and is used in certification operation of the tool

Version	2 (that is X.509v3 is identified by this value)
SerialNumber	The serial number of the certificate. It is defined by the issuer which is the AXCS
Signature	The encryption algorithm used to encrypt the signature. In this case, the signature algorithm is the RSA with SHA-1 identified by the following ASN.1 object identifier: sha-1WithRSAEncryption OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 5 }
Issuer	The AXCS which issued the certificate expressed in the X.500 format
Validity	The validity period of the certificate which indicates the expiration date of validity of the tool
Subject	The AXRTID of the tool used for the first certification
SubjectPublicKeyInfo	The encryption algorithm used to generate the public key and the public key itself. In this case, the used encryption algorithm is the RSA which is identified by the following ASN.1 object identifier: rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1 }
Encrypted	The encrypted digital signature of the certificate
Extensions	Extension to the certificate.

This certificate is included in tool installation and contains tool information needed to certify the tool. It is not stored in the system certificate repository nor in the secure cache. Its presence is bounded to a predefined path in the installation folder. Removal of this certificate from the right path will be treated as a tampering action, thus disabling the tool until a recovery action is performed. Security about replacement of this certificate is granted by his signature validated through public key cryptography .

8.1.1.4 User certificate

User certificate is issued by an AXCS to the user at the registration time. It is useful to recognize who is using a tool or is making action on an object. The certificate is formatted in the X.509v3 format. It contains the following information:

Version	2 (that is X.509v3 is identified by this value)
SerialNumber	The serial number of the certificate. It is defined by the issuer which is the AXCS
Signature	The encryption algorithm used to encrypt the signature. In this case, the signature algorithm is the RSA with SHA-1 identified by the following ASN.1 object identifier: sha-1WithRSAEncryption OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 5 }
Issuer	The AXCS which issued the certificate expressed in the X.500 format
Validity	The validity period of the certificate which for the AXMEDIS purposes could be one/five year since the issuing time
Subject	The AXUID of the registered user
SubjectPublicKeyInfo	The encryption algorithm used to generate the public key and the public key itself. In this case, the used encryption algorithm is the RSA which is identified by the following ASN.1 object identifier: rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1 }
Encrypted	The encrypted digital signature of the certificate
Extensions	Extension to the certificate. This field can contain the email of the user and other information

Before generating the certificate, the AXCS requests to the Key Generator (component of the PMS Server) to generate a key pair (public/private keys). As said above, the AXCS inserts the public key into the user certificate and sends certificate and private key to the user. They are packed together as stated in PKCS #12. The integrity of transmitted information is guaranteed by a signature using the private key of the AXCS (see PKCS #12 – Public-key integrity mode). The privacy of that information is guaranteed because it is encrypted using something the user knows, e.g. a password given by the user during the registration (see PKCS #12 – Password privacy mode).

The certificate can be delivered to the user using the email address he gave as reference during the registration.

Another suitable solution is to send certificate and private key (possibly packed together) to the user at the end of the registration through a secure channel, e.g. using secure protocol https (http on SSL) instead of the un-secure http.

The certificate and the private key are stored in the specific device certificate repository and in the PMS Client secure cache. In that way, their consistence can be tested every time the tool is used, the Protection Processor is in charge of doing this check.

User certificate import and export have to be someway controlled, i.e. if a user tries to export a certificate the system have to check if he/she is the subject the certificate was issued to. For example, the PKCS #12 formatted packet the user receives at registration time is protected by the password he/she gave at that time. The main issue is to be sure that an user cannot extract a certificate installed on a device without proving he/she is really the certificate owner.

8.1.1.5 User Identifier/Identification

As stated in the previous sub-section, the AXUID (AXMEDIS User Identifier) of a user is contained in his/her certificate. On a multi-user device (e.g. a personal computer with MS Windows), several certificates for different users can be stored. In that scenario, the main issue is how an AXMEDIS tool can get the right certificate (thus the right AXUID) for the current user.

Usually a multi-user device manages one certificate repository for each system users. In that case, the AXMEDIS tool can look for a certificate issued by the AXCS in the current user certificate repository and use that certificate, if exists, as reference certificate for the current system user. Note that each user should have only one personal certificate issued by the AXCS.

The Protection Processor is in charge of looking for the certificate in the system repository and to get the contained AXUID which univocally identified the AXMEDIS user.

8.1.1.6 Date and time

Since licenses can be based on date/time condition, date and time control is a fundamental issue to be addressed.
Date and time have to be measured at the registration and every time an action is made. These measures (or at least the last one) should be stored in a trusted place which can be suitably the AXCS.
In that way, during certification and authentication, the AXCS can verify if the nominal sequence of the actions (i.e. the sequence according to which actions were stored) matches the timeline sequence (obtainable using the measured date and time).
Each record of the history of actions (see below) have to be labelled with the date and time of execution.

8.1.1.7 Action history

Action history permits to control if the user made some not-allowed actions during off-line working. That is, when the tool is used on-line (i.e. it can freely communicate with the PMS Server) every time the user request to do an action on an object the request is processed and, if it is authorized, an action log (containing the action type, the user id, the tool id, date and time, etc…) is sent to the AXCS. On the other hand, if for a while the tool has been used off-line (i.e. without communication to and from the PMS Server), each authorized action generates an action log which is cached on the device (within the a secure cache). As soon as a connection to the PMS Server is available, the set of cached action logs is sent to it.
The information involved in this kind of control is:

1. the action logs which are stored on the device
2. a hash (e.g. MD5) of the past history of actions which is stored on the AXCS and on the device.

Every time the tool can communicate with the PMS Server, the former sends to the latter the cached action logs (or a single action log if it is working on-line) and the updated hash of the history, i.e. a hash which is function of the old hash and the cached action logs. Managing of action history and its verification is complete responsibility of PMS (client and server) that implements internal methods used to store action logs and calculate history. Protection processor is in charge of providing to the PMS all data PMS cannot recover itself in order to fill action logs (see below). Data needed to create an action log are provided to the PMS Client by the Protection Processor every time the latter receives a Grant Authorisation Request.

8.1.1.8 Enabling code

Enabling code is issued by the AXCS to the tool during the registration (it is stored in the tool certificate). The code is function of the fingerprint of the device, and is used to compare current tool fingerprint vs fingerprint registered at verification time. This check is done offline while enabling code is calculated at certification time. The calculation is made using a one way hash function, SHA512, applied on the fingerprints XML representation. With this method any changes in current tool fingerprint is detected, enabling protection processor to perform further verify and recovery actions.

8.1.1.9 Trustiness of a tool

In this sub-section, a feasible mechanism to test the trustiness of a tool (the pair application/device) is depicted. The information described above is involved in that check.

		Protection Processor	**PMS Client**	**AXCS**
Fingerprint	registration	estimates FP_0;	-	stores FP_0;
	time t	estimates FP_t;	joins FP_t to action logs;	verifies FP_t w.r.t. FP_0;
Fingerprint digest		calculate;	-	calculate; verifies FP_t digest w.r.t. FP_0 digest;
Tool certificate		verifies; store in system repository;	stores;	generates keys; generates certificate;
User certificate		verifies; store in system repository;	stores;	generates keys; generates certificate;
User Identifier		gets from user certificate;	joins to action logs;	-
Tool Identifier		gets from tool certificate;	joins to action logs;	-

Date and time	measures;	joins to action logs;	verifies; stores date and time of the last action log;
Action history	-	estimate new history hash; manages action logs; stores history hash;	verifies action logs; stores new history hash;
Enabling code	tests;	-	generates as function of FP_0; put into the tool certificate

In the above table, all task of each involved component are reported with respect to the needed information, i.e. which component generates/estimates an information and which other uses/verifies it.
If a tool have been successful registered by an user, at runtime the tool is considered trusted if, and only if, all the following tests succeed:

1. The user certificate is valid, i.e. it is digitally signed by the certification authority (which is an AXCS). Control performed by the Protection Processor;
2. The tool certificate is valid. Control performed by the Protection Processor;
3. The enabling code contained in the tool certificate and the re-estimated fingerprint are compatible according to the test described above. Control performed by the Protection Processor;
4. The re-estimated fingerprint digest matches the registration-time fingerprint digest which is stored on the AXCS. Control performed by the AXCS;
5. The history hash calculated by the PMS Client, using the old one and the action logs (both securely stored by the PMS Client into the Secure Cache), coincides with the same hash estimated by the AXCS using its own copy of the old history hash and the action logs received;
6. The execution date and time of all the action listed in the action logs are consistent each other (i.e. the nominal order matches the timeline order) and with respect to the last action execution date and time stored on the AXCS;

8.1.1.10 Certified software

AXMEDIS tool features can be enriched by means of plug-ins. Usually plug-ins are pieces of software exposing functions for specific purposes. Thus it is not suitable to equip a (likely) simple piece of software (as a plug-in can be) with the AXOM to ensure DRM respect. Nevertheless, a plug-in can be used to manipulate DRM-liable data, e.g. a fingerprint extractor plug-in.
To tackle this lack of security, each pieces of software created to enrich AXOM functionalities via plug-in and to manage content have to be previously registered by the AXCS (see DE3.1.2.2.13, AXCS Tool-Offline Registration Web Application). Registration process results in the creation of a signed description of the plug-in, from now on let call it “manifest”. For more details about the manifest see Plug-in description format in DE3.1.2.2.4.
Every time a piece of protected content have to be passed to an external software, the Protection Processor controls if the binary file containing the software corresponds to the signature contained in the manifest. The latter has to be placed in the same directory where the plug-in is placed.

8.1.1.11 Execution controls

Protection Processor should inhibit AXMEDIS tool functioning as soon as it reveals that tool execution is under tracking. That is, Protection Processor should test if execution is under debug. That control is useful to avoid malicious user to disclose all security mechanism used within AXOM to guarantee trustiness of the environment.
This feature is system-dependant and it will be specifically developed for each platform AXMEDIS Tool should be capable to run on.

8.1.2 Robustness against malicious user actions

In the following table are listed several feasible actions which a malicious user can carry out to illegally use DRM-controlled content. Each attack is shortly described and

Feasible attacks description	**System avoid/detect capabilities**
Migrate an AXMEDIS tool installation from the registered	The software and hardware fingerprints of the two devices are not equal, even if the two device has been assembled with same

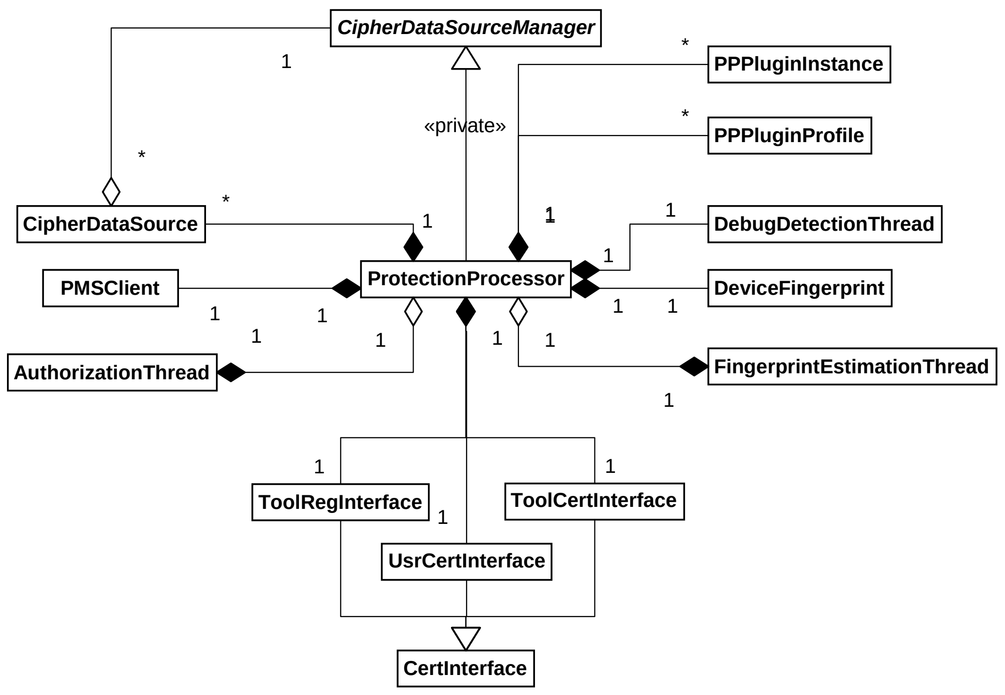
device to another	components, thus enabling code will fail.
Software under debug during execution	Protection processor monitors if the application embedding it is under debug, if it is protection processor will clear all sensible data on the device. Moreover, if this monitoring system is deactivated modifying the appliacation file, the fingerprint of the tool will change and it will stop working.
Change system time to use time-constrained content out of DRM-allowed period	This attack will work until the device works off-line. As soon as the tool can communicate with an AXCS thus, if IPR infringement is detected by the latter, the tool will be disabled and the sensible information will be cleared.
Migration of tool certificate	There are two countermeasurs against this action: • The certificate is stored in two different places: the system certificate repositorty and the secure cache of PMS Client. The user can move the one stored in the system repository to another device but a simple consistency control between the two copies will defeat the attack • The certificate contains the signed activation code used to verify the tool integrity. Therefore, a certificate copied by another device will never work
An user uses content by illegally exposing the identity of another user	If a user exposes all the credential of another user the attack will be not detect. Nevertheless, this attack is similar to the usage of stolen credit card and it cannot be considered solvable problem: the privateness of the personal certificate is exclusive interest of the user himself/herself
Change pieces of hardware	Depending on the wideness of changes, the tool is disabled and a re-certify action is required to restore the tool's functionalities
Create a backup of the newly-installed AXMEDIS tool. Illegally use content off-line (i.e. without external controls). Restore the virgin version of the tool before re-connect to the network.	If a ghost image of the hard disk is made before using the content and then it is restored on the original hard-drive after content has been illegally used off-line, the attack will not be revealed by the protection processor or the AXCS. Nevertheless, this action requires such technical support and knowledge that it is not applicable by the common user and it do not allow a spread distribution of unprotected content. Thus it is not of interest for us.

8.2 Module Design in terms of Classes

8.2.1 General Structure

The following figure sketch the general architecture of Protection Processor.
ProtectionProcessor is the main class of the Protection Processor package. It provides protection services to the Data Model Support, the Command Manager and the PMS Client. As described in the previous sub-sections, it is in charge of:

1. certifying and verifying an AXMEDIS tool
2. controlling software which uses sensible content and does not contain AXOM
3. revealing attacks during tool execution
4. protecting and un-protecting AXMEDIS elements

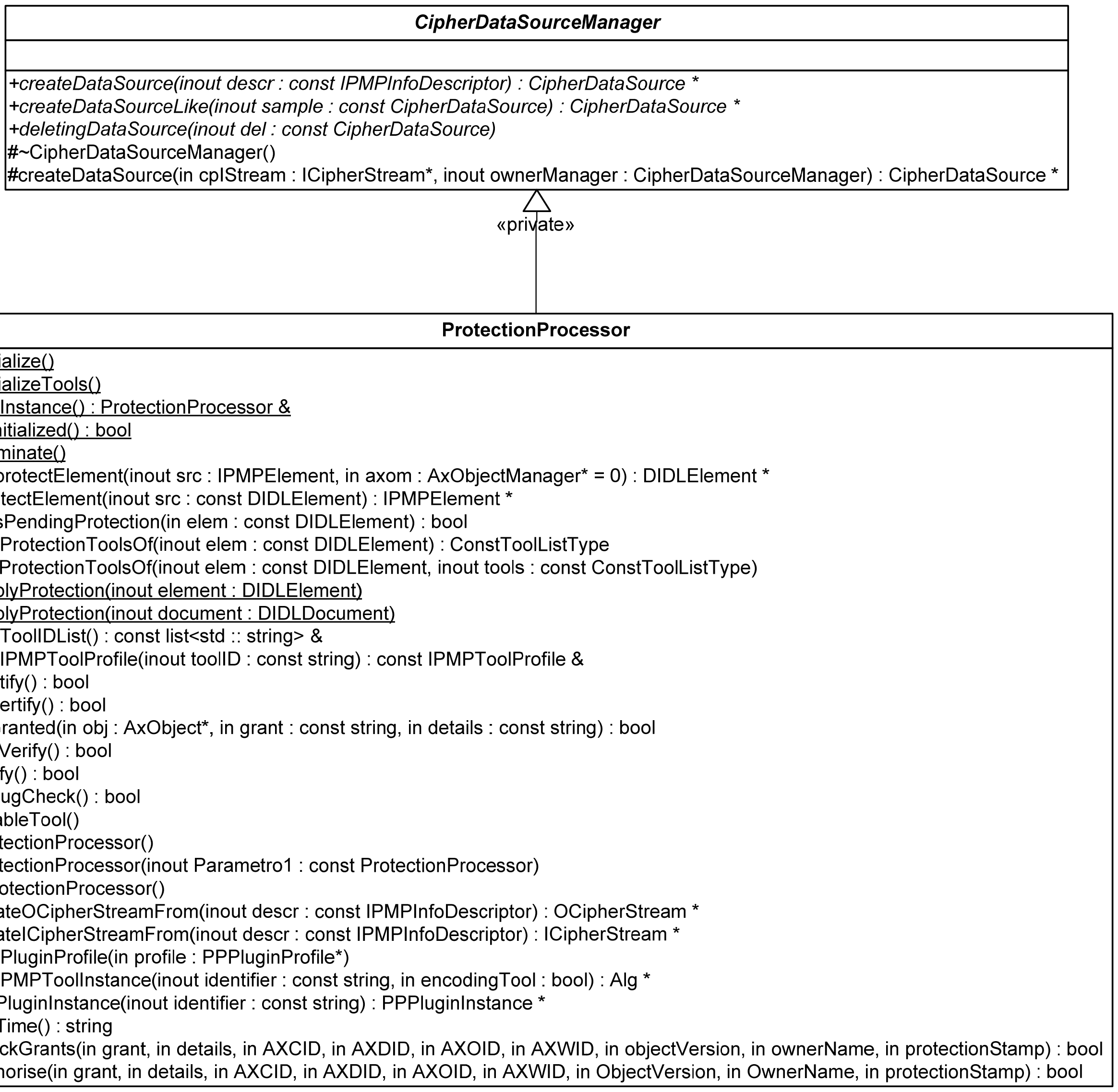


The above figure shows the general architecture of Protection Processor module. As depicted there, ProtectionProcessor is the coordinator of all other classes. Moreover, it is the direct interface trough which AXOM and AXMEDIS Tools can exploit all security features.
In the following subsections, each classes (or set of classes) will be described and their methods will be explained.

8.2.2 ProtectionProcessor and CipherDataSourceManager

As depicted in the picture above, *ProtectionProcessor* class is the core of this package. It is the connection point among all other utilities (fingerprint estimator, control thread, protection tools, etc…). It also implements a private interface (*CipherDataSourceManager*) which is needed for an easy and transparent use of *CipherDataSource*. Please notice that Protection Processor is a singleton, i.e. only one instance of *ProtectionProcessor* can be instantiated for each tool.
In the following the relevant methods of *ProtectionProcessor* are described:

- *initialize* – it is a static function which initializes the unique *ProtectionProcessor* instance and part of its related classes. An AXMEDIS compliant tool has to call this function after the initialization of Configuration Manager and before any other action.
- *initializeTools* – it is a static function which initializes the list of available protection tools. It has to be called after Protection Processor and Plug-in Manager initializations and before any other action.
- *getInstance* – it is a static function which allows retrieving the unique instance of *ProtectionProcessor* (Singleton design pattern). If the method is invoked before Protection Processor has been initialized, it throws an exeception.
- *isInitialized* – it test whether te Protection Processor has been already initialized or not.
- *terminate* – it is a static function which terminates the Protection Processor. It has to be called when no more security features are needed.



```
CipherDataSourceManager

+createDataSource(inout descr : const IPMPInfoDescriptor) : CipherDataSource *
+createDataSourceLike(inout sample : const CipherDataSource) : CipherDataSource *
+deletingDataSource(inout del : const CipherDataSource)
#~CipherDataSourceManager()
#createDataSource(in cplStream : ICipherStream*, inout ownerManager : CipherDataSourceManager) : CipherDataSource *
```

```
ProtectionProcessor
+initialize()
+initializeTools()
+getInstance() : ProtectionProcessor &
+isInitialized() : bool
+terminate()
+unprotectElement(inout src : IPMPElement, in axom : AxObjectManager* = 0) : DIDLElement *
+protectElement(inout src : const DIDLElement) : IPMPElement *
+hasPendingProtection(in elem : const DIDLElement) : bool
+getProtectionToolsOf(inout elem : const DIDLElement) : ConstToolListType
+setProtectionToolsOf(inout elem : const DIDLElement, inout tools : const ConstToolListType)
+applyProtection(inout element : DIDLElement)
+applyProtection(inout document : DIDLDocument)
+getToolIDList() : const list<std :: string> &
+getIPMPToolProfile(inout toolID : const string) : const IPMPToolProfile &
+certify() : bool
+recertify() : bool
+isGranted(in obj : AxObject*, in grant : const string, in details : const string) : bool
-selfVerify() : bool
-verify() : bool
-debugCheck() : bool
-disableTool()
-ProtectionProcessor()
-ProtectionProcessor(inout Parametro1 : const ProtectionProcessor)
-~ProtectionProcessor()
-createOCipherStreamFrom(inout descr : const IPMPInfoDescriptor) : OCipherStream *
-createICipherStreamFrom(inout descr : const IPMPInfoDescriptor) : ICipherStream *
-addPluginProfile(in profile : PPPluginProfile*)
-getIPMPToolInstance(inout identifier : const string, in encodingTool : bool) : Alg *
-getPluginInstance(inout identifier : const string) : PPPluginInstance *
-getTime() : string
-checkGrants(in grant, in details, in AXCID, in AXDID, in AXOID, in AXWID, in objectVersion, in ownerName, in protectionStamp) : bool
-authorise(in grant, in details, in AXCID, in AXDID, in AXOID, in AXWID, in ObjectVersion, in OwnerName, in protectionStamp) : bool
```

- *unprotectElement* – it unprotects the given *IPMPElement* returning the corresponding *DIDLElement*. Needed Protection Information is retrieved from the protected element itself or from the Secure Cache (on the base of AXOID, version and Protection Stamp).
- *protectElement* – it protects the given *DIDLElement* returning the corresponding *IPMPElement*. In order to protect the clear-text element, Protection Processor uses:
    - Protection Information previously set by the User (e.g. through the AXMEDIS Editor);
    - the Protection Information previously used to unprotect the original *IPMPElement*, only if the given *DIDLElement* derives from a previously unprotected *IPMPElement*
- *hasPendingProtection* – it test whether there is a list of Protection Tools associated with the given DIDLElement
- *getProtectionToolsOf* – it returns the list of Protection Tools associated with a given DIDLElement. This list is returned if and only if Protection Information has been set by the User or it was contained in the original *IPMPElement*.
- *setProtectionToolsOf* – it allows associating a list of Protection Tools to a given *DIDLElement*. This method overrides the old Protection Tool list.
- *applyProtection* – these two static functions recursively protect all those elements which have pending Protection Information starting from the given root element (which can be a *DIDLDocument* or a generic *DIDLElement*). These functions visit the model tree in a post-ordered way.
- *getToolIDList* – it returns the list of all available Protection Tool identifiers.

- *getIPMPToolProfile* – if the required profile exists, this method retrieves the profile of the Protection Tool given its identifier.
- *certify* – this function is called at first tool activation to certify it, sending the required information to the AXCS.
- *recertify* – this function is called whenever a user needs to recover a disabled tool. The method try to recertify the tool through a call to PMS, and if this operation give positive result, the tool is enabled once again. Elsewhere the tool remains disabled (no other operation is possible)
- *isGranted* – this function has to be called before the execution of any action on an AXMEDIS object. This function is called by the AXMEDIS Object Manager (see related section). This method receives from upper levels information needed to check, through PMS Client interface, if the requested action is performable in accordance with the licenses that user/tool pair owns. Calling this method starts a chain of operations, leading to a complete tool, user and environment verification. Direct verification functionalities are hidden by this public method.

The above methods are the public interface of ProtectionProcessor. It also implement the following functions which are used internally:

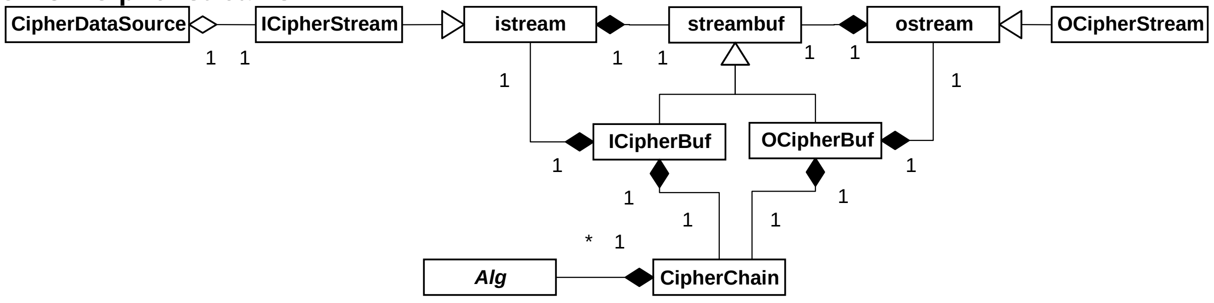
- *selfVerify* – check tool fingerprint vs. enabling code comparing the SHA-512 of xml tool fingerprint with the enabling code. If the test fails, the tool is blocked, all sensible data are removed and the tool needs to be recertified.
- *verify* – interface between PMS Client verify method and Protection Processor. In this context the tool fingerprint digest , used for verification, is calculated using SHA-1 on extracted tool fingerprint
- *debugCheck* – This method reveals if some debug application is active on the tool. If a positive result happens, the tool is disabled.
- *disableTool* – Through a call at this method protection processor block the tool, deleting all sensible data contained in secure cache of PMS Client and denies all further operations on the tool. The only means to recover a disabled tool is to *recertify* it.
- *Constructor* and *Destructor* – they are private since *ProtectionProcessor* can be instantiated and deleted only by static functions *initialize* and *terminate* respectively.
- *createOCipherStream* – this private functions creates an output cipher stream given an *IPMPInfoDescriptor*. See next subsections for details.
- *createICipherStream* – this private functions creates an input de-cipher stream given an *IPMPInfoDescriptor*. See next subsections for details.

Moreover, as stated before, *ProtectionProcessor* implements the interface *CipherDataSourceManager* in order to be able to manage *CipherDataSource* obtained by protected elements. *CipherDataSourceManager* declares the following methods:

- *createDataSource* – it creates a *CipherDataSource* given the Protection Information.
- *createDataSourceLike* – it creates a *CipherDataSource* starting from a given one. This method does not make an exact copy of the passed data source. Instead it creates a cipher data source which conforms to the Protection Information of the given one.
- *deletingDataSource*- this function is called in the destructor of *CipherDataSource* in order to warn the related *CipherDataSourceManager* that it has been deleted.

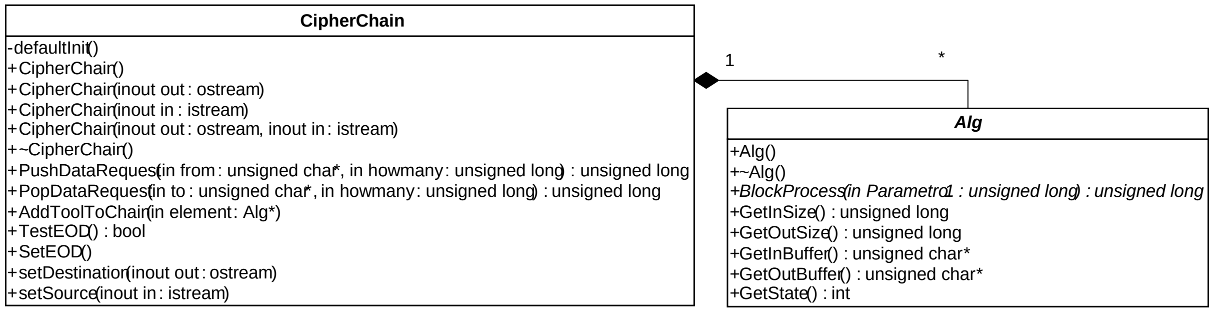
The other private functions are simply utilities which are not explained here.

8.2.3 Cipher streams



In order to easily integrate the ciphering and de-ciphering mechanisms with the other modules (such as MPEG-21 Loader and Saver, etc…), a stream-based approach has been adopted. That is, given a blob of

bytes representing a protected DIDL element, and the related Protection Information, *ProtectionProcessor* is able to create an instance of *ICipherStream* class. As show in the figure above, *ICipherStream* is derived from *std::istream* class of the STL thus exposing the same interface. An instance of this class allows reading, using the common C++ stream functions, the clear-text version of the blob of bytes.
On the other hand, an instance of *OcipherStream*, which derives from *std::ostream*, protects data written through it on the basis of given Protection Information. Therefore, an instance of *OCipherStream* could be used as normal C++ stream but it also performs protection actions on the written data.
Both ciphering streams act as Decorator (see Design Pattern) for other C++ streams. For example, a standard C++ file input stream (i.e. *std::ifstream*) is opened on a file containing protected blob of bytes. This stream is wrapped by an instance of *ICipherStream*. Every time the user request some data, the cipher stream reads (protected) data using the wrapped file stream and, on the basis of its settings, unprotect them thus returning clear-text data to the user.


CipherChain
-defaultInit()
+CipherChain()
+CipherChain(inout out : ostream)
+CipherChain(inout in : istream)
+CipherChain(inout out : ostream, inout in : istream)
+~CipherChain()
+PushDataRequest(in from : unsigned char*, in howmany : unsigned long) : unsigned long
+PopDataRequest(in to : unsigned char*, in howmany : unsigned long) : unsigned long
+AddToolToChain(in element : Alg*)
+TestEOD() : bool
+SetEOD()
+setDestination(inout out : ostream)
+setSource(inout in : istream)
1
*
Alg
+Alg()
+~Alg()
+BlockProcess(in Parametro1 : unsigned long) : unsigned long
+GetInSize() : unsigned long
+GetOutSize() : unsigned long
+GetInBuffer() : unsigned char*
+GetOutBuffer() : unsigned char*
+GetState() : int

The main classes for the realization of cipher streams are *Alg* and *CipherChain*. *Alg* is an abstract class which is the base class of all Protection Tools. That is, each Protection Tool has to expose the interface provided by *Alg*. In particular, a given Protection Tool has to implement its own specific *BlockProcess* function which, given a block of bytes, has to produce the clear-text version of this block w.r.t. the implemented algorithm. In this way, different Protection Tools can be treated in a common manner.
CipherChain is the coordinator of a set of Protection Tools, i.e. it is in charge of providing to each tools the appropriate amount of data and applying to each block of bytes the required tools in the correct order. *CipherChain* can work in two different modalities: push and pop. The push mode is used by the *OCipherStream* class since usually the user pushes the data to be protected into the stream. On the other hand, the pop mode is used by the *ICipherStream*.
For the sake of completeness, it is worth to point out that *ICipherBuf* and *OCipherBuf* are the real owners of *CipherChain* instances. In fact, due to the STL architecture, these are the classes which customize the behaviour of streams.

8.2.4 Protection Information interpretation

In the AXMEDIS project, Protection Information is represented using the MPEG-21 IPMP standard as explained in the section "AXMEDIS Protection Info". This format can be easily interpreted using the MPEG-21 Loader (see related section) since the MPEG-21 IPMP XML Schema has been integrated in the *MPEG21Element* hierarchy. Once created the object model of a given piece of Protection Information, the *ProtectionProcessor* can create cipher streams. In particular, it is able to retrieve Protection Tool instances from loaded Plug-ins on the basis of their profiles, and to combine them together (see section 8.2.11). After that, this list of tool instances will be used by the cipher streams in order to protect (or unprotect) content.
Protection Information may include:

- How each element of an AXMEDIS object has been protected, i.e. encrypted, encoded, compressed and scrambled
- How each chunk of a resource has been protected, e.g. specifying that a given set of protection tools has to be applied from byte X to byte Y of a given resource (and not to the whole resource). In that way, different protection can be applied to a resource along its consumption.
- It is based on an XML schema which allows to describe sort of protection procedures

Different tools can be selected by means unique identifiers defined in the framework. The tools can on the device, downloaded from some server or directly contained within the protection info.

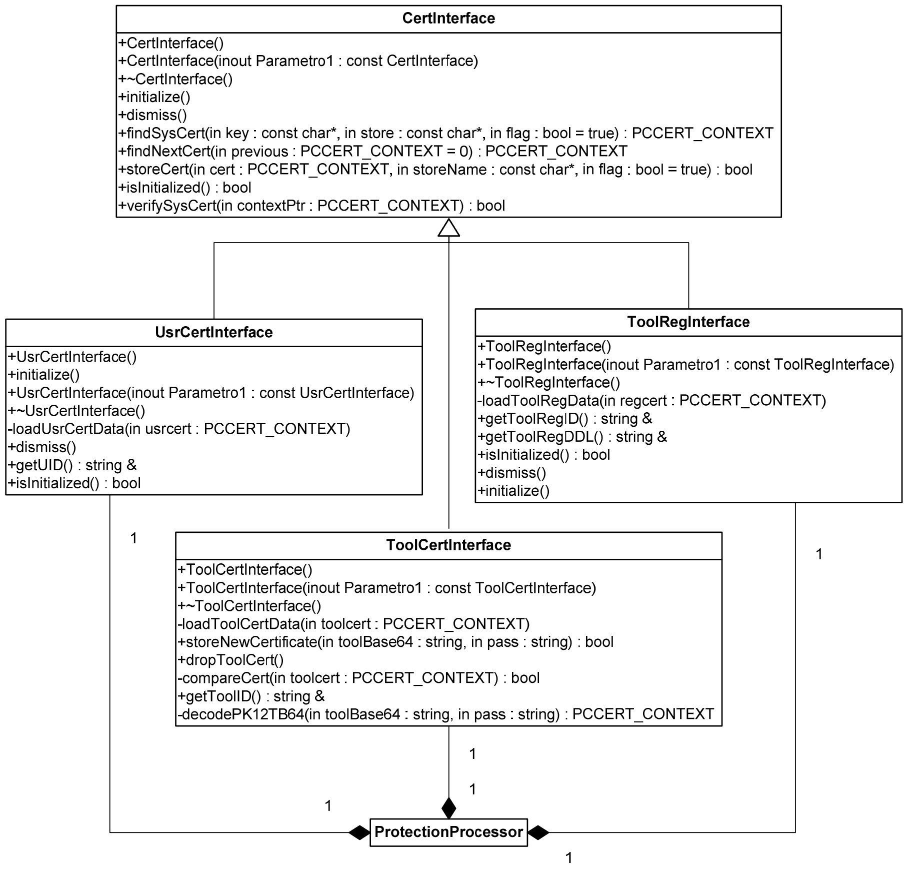
The decomposition and the application of the dynamic IPMP can be performed at level of data segments and blocks and may change over time if different coding protection models are enforced into the same resource. This is quite different with respect to dynamically change the IPMP rules when the resource is streamed. Note that the single resource may have different tools, different keys, different combination of tools, etc…

8.2.5 Threads

Protection Processor module contains three thread classes. These have been designed in order to do specific functions in an asynchronous way w.r.t. the execution of the program. That is:

- *DebugDetectionThread* – at random time instants, it tests whether the application is under debug or not. It simply calls the *debugCheck* method provided by *ProtectionProcessor*.
- *FingerprintEstimationThread* – at random time instants, it estimates the fingerprint and its hashes which are needed for the control during program execution. This has been designed since fingerprint estimation is a time consuming task which cannot be done every time an action is done
- *AuthorizationThread* – at random time instants, it calls the verify function provided by *ProtectionProcessor*. In that way the tool is verified even when any action is done.

8.2.6 Certificate Interface Module


CertInterface
+CertInterface()
+CertInterface(inout Parametro1 : const CertInterface)
+~CertInterface()
+initialize()
+dismiss()
+findSysCert(in key : const char*, in store : const char*, in flag : bool = true) : PCCERT_CONTEXT
+findNextCert(in previous : PCCERT_CONTEXT = 0) : PCCERT_CONTEXT
+storeCert(in cert : PCCERT_CONTEXT, in storeName : const char*, in flag : bool = true) : bool
+isInitialized() : bool
+verifySysCert(in contextPtr : PCCERT_CONTEXT) : bool
UsrCertInterface
+UsrCertInterface()
+initialize()
+UsrCertInterface(inout Parametro1 : const UsrCertInterface)
+~UsrCertInterface()
-loadUsrCertData(in usrcert : PCCERT_CONTEXT)
+dismiss()
+getUID() : string &
+isInitialized() : bool
ToolRegInterface
+ToolRegInterface()
+ToolRegInterface(inout Parametro1 : const ToolRegInterface)
+~ToolRegInterface()
-loadToolRegData(in regcert : PCCERT_CONTEXT)
+getToolRegID() : string &
+getToolRegDDL() : string &
+isInitialized() : bool
+dismiss()
+initialize()
ToolCertInterface
+ToolCertInterface()
+ToolCertInterface(inout Parametro1 : const ToolCertInterface)
+~ToolCertInterface()
-loadToolCertData(in toolcert : PCCERT_CONTEXT)
+storeNewCertificate(in toolBase64 : string, in pass : string) : bool
+dropToolCert()
-compareCert(in toolcert : PCCERT_CONTEXT) : bool
+getToolID() : string &
-decodePK12TB64(in toolBase64 : string, in pass : string) : PCCERT_CONTEXT
1
1
1
1
1
1
ProtectionProcessor
1

This module has the task to coordinate access to certificate data, meanwhile providing trustness, verification and management of those structures. Methods for this module are described below.

initialize – This method provide loading data to the proper certificate interface. Meanwhile certificate verification, management and validation checks are performed.

dismiss – This method delete all information stored in calling interface, resetting to a starting state. This operation is useful to prevent tampering user to read and maybe change memory values for certificates data.

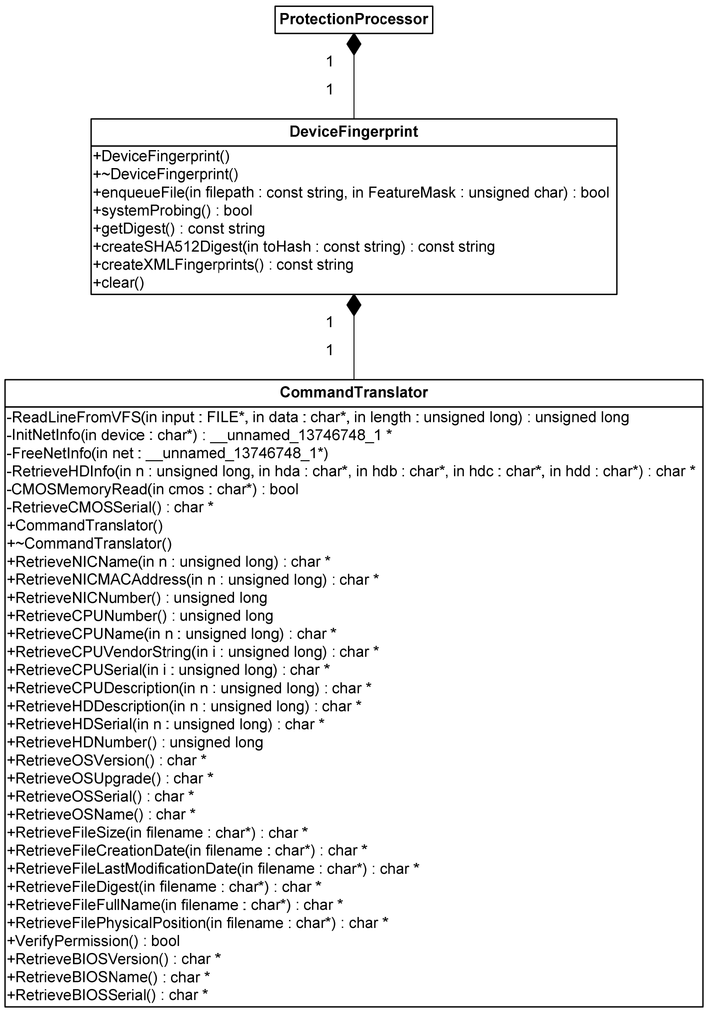
storeNewCertificate – is used in certify operation when the tool receives its certificate. This method automatically store certificate instance in the proper system repository for future use.

dropToolCertificate – is used in a recertify operation when the tool receives a new certificate. The old and invalid certificate is dropped.

getUID getToolID getToolRegID… – those methods provide access to data loaded from the certificates. If the certificate is not initialized the content is an empty string.

verifySysCert – Implements a validity check for certificate and certificate validation chain. If the verification fails (due to expiration date reaching or for un-trusted certification chain or error in some certificate signature) the certificate interface cannot be initialized. Used as private method for the specific certificate interfaces.

8.2.7 Fingerprint Module


ProtectionProcessor
1
1
DeviceFingerprint
+DeviceFingerprint()
+~DeviceFingerprint()
+enqueueFile(in filepath : const string, in FeatureMask : unsigned char) : bool
+systemProbing() : bool
+getDigest() : const string
+createSHA512Digest(in toHash : const string) : const string
+createXMLFingerprints() : const string
+clear()
1
1
CommandTranslator
-ReadLineFromVFS(in input : FILE*, in data : char*, in length : unsigned long) : unsigned long
-InitNetInfo(in device : char*) : __unnamed_13746748_1 *
-FreeNetInfo(in net : __unnamed_13746748_1*)
-RetrieveHDInfo(in n : unsigned long, in hda : char*, in hdb : char*, in hdc : char*, in hdd : char*) : char *
-CMOSMemoryRead(in cmos : char*) : bool
-RetrieveCMOSSerial() : char *
+CommandTranslator()
+~CommandTranslator()
+RetrieveNICName(in n : unsigned long) : char *
+RetrieveNICMACAddress(in n : unsigned long) : char *
+RetrieveNICNumber() : unsigned long
+RetrieveCPUNumber() : unsigned long
+RetrieveCPUName(in n : unsigned long) : char *
+RetrieveCPUVendorString(in i : unsigned long) : char *
+RetrieveCPUSerial(in i : unsigned long) : char *
+RetrieveCPUDescription(in n : unsigned long) : char *
+RetrieveHDDescription(in n : unsigned long) : char *
+RetrieveHDSerial(in n : unsigned long) : char *
+RetrieveHDNumber() : unsigned long
+RetrieveOSVersion() : char *
+RetrieveOSUpgrade() : char *
+RetrieveOSSerial() : char *
+RetrieveOSName() : char *
+RetrieveFileSize(in filename : char*) : char *
+RetrieveFileCreationDate(in filename : char*) : char *
+RetrieveFileLastModificationDate(in filename : char*) : char *
+RetrieveFileDigest(in filename : char*) : char *
+RetrieveFileFullName(in filename : char*) : char *
+RetrieveFilePhysicalPosition(in filename : char*) : char *
+VerifyPermission() : bool
+RetrieveBIOSVersion() : char *
+RetrieveBIOSName() : char *
+RetrieveBIOSSerial() : char *

This module is in charge for tool fingerprint extraction and fingerprint hash calculation. The architecture of the module is simple, a class DeviceFingerprint, provide coordination and interface to Protection Processor. Instead, in CommandTranslator, methods to access system information are implemented.
systemProbing – This method recover system information and installation information from the local machine using CommandTranslator functionalities. A call to this method is requested whenever a digest or fingerprint creation operations are needed.

getDigest – Returns SHA1 digest of the ToolFingerprint retrived by means of systemProbing. The hash is calculated on the chained fingerprint text.with no added xml tags.

getXMLFingerprints – Returns XML Representation of ToolFingerprints stored in a std::string field. A call to systemProbing is needed before this operation.

createSHA512Digest – this method implements SHA512 interface. It returns a 64 byte string that contains the argument hash.

clear – Clear the device class deleting all stored contents. Needed to prevent fingerprint screening by malicious user.

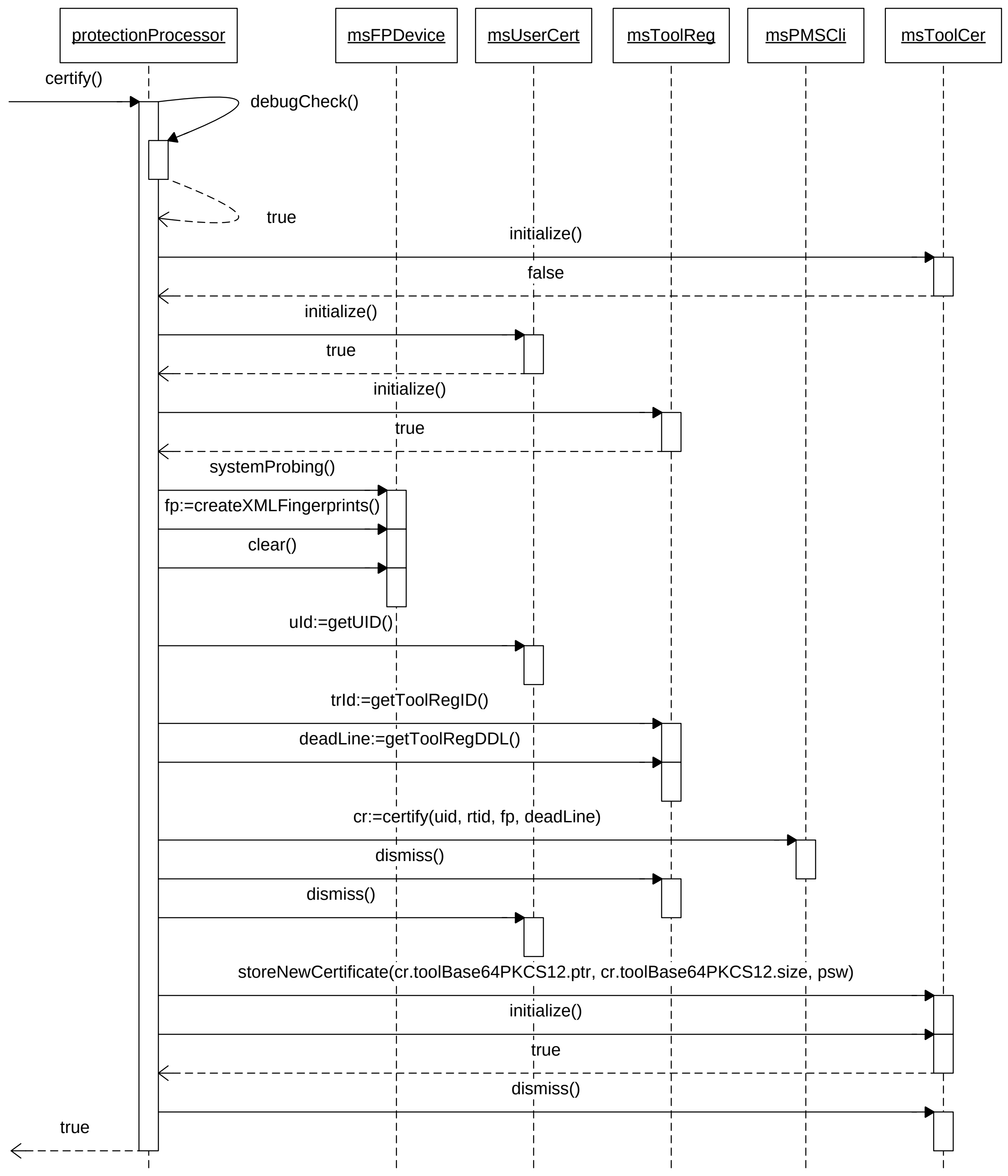
8.2.8 Tool certification

During tool certification, the protection processor checks:

- If some process is debugging the application. In that case operations ends.
- Certificate status. For each user and tool registration certificates ProtectionProcessor checks existence, validity and authenticity. If checks fails operations ends
- Existence of a tool certificate. If this certificate exists this operation ends because only a certification for each tool is permitted. To restore disabled tool and recover a new tool certificate the operation recertify has to be exploited.

ProtectionProcessor call tool fingerprint module for fingerprint extraction and send obtained XML fingerprint representation, along with tool registration id, deadline and user id to PMS client calling certify method. PMS Client have to send those information to AXCV to evaluate certification request.
If certification succeeds a certificate for the tool containing tool id (TID), enabling code and the public key is issued, along with the private key for the tool, in PKCS#12 format.
ProtectionProcessor store the new certificate through the ToolCertInterface module, completing certify operation.
All sensible data is cleaned from memory, as soon as it is no more useful for method workflow.

The next sequence diagram depict a complete certification scenario.

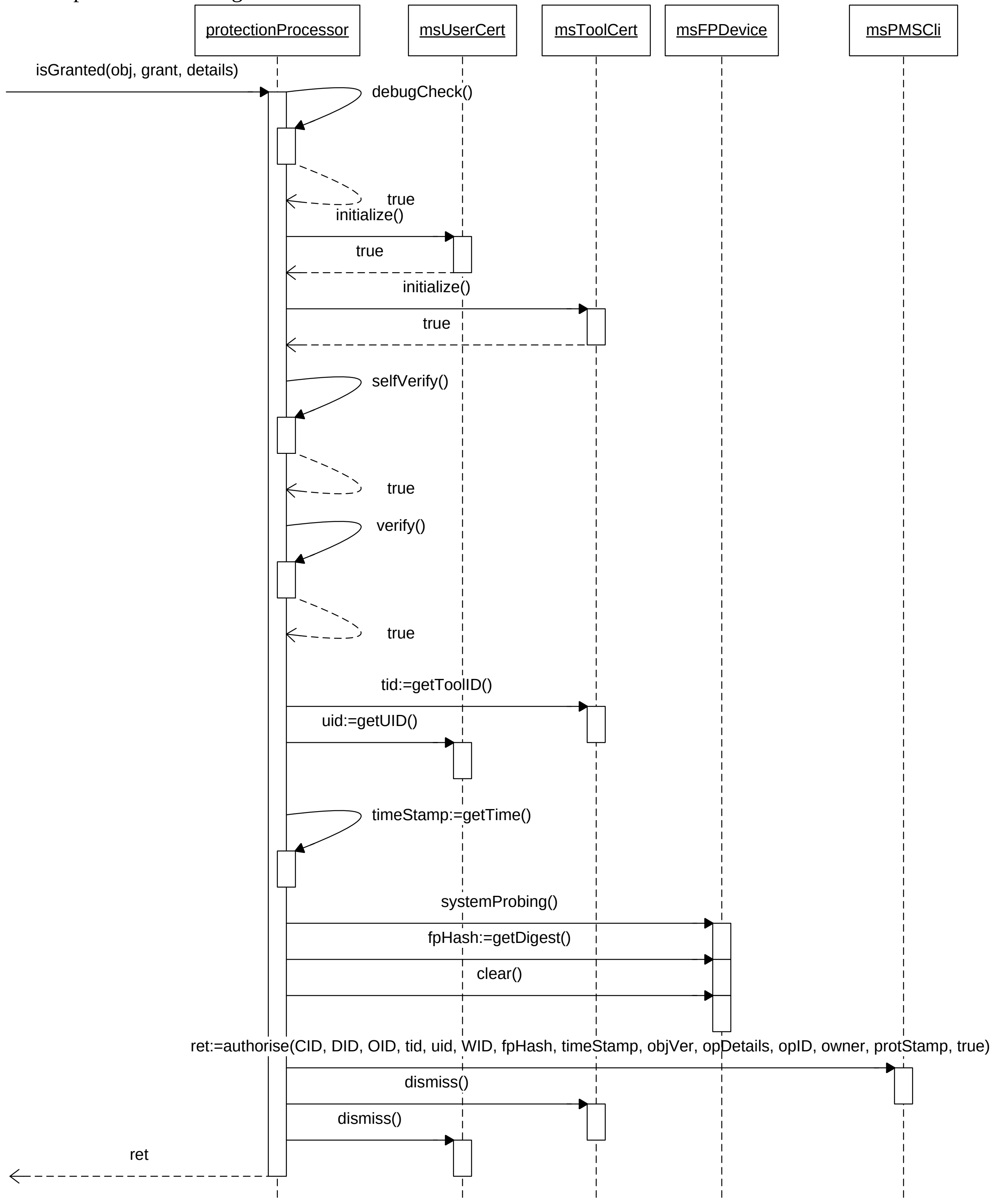


The figure above shows the case of successful certification. This can fail in several points. If this happens the method ends in several ways (summarized below) and it returns *false*. The possible failure points are:

- *debugCheck()* return *false* – someone is debugging the tool. The latter is disabled by calling disableTool() method;
- first *initialize()* called on *msToolCert* return *true* – this means the tool has been already certified;
- *initialize()* called on *msUserCert* or *msToolReg* return *false* – the tool misses user or tool registration certificate. The already initialized certificate data are deleted calling *dismiss()*;
- *cr.certificationResult* is *false* – the certification failed;
- last *initialize()* called on *msToolCert* return *false* – this means the tool certificate has not been correctly stored;

8.2.9 Grant Authorisation Requests

The figures below depict the interaction among protection processor and PMS client during content consumption and handling.



The figure above shows interactions among objects involved in an authorization request. In particular, Protection Processor makes some controls in order to ensure the trustiness of the tool and, if these controls succeed, invoke the *authorise* method of the PMS Client. In the above sequence diagram all the controls succeed however, the are several point of failure. In all the following cases, the tool is disabled and the method returns *false*:

- *debugCheck()* returns *false* – someone is debugging the tool;

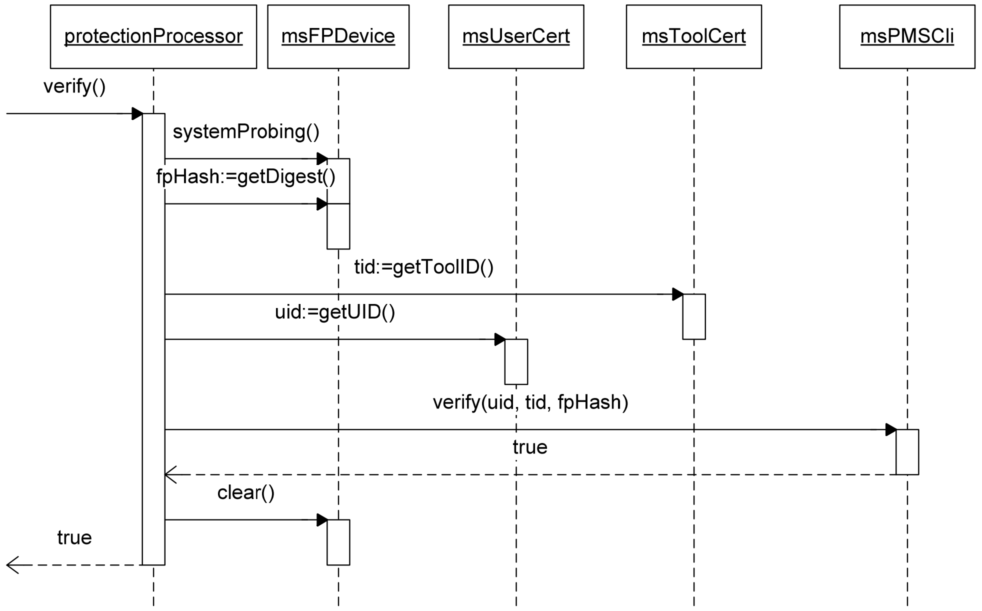
- *initialize()* called on *msUserCert* or *msToolCert* return *false* – the tool misses user or tool certificate. The already initialized certificate data are deleted calling *dismiss()*;
- *selfVerify* returns *false* – the activation code has not been verified;
- *verify* returns *false* – the verification against data on the AXCS fails;

Once the tool has been disabled, recovering is allowed through recertify method.
As stated above and depicted in the figures, Protection Processor uses PMS Client to communicate the verification information to the AXCS. That data consist of:

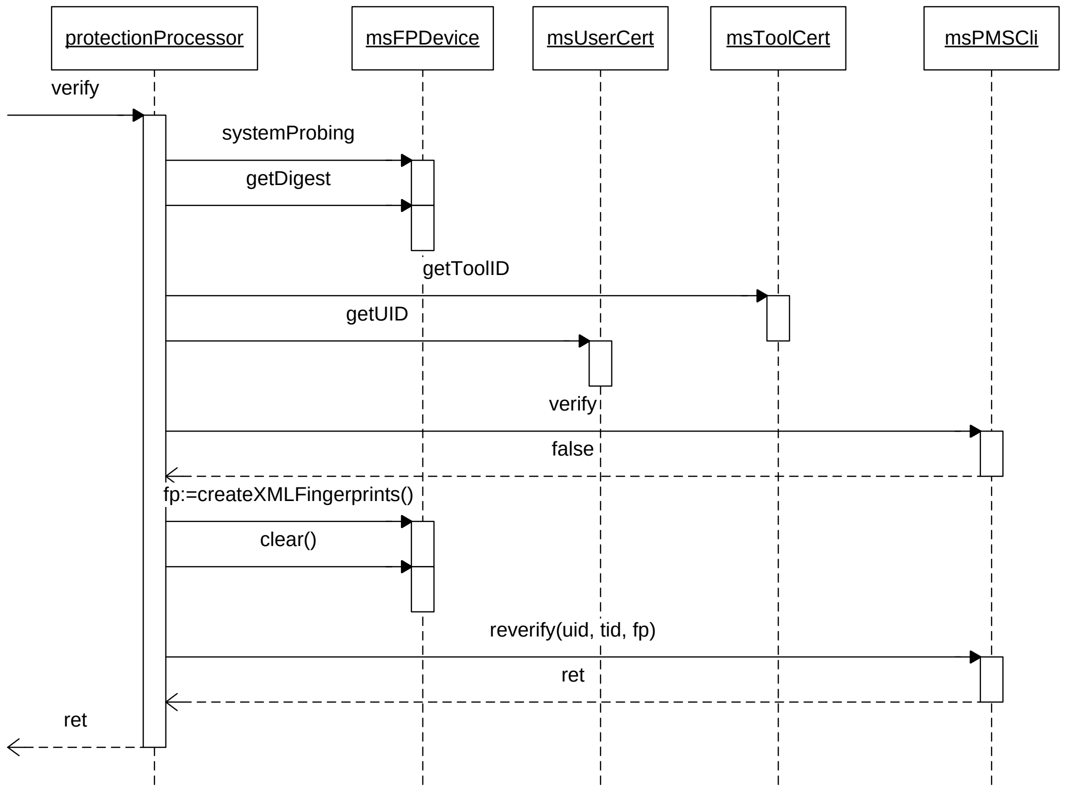
- TID – the tool identifier assigned to the tool at certification time by the AXCS
- Tool fingerprint hash – to make the communication lighter, the whole tool fingerprint is transmitted only if its hash do not match the one stored on the AXCS. This operation is performed through reverify method.

Communication between Protection Processor and AXCS works on a secure channel provided by the PMS Client. It has to be pointed out that even if communication is established by the PMS client certificates for communication are managed by the protection processor (as all the other security information) thus PMS client has to request the communication certificates to the protection processor every time it has to open a secure channel.
Disabling a tool include removal of all sensible information about tool/user from the device. All data stored in the secure of PMS Client cache has to be removed in this context. In the sequence diagrams below some key operation in certifying and verifying are explained and analyzed.

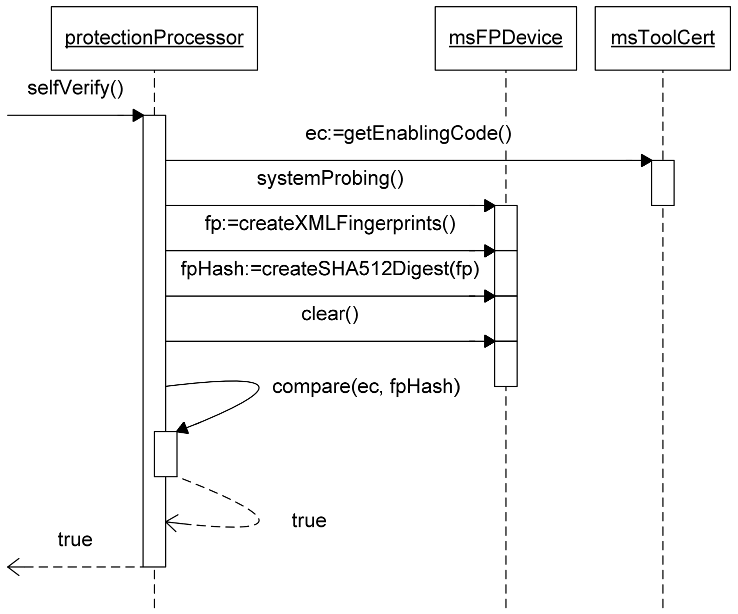
Next diagrams deeply explain tool verification in order to point out case where tool verification fails. In this case, the tool is disabled since it means that the user has performed something which could be considered as a tampering action. Reverify method is used in all the cases where fingerprint digest does not match the server side counterpart. This situation occurs whenever some change on the device has been made. However, minor changes are allowed by the system that can use recertify inputs to deeply analyze tool fingerprint and decide to block or not the tool.



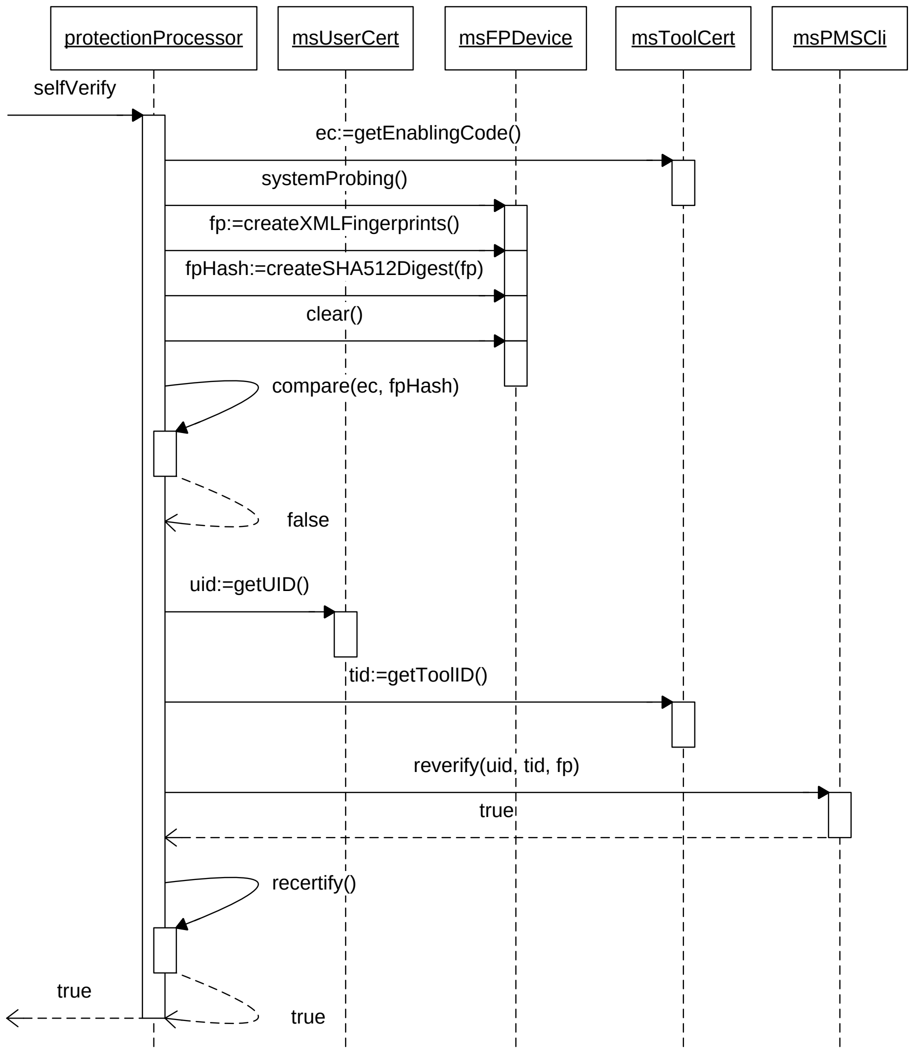
Please notice that the return value of *reverify* method is also the return value of the *verify* method. In fact, in the case *reverify* returns true, the tool can be considered verified. On the other hand, if the former does not, the tool will be considered tampered and therefore disabled.



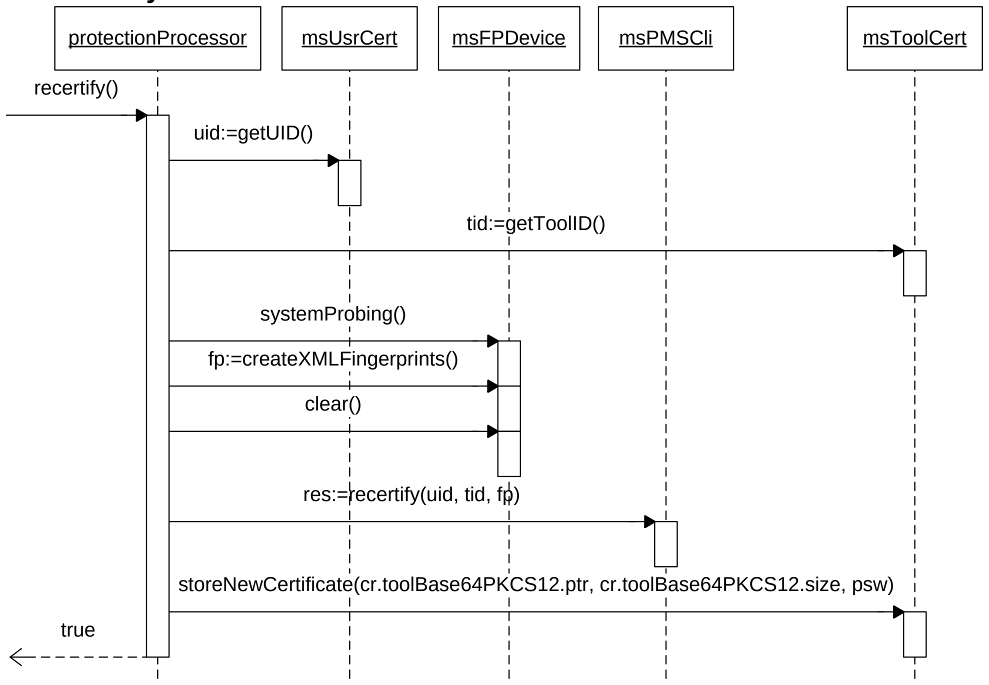
Next diagrams explain self verification and actions taken to manage detection of changing in the fingerprints of the module.



In an offline environment there are no chances to further verifications and the tool is automatically disabled. Instead, in an online scenario, some actions are taken to recover the situation through calling reverify. If the checks give positive result a new tool certificate along with a new enabling code is issued restoring a consistent status of the tool. Otherwise, the tool is disabled and it has to be explicitly recertified.

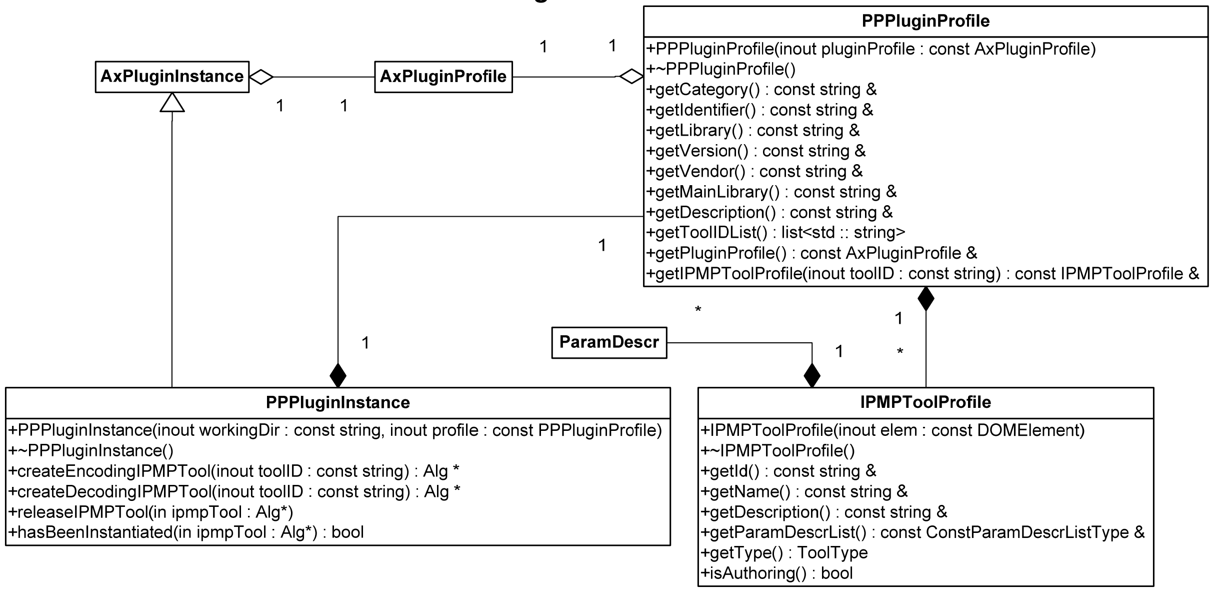

protectionProcessor
msUserCert
msFPDevice
msToolCert
msPMSCli
selfVerify
ec:=getEnablingCode()
systemProbing()
fp:=createXMLFingerprints()
fpHash:=createSHA512Digest(fp)
clear()
compare(ec, fpHash)
false
uid:=getUID()
tid:=getToolID()
reverify(uid, tid, fp)
true
recertify()
true
true

8.2.10 Tool recovery/recertification


protectionProcessor
msUsrCert
msFPDevice
msPMSCli
msToolCert
recertify()
uid:=getUID()
tid:=getToolID()
systemProbing()
fp:=createXMLFingerprints()
clear()
res:=recertify(uid, tid, fp)
storeNewCertificate(cr.toolBase64PKCS12.ptr, cr.toolBase64PKCS12.size, psw)
true

In order to recover a disabled tool recertify operation is provided. Protection Processor request a verification of tool fingerprint at AXCV through PMS Client. If the operation succeeds a new tool certificate and a new enabling code for the tool are issued by AXCV.

8.2.11 Protection Tools as AXMEDIS Plug-ins


PPPluginProfile
+PPPluginProfile(inout pluginProfile : const AxPluginProfile)
+~PPPluginProfile()
+getCategory() : const string &
+getIdentifier() : const string &
+getLibrary() : const string &
+getVersion() : const string &
+getVendor() : const string &
+getMainLibrary() : const string &
+getDescription() : const string &
+getToolIDList() : list<std :: string>
+getPluginProfile() : const AxPluginProfile &
+getIPMPToolProfile(inout toolID : const string) : const IPMPToolProfile &
AxPluginInstance
AxPluginProfile
ParamDescr
PPPluginInstance
+PPPluginInstance(inout workingDir : const string, inout profile : const PPPluginProfile)
+~PPPluginInstance()
+createEncodingIPMPTool(inout toolID : const string) : Alg *
+createDecodingIPMPTool(inout toolID : const string) : Alg *
+releaseIPMPTool(in ipmpTool : Alg*)
+hasBeenInstantiated(in ipmpTool : Alg*) : bool
IPMPToolProfile
+IPMPToolProfile(inout elem : const DOMElement)
+~IPMPToolProfile()
+getId() : const string &
+getName() : const string &
+getDescription() : const string &
+getParamDescrList() : const ConstParamDescrListType &
+getType() : ToolType
+isAuthoring() : bool
1
1
1
1
1
1
*
1
1
*

Protection Tools used by the Protection Processor are distributed as AXMEDIS Plug-ins (see section “Module - AXMEDIS Editor Plug-in Manager” in DE3.1.2.2.4). Each plug-ins contains a set of Protection Tools. Moreover, the manifest of the plug-in contains a specific description which reports the identifiers of the provided tools and their main features.
In particular, an AXMEDIS Plug-in containing some protection tools has to export the following functions:

```
extern "C" Alg* createIPMPTool(const std::string& toolID, bool encoding)
extern "C" void releaseIPMPTool(Alg* tool)
```

`createIPMPTool` allows creating Protection Tool instances by providing the tool identifier (`toolID`). When the `encoding` parameter is `true`, the function has to return the encoding version of the tool itself. Otherwise, the function has to return the decoding part of the required tool. That is, a tool identifier identifies the couple of encoding and decoding algorithms.
`realeaseIPMPTool` releases the given `tool` instance. This function has been introduced for the tool instances are created by the dynamic library and they have to be deleted by it.
The above figure shows the main classes involved in the management of protection plug-ins (i.e. AXMEDIS Plug-ins containing Protection Tools). In particular:

- `PPPluginInstance` is an extension of `AxPluginInstance`. An instance of this class represents a loaded protection plug-in. This class allows exploiting specific functions of this kind of plug-ins (e.g. create and release Protection Tools).
- `PPPluginProfile` 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 identifiers of contained Protection Tools and to get the description of any of them (given the related identifier).
- `IPMPToolProfile` allows accessing the data contained in the description of a Protection Tools (see section 20).

Please refer to "Module - AXMEDIS Editor Plug-in Manager" in DE3.1.2.2.4 for more details about `AxPluginInstance` and `AxPluginProfile`.
During `ProtectionProcessor::initializeTools` execution, Protection Processor gets all the plug-in profiles belonging to "IPMPTool" category. For each profile, it creates a new `PPPluginProfile` instance which wraps the generic profile and parses the specific descriptor contained in the latter.
When a Protection Plug-in is needed, a new instance of PPPluginInstance is allocated passing it the related profile. In this way, the plug-in instance is able to retrieve all the data it needs directly from the profile. Once created, the plug-in instance is registered to the Plug-in Manager.

8.3 Integration and compilation issues

Protection Processor relays on the PMS Client.
The following table summarizes the needed library in order to use the Protection Processor.

Name	OS/Platform		Library file	Description
	Windows/PC	Linux/PC		
OpenSSL	X	X	libeay32.lib ssleay32.lib	Provides several functions for certificate management and SSL connections.
Crypto++	X	X	cryptlib.lib	Provides functions for exploiting SHA-512
Common	X	X	common.lib	
Crypto API	X		crypt32.lib	
SNMP API	X		SnmpAPI.lib	

8.4 Configuration Parameters

Config parameter	Possible values
`PMSCLIENT – PMSClientEndpoint`	Any valid URL which points to the reference PMS Server
`PMSCLIENT – PMSClientLocalDSN`	Any valid DSN name for the secure cache database
`PMSCLIENT – PMSClientUser`	Any valid user name for the secure cache database
`PMSCLIENT – PMSClientPsw`	Any valid password for the secure cache database

8.5 Errors reported and that may occur

Error code	Description and rationales
0	`The protection processor has not been initialized`

1	Tools has to be initialized after protection processor initialization
2	Unable to unprotect an ipmp element which does not contain protection info and it even is not an AXMEDIS object
3	Unable to parse protection information which are not MPEG21 IPMP compliant
4	The given DIDL element has not pending protection information
5	IPMP Tool not available
6	A plugin with the given identifier has been already loaded
7	An IPMP tool with the given identifier has been already loaded
8	IPMP Tool profile not available
9	Key for retriving certificate not found

9 Encryption/Decryption Support (FUPF)

Module/Tool Profile		
Encryption/Decryption Support		
Responsible Name	Victor Rodriguez	
Responsible Partner	FUPF	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	Implemented	
Executable or Library/module (Support)	Static library	
Single Thread or Multithread	Multithread	
Language of Development	C and C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/encdecsup	
Reference to the AXFW location of the demonstrator executable tool for internal download	**Errore. Riferimento a collegamento ipertestuale non valido.** N/A	
Reference to the AXFW location of the demonstrator executable tool for public download	N/A	
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any	N/A	
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
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
Open SSL		

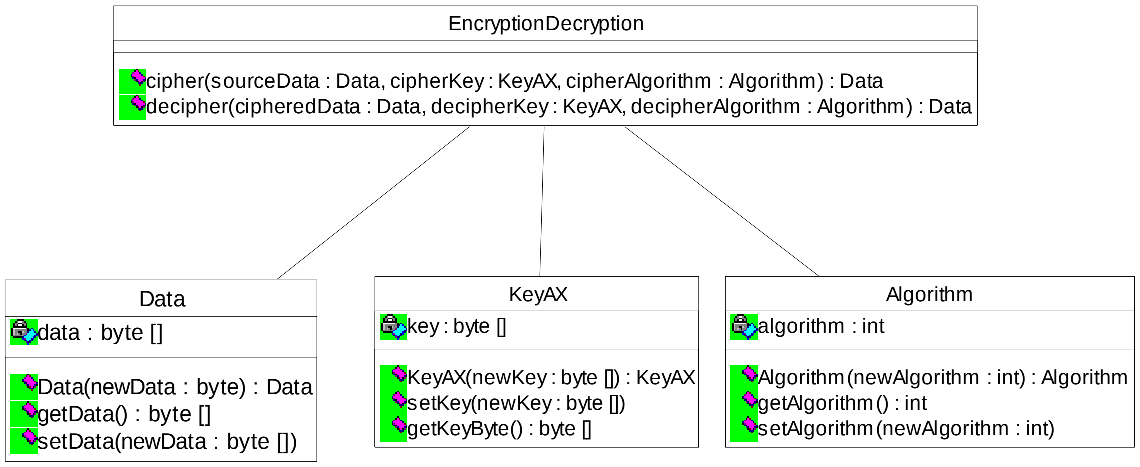
9.1 General Description of the Module

This module provides the needed functionality for encrypting / decrypting AXMEDIS.
The functionalities of this module will also be available as an AXMEDIS plugin for the Protection Processor. The form of the module in such a case will change from a static library (LIB) to a dynamic library (DLL), at least for the Windows version (as specified).
The functionality offered by this module is "function-oriented" in nature, rather than "object-oriented", therefore main functions are offered as static members. This will remain this way, as it was. However, while for an static library the model was perfectly valid, in the case of a dynamic library; it has to be stressed that the resulting DLL may not be multi-thread compliant.

9.2 Module Design in terms of Classes

9.2.1 Architecture for encryption / decryption support

Next figure shows the description of this module.
The algorithmic workload lies on OpenSSL, making thus this module as a simple adapter to the AXMEDIS requirements. The front-end class is EncryptionDecryption, which provides two evident methods: cipher and decipher.
Three classes are needed to deal with EncryptionDecryption, namely, *Data*, *KeyAX* and *Algorithm* whose use is self-explaining.


EncryptionDecryption
cipher(sourceData : Data, cipherKey : KeyAX, cipherAlgorithm : Algorithm) : Data
decipher(cipheredData : Data, decipherKey : KeyAX, decipherAlgorithm : Algorithm) : Data
Data
data : byte []
Data(newData : byte) : Data
getData() : byte []
setData(newData : byte [])
KeyAX
key : byte []
KeyAX(newKey : byte []) : KeyAX
setKey(newKey : byte [])
getKeyByte() : byte []
Algorithm
algorithm : int
Algorithm(newAlgorithm : int) : Algorithm
getAlgorithm() : int
setAlgorithm(newAlgorithm : int)

Class diagram for the Encryption / Decryption Support

The functionality of the classes inside the UML diagram is as follows:
EncryptionDecryption: Provides de basic functions for ciphering and deciphering of data, hiding to the calling application the complexity derived of the use of the OpenSSL library.
Data: Represents the data (either in clear or ciphered) used by the EncryptionDecryption class.
KeyAX: Represents the key for ciphering / deciphering the data.
Algorithm: Represents the algorithm for ciphering / deciphering data. The list of supported algorithms will be also implemented in this class by using constants in the corresponding programming language (C/C++).

9.3 Implementation of the algorithms

The implementation of the algorithms is carried out through the OpenSSL library. It defines itself as:
The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library.
OpenSSL license follows the Apache fashion, what allows its use freely both for commercial and non-commercial application. Axmedis has already approved its use, and it is a robust, reliable library. It has been ported into different platforms, and the ease of use and its completeness make from it the right choice.
The use of OpenSSL is transparent for the user of the EncryptionDecryption module: modules that use it does not need to know how EncryptionDecryption is internally implemented and which libraries it rely on. However, EncryptionDecryption is a static library, and as such, does not include itself the third parties code. Therefore, at linking time of an executable tool that requires EncryptionDecryption module, not only the encryptiondecryption.lib has to be linked, but also the corresponding OpenSSL libraries.

9.4 Examples of usage

It will be shown how to cipher a sample text.

```
Data cleardata(“this is a clear text”);   //holds the clear text to be ciphered
Data ciphered;                                //will hold the ciphered data
Algorithm *cipher = new Algorithm( axeds::AES_128_CBC_ALGORITHM);  //algorithm
axeds::byteType *cKey = new axeds::byteType[cipher->getSizeKey()]; //key
KeyAX clave(cKey,cipher->getSizeKey());                       //key
clave.setKey(Data("abracadabra").getData(),12);               //key is abracadabra

ciphered=EncryptionDecryption::encrypt(cleardata,clave,*cipher); //encrypt!

delete arr;
delete key;
delete cipher;
```

9.5 Formal description of Encryption / Decryption Support operations

EncryptionDecryption	
Method	cipher
Description	This method ciphers the sourceData passed as parameter using the KeyAX ciphering key and the algorithm indicated by the cipherAlgorithm. The returned information is the ciphered data. This method makes use of the corresponding operations inside the OpenSSL library for the different ciphering algorithms supported by it.
Input parameters	sourceData : Data, the data to be ciphered cipherKey : KeyAX, the key to be used to cipher data cipherAlgorithm : Algorithm, the algorithm used to cipher data
Output parameters	Data, the ciphered data
Method	decipher
Description	This method deciphers the cipheredData passed as parameter using the KeyAX deciphering key and the algorithm indicated by the decipherAlgorithm. The returned information is the data in clear. This method makes use of the corresponding operations inside the OpenSSL library for the different deciphering algorithms supported by it.
Input parameters	cipheredData : Data, the data to be deciphered decipherKey : KeyAX, the key for deciphering the data decipherAlgorithm : Algorithm, the algorithm for the deciphering the data
Output parameters	Data, the original data, in clear

Data	
Method	Data
Description	Constructor of the class which receives as parameter the data to be ciphered / deciphered.
Input parameters	NewData: byte[], an array of bytes containing the data either ciphered or in clear
Output parameters	A new instance of the Data class
Method	getData
Description	This method requests the data stored inside this class.
Input parameters	None
Output parameters	byte[], the byte array representing the data contained inside this class
Method	SetData
Description	This method allows setting new data inside this class.
Input parameters	byte[], the byte array representing the data contained inside this class
Output parameters	None

KeyAX	
Method	KeyAX
Description	Constructor of the class which receives as parameter the key for ciphering / deciphering data.
Input parameters	NewKey: byte[], an array of bytes containing the key
Output parameters	A new instance of the KeyAX class

Method	getKey
Description	This method requests the key stored inside this class.
Input parameters	None
Output parameters	byte[], the byte array representing the key contained inside this class
Method	setKey
Description	This method allows setting new key inside this class.
Input parameters	byte[], the byte array representing the key contained inside this class
Output parameters	None

Algorithm	
Method	Algorithm
Description	Constructor of the class which receives as parameter the algorithm identifier
Input parameters	NewAlgorithm:int, the identifier of the algorithm contained inside this class. It will depend on the values
Output parameters	A new instance of the Algorithm class
Method	getAlgorithm
Description	This method requests the algorithm stored inside this class.
Input parameters	None
Output parameters	int, the identifier of the algorithm contained inside this class. It will depend on the values defined by OpenSSL
Method	setAlgorithm
Description	This method allows setting new algorithm inside this class.
Input parameters	int, the identifier of the algorithm contained inside this class. It will depend on the values defined by OpenSSL
Output parameters	None

10 Compress/uncompress Support (DSI)

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

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

10.1 General Description of the Module

Compress and uncompress support is provided by the wxWidgets library. It provides input stream to access to ZIP files (wxZipInputStream) and input/output streams for gzip compression (wxZlibInputStream, wxZlibOutputStream).

wxZlibInputStream and wxZlibOutputStream are filtering streams they get/send information from/to other wxInput/OutputStream to uncompress/compress information.

10.2 Examples of usage

For example to compress a buffer of 1024 bytes to a file named "compressed.dat":

```
char data[1024];

// fill the data buffer

wxFileOutputStream ofile("compressed.dat");
wxZlibOutputStream compress(ofile);

compress.Write(data, 1024);
```

While to uncompress it:

```
wxFileInputStream ifile("compressed.dat");
wxZlibInputStream uncompress(ifile);
```

```
char data[1024];

uncompress.Read(data,1024);
if(uncompress.LastRead()!=1024)
      …
```

11 Scramble/Descramble Support (EPFL)

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

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

[DSI] Very nice description of Cryptlib library! I understand you would like to use Cryptlib in order to provide Scramble and Descramble Support but I would like to have also a description of your module. Your module should be integrated with Protection Processor using Protection Plug-in specification. Please, provide something more than the trivial description of a third-party library.

11.1 General Description of the Module

The word “scrambling” is often mis-used as a synonym of cryptography.
The term cryptography refers to the tools and mechanisms enabling:

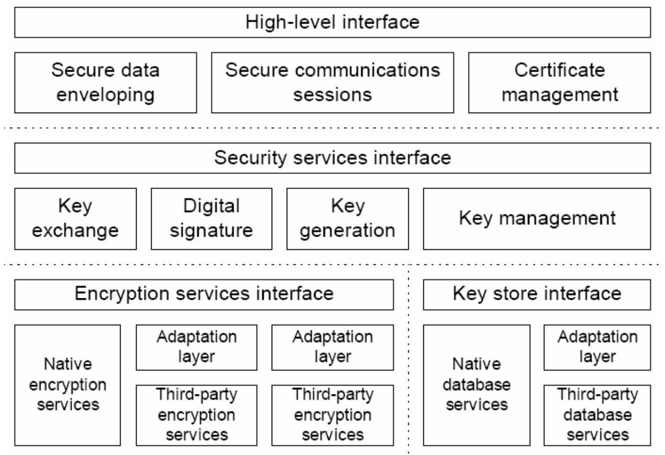
- Tamper detection allows the information receiver to verify that it has not been modified during transmission. If there were any attempt to modify or substitute data, a false message would be detected.
- Authentication allows the information receiver to determine who sent the message.
- Privacy/confidentiality ensures that no one can read the message except the intended receiver. Integrity assures the receiver that the message that they received was not modified in any way since it was sent from the origin.
- Non-repudiation is a mechanism that proves that the sender really sent the message.

Lastly, scrambling allows two communication parties to disguise information they send to each other. The sender encrypts/scramble the information before sending it. The receiver decrypts/descramble the information after receiving it.
In cryptographic terminology, the message is called plaintext or cleartext. Encryption is encoding the contents of the message in such a way that hides its contents from outsiders. The encrypted message is called the ciphertext. The process of retrieving the plaintext from the ciphertext is called decryption. Encryption and decryption usually make use of a key, and the coding method is such that decryption can be performed only by knowing the proper key.

Scrambling / Descrambling algorithms are based on secret key algorithms.
In secret key cryptography, a single key is used for both encryption and decryption. The sender uses the key to encrypt the plaintext and then sends the ciphertext to the receiver. The receiver applies the same key to decrypt the message and recover the plaintext. There are several widely used secret key cryptography schemes [Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB) and Counter (CTR) modes] and they are generally categorized as being either block ciphers or stream ciphers. A block cipher is so-called because it encrypts blocks of data at a time. The same plaintext block will always be encrypted into the same ciphertext when using the same key. Stream ciphers operate on a single bit, byte, or word at a time, and implements a feedback mechanism so that the same plaintext will yield different ciphertext every time it is encrypted.
Usually, scrambling / descrambling algorithms refer to stream ciphers algorithms.
In the past, the scrambling process did not change the information with the content but it only "mix" it. In this way the "preview" of the content could be done with low cost device.
Scrambling algorithms used to allow "content preview" functionality are strongly linked to proprietary and "unknown" solutions and implementation. This is what in cryptography is called "security by obscurity" that it is not an "open" approach.
Due to the evolution of the attacks and the openness of the approach, AXMEDIS should use more sophisticated scrambling algorithms that use strong known secret key algorithms. A well-know library called cryptlib provides many functions, including all what may be necessary in the project.

11.2 Module Design in terms of Classes

The cryptlib library consists of a set of layered security services and associated programming interfaces that provide an integrated set of information and communications security capabilities. Much like the network reference model, cryptlib contains a series of layers that provide each level of abstraction, with higher layers building on the capabilities provided by the lower layers.
At the lowest level are basic components such as core encryption and authentication routines, which are usually implemented in software but may also be implemented in hardware (due to the speed of the software components used in cryptlib, the software is usually faster than dedicated hardware).
Scrambling can be supported by mixing the several available symmetric encryption mechanisms provided at the lowest layer of the security stack.



The application programming interface (API) serves as an interface to a range of plug-in encryption modules that allow encryption algorithms to be added in a fairly transparent manner, so that adding a new algorithm or replacing an existing software implementation with custom encryption hardware can be done without any trouble.
The standardised API allows any of the algorithms and modes supported by cryptlib to be used with a minimum of coding effort. In addition the easy-to-use high-level routines allow for the exchange of encrypted or signed messages or the establishment of secure communications channels with a minimum of programming overhead.
Language bindings are available for C / C++, C# / .NET, Delphi, Java, Python, Tcl, and Visual Basic (VB).

cryptlib has been written to be as foolproof as possible. On initialisation it performs extensive self-testing against test data from encryption standards documents, and the APIs check each parameter and function call for errors before any actions are performed, with error reporting down to the level of individual parameters. In addition logical errors such as, for example, a key exchange function being called in the wrong sequence, are checked for and identified.

11.3 Technical and Installation information

All necessary constants, types, structures, and function prototypes are defined in a language-specific header file as described below. You need to use these files for each module that makes use of cryptlib. Although many of the examples given in this manual are for C/C++ (the more widely-used ones are given for other languages as
well), they apply equally for the other languages. All language bindings for cryptlib are provided in the bindings subdirectory. Before you can use a specific language interface, you may need to copy the file(s) for the language that you're using into the cryptlib main directory or the directory containing the application that you're building. Alternatively, you can refer to the file(s) in the bindings directory by the absolute pathname.

11.3.1 Initialisation

Before you can use any of the cryptlib functions, you need to call the **cryptInit** function to initialise cryptlib. You also need to call its companion function **cryptEnd** at the end of your program after you've finished using cryptlib. **cryptInit** initializes cryptlib for use, and **cryptEnd** performs various cleanup functions including automatic garbage collection of any objects you may have forgotten to destroy. You don't have to worry about inadvertently calling **cryptInit** multiple times (for example if you're calling it from multiple threads), it will handle the initialisation correctly.
However you should only call **cryptEnd** once when you've finished using cryptlib. If you call **cryptEnd** and there are still objects in existence, it will return CRYPT_- ERROR_INCOMPLETE to inform you that there were leftover objects present.
cryptlib can tell this because it keeps track of each object so that it can erase any sensitive data that may be present in the object (**cryptEnd** will return a CRYPT_- ERROR_INCOMPLETE error to warn you, but will nevertheless clean up and free each object for you). To make the use of **cryptEnd** in a C or C++ program easier, you may want to use the C `atexit()` function or add a call to **cryptEnd** to a C++ destructor in order to have **cryptEnd** called automatically when your program exits. If you're going to be doing something that needs encryption keys (which is pretty much everything), you should also perform a randomness poll fairly early on to give cryptlib enough random data to create keys:

```
cryptAddRandom( NULL, CRYPT_RANDOM_SLOWPOLL );
```

Randomness polls are described in more detail in "Random Numbers" on page 279. The randomness poll executes asynchronously, so it won't stall the rest of your code while it's running. The one possible exception to this polling on startup is when you're using cryptlib as part of a larger application where you're not certain that cryptlib will actually be used. For example a PHP script that's run repeatedly from the command line may only use the encryption functionality on rare occasions (or not at all), so that it's better to perform the slow poll only when it's actually needed rather than unconditionally every time the script is invoked. This is a somewhat special case though, and normally it's better practice to always perform the slow poll on startup.
As the text above mentioned, you should initialize cryptlib when your program first starts and shut it down when your program is about to exit, rather than repeatedly starting cryptlib up and shutting it down again each time you use it. Since cryptlib consists of a complete crypto operating system with extensive initialisation, internal
security self-tests, and full resource management, repeatedly starting and stopping it will unnecessarily consume resources such as processor time during each initialisation and shutdown. It can also tie up host operating system resources if the host contains subsystems that leak memory or handles (under Windows, ODBC and LDAP are particularly bad, with ODBC leaking memory and LDAP leaking handles. DNS is also rather leaky — this is one of the reasons why programs like web browsers and FTP clients consume memory and handles without bounds). To avoid
this problem, you should avoid repeatedly starting up and shutting down cryptlib:

Right

```
cryptInit();
serverLoop:
process data;
cryptEnd();
```

Wrong

```
serverLoop:
cryptInit();
process data;
cryptEnd();
```

C / C++

To use cryptlib from your C or C++ application you would use:

```
#include "cryptlib.h"

cryptInit();

/* Calls to cryptlib routines */

cryptEnd()
```

11.4 Integration and compilation issues

Cryptlib is re-entrant and completely thread-safe, allowing it to be used with multithreaded applications on systems that support threads. Because it is thread-safe, lengthy cryptlib operations can be run in the background if required while other processing is performed in the foreground. In addition cryptlib itself is multithreaded so that computationally intensive internal operations take place in the background without impacting the performance of the calling application.

12 MPEG-21 DIBO (EPFL)

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

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

12.1 MPEG-21 DIP in AXMEDIS

In the MPEG-21 Digital Item Processing standard, Digital Item Base Operations (DIBOs) are defined as the functional building blocks utilized by a Digital Item Method (DIM). They can be considered somewhat analogous to the standard library of functions of a programming language. Their implementation as a software library of functions can be then derived quite straightforwardly from the MPEG-21 specification. In fact, a DIBO is described by a normatively defined interface; and normatively defined semantics. The module architecture shall then be structured as software implementation of the normative interfaces and shall be compliant with the normatively defined semantics.
AXMEDIS will support only those Basic Operations needed for the implementation of interactivity at the level of content consumption. Examples of DIBOs that may be supported:

```
Play(element, async)
```

This DIBO causes the DIDL element represented by the `element` parameter to be rendered into a transient and directly perceivable representation. The `element` parameter shall be a DOM `Element` object representing a COMPONENT or DESCRIPTOR to be played. It is an error to invoke this DIBO if the `element` parameter is not a DOM `Element` object representing a COMPONENT or DESCRIPTOR, in which case an invalid parameter exception is generated.
The manner of playing the element, appropriate to its content, is left as an implementation choice of the DIBO implementer.

```
Release(PlayStatus)
```

This DIBO causes playing of the DIDL element associated with the specified `playStatus` to be stopped and any state information to be released. The `playStatus` parameter shall be an object that was returned by a call to the `play` DIBO to play the associated element which is to be stopped.

```
Adapt(element, metadata);
```

This DIBO allows a DIM author to explicitly request an adaptation of the DIDL element represented by the `element` parameter.
The `metadata` parameter is an array of `Element` objects representing additional information the DIM author suggests can be considered when adapting the resource. The `metadata` parameter may be null if the DIM author does not wish to provide any such suggestions. If the metadata parameter is not null and is not an array of DOM `Element` objects then an invalid parameter exception is generated.
If an adaptation of the element does take place and is successful, then an `Element` object representing the DIDL element for the adapted element is returned. This can then be utilized as a parameter to other appropriate DIBOs. The original element remains unchanged.
The MPEG-21 DIA specification provides the following tools related to resource adaptation which an implementer of this DIBO can consider supporting.

- Usage Environment Descriptions;
- Bitstream Syntax Description based adaptation;
- Terminal and network quality of service;
- Usage Constraints Descriptions; and
- DIA Configuration.

If the available DIBOs standardized by MPEG, should not support the entire range of AXMEDIS use cases, an extension to the provided functionality can be implemented by means of User defined Opraations called DIXOs (Digital Item eXtensions Operation).

12.2 General Description of the Module

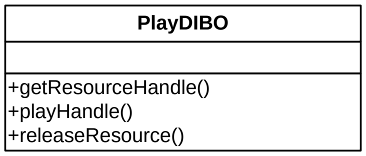
This module is in charge of implementing the MPEG-21 Digital Item Base Operations (DIBOs) relevant for the purposes of AXMEDIS. It will be implemented as a library of processing functions to be called upon requests coming from the Digital Item Method Engine (see following section).
Once the AXMEDIS Object Manager retrieves MPEG-21 DIP information in a managed AX Object it dispatches the DIP excerpt to the DI Method Engine (DIME). This component is in charge of executing the received DIP script either by calling the appropriate DIBOs implementation or, if present, by executing the ECMA script embedded in the DIP description.

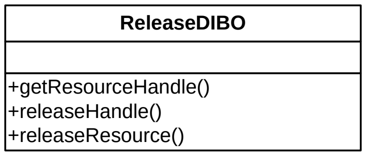
12.3 Classes

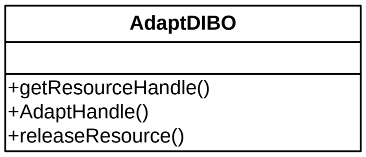
An implementation of the DIBO in terms of C++ classes shall include methods for

- getting an handle to the concerned resource to be processed (`getHandle(…)`) ;
- actually implement the required operation on the resource ((e.g. `playDIBO(…)`);
- send back the control on the provided handle to the DIM engine (`releaseHandle(…)`).

A formalization of the concerned classes is provided below:


PlayDIBO
+getResourceHandle()
+playHandle()
+releaseResource()


ReleaseDIBO
+getResourceHandle()
+releaseHandle()
+releaseResource()


AdaptDIBO
+getResourceHandle()
+AdaptHandle()
+releaseResource()

13 MPEG-21 DIM (EPFL)

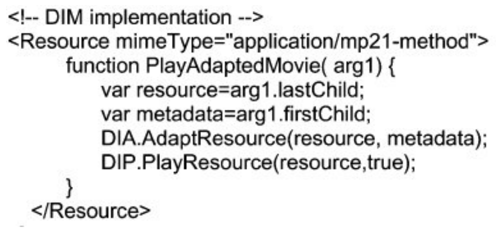
Module/Tool Profile		
Compress/uncompress Support		
Responsible Name		
Responsible Partner		
Status (proposed/approved)		
Implemented/not implemented		
Status of the implementation		
Executable or Library/module (Support)		
Single Thread or Multithread		
Language of Development		
Platforms supported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

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

13.1 MPEG-21 DI Methods in AXMEDIS

This module implements the execution of Digital Item Methods (DIMs) embedded into an AXMEDIS object. Starting from a Digital Item Declaration (DID) describing the structure of the DI it is possible to add interactivity to that DI using DIMs. A DIM expresses the interaction of an MPEG-21 User with a DI. It contains calls to DIBOs and describes the possible interactions of an MPEG-21 User (e.g., a human consumer) with the DI.
A DIM can be implemented either as a combination ECMAScript constructs (e.g. for and ++ operators) and DIBOs or as a pure sequence of DIBOs. This module shall parse the received DID, retrieve the DIM to be executed and call the appropriate DIBOs or ECMAScript interpreter in order to trigger the actual execution.

Examples of DIMs are shown below:



```
<!-- DIM implementation -->
<Resource mimeType="application/mp21-method">
        function PlayAdaptedMovie( arg1) {
             var resource=arg1.lastChild;
             var metadata=arg1.firstChild;
             DIA.AdaptResource(resource, metadata);
             DIP.PlayResource(resource,true);
        }
   </Resource>
```

This is an example of DIM using both ECMAScript and DIBOs

```
<!-- DIM implementation -->
<Resource mimeType="application/mp21-method"><![CDATA[
  function main(){
    var movies = ObjectMap.getObjects("urn:foo:Movie" );
    for( i = 0; i < movies.length; i++ ) {
       RunDIM("current", null, "PlayAdaptedMovie", movies[i]);
    }
  }   ]]>
  </Resource>
```

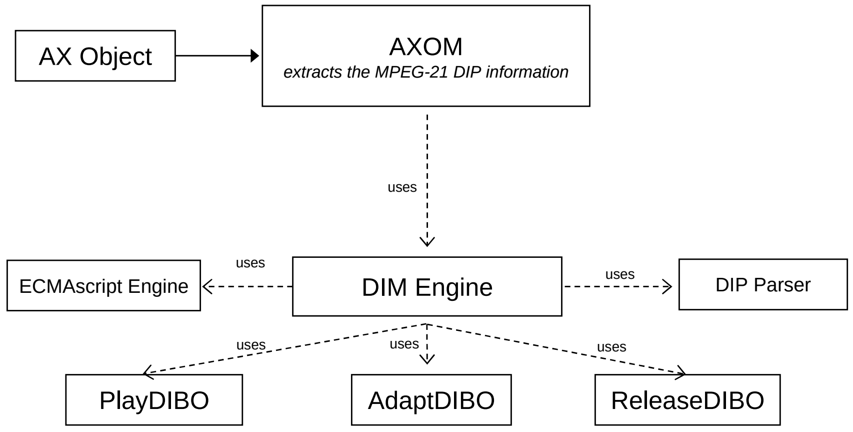
This is an example of DIM implemented uniquely as a chian of DIBOs.

13.1.1 Relationship between DIMs, DIBOs, and DIXOs

The DIMs call the DIBOs and the DIXOs to delegate processing to keep the DIM script simple. DIMs may choose to use DIBOs when they are available and DIXOs when the required functionality is not available as a DIBO.
The invocation mechanisms for DIBOs and the DIXOs from DIMs are different. The DIBOs have a mapping to DIML and this is used to call the DIBOs from within a DIM. The invocation mechanism for DIXOs is unique to the DIXO Language used to write the DIXOs. The same call for a given DIXO Language is used to invoke all the DIXOs in that DIXO Language from a DIM, where the name and the arguments of the DIXO are all in turn arguments to that invocation call.
DIXOs while implementing the extended processing may call normatively defined DIBOs and other DIXOs. The invocation mechanism of DIBOs and other DIXOs from any DIXO is direct using the bindings of DIBOs in that particular DIXO Language.

13.2 General description of the module

This module is in charge of receiving MPEG-21 DIP scripts from the AX Object Manager, to parse the received XML excerpt and coordinate the processing of the DIBOs implementation and the ECMAscript engine. A received MPEG-21 DIP script can be either a mixture of ECMA scripts and DIBOs calls or a simple chain of DIBOs. The DIM Module will be in charge of traducing the actions expressed MPEG-21 DIP descriptors in synchronous calls to the relevant libraries.



Once the AXMEDIS Object Manager retrieves MPEG-21 DIP information in a managed AX Object it dispatches the DIP excerpt to the DI Method Engine (DIME). This component is in charge of executing the received DIP script either by calling the appropriate DIBOs implementation (Play, Release, Adapt, etc.) or, if present, by executing the ECMA script embedded in the DIP description.

14 MPEG-21 DIA Processing(EPFL)

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

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

14.1 General Description of the Module

To achieve interoperable transparent access to (distributed) advanced multimedia content, the adaptation of Digital Items is required. This concept is illustrated in the figure below. As shown in this conceptual architecture, a Digital Item is subject to a resource adaptation engine, as well as a description adaptation engine, which together produce the adapted Digital Item.

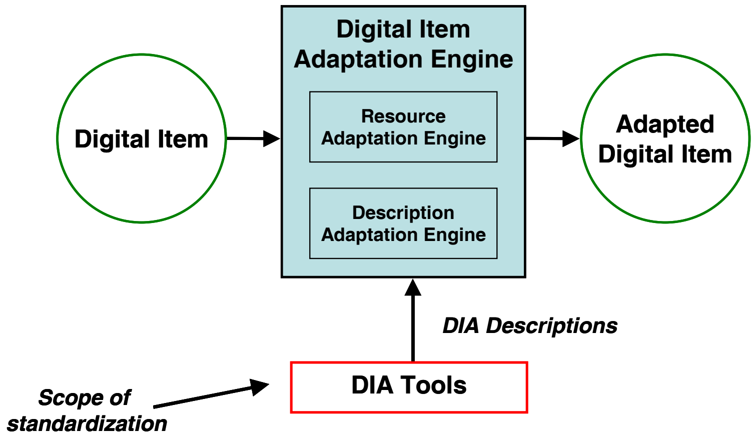

Digital Item
Digital Item Adaptation Engine
Resource Adaptation Engine
Description Adaptation Engine
Adapted Digital Item
DIA Descriptions
DIA Tools
Scope of standardization

Illustration of Digital Item Adaptation

The seventh part of ISO/IEC 21000 (MPEG-21) specifies descriptions and format-independent mechanisms to provide support for Digital Item Adaptation in terms of resource adaptation, description adaptation, and/or Quality of Service management and are collectively referred to as DIA Tools. It is important to emphasise that the adaptation engines themselves are non-normative tools of Digital Item Adaptation.

In the context of AXMEDIS, the MPEG-21 DIA Tools of interest are the Usage Environment Description Tools, which include **User characteristics**, **Terminal capabilities**, **Network characteristics** and **Natural environment characteristics**. These tools provide descriptive information about the various properties of the usage environment, which originate from Users, to accommodate, for example, the adaptation of Digital Items for transmission, storage and consumption.

1. **User characteristics:** These are tools for describing various characteristics of Users, including general User information, usage preferences and usage history, presentation preferences, accessibility characteristics, mobility characteristics and destination.
2. **Terminal capabilities:** The description of a terminal's capabilities is primarily required to satisfy consumption and processing constraints of a particular terminal. Terminal capabilities are defined by a wide variety of attributes. Among them are codec capabilities, which include encoding and decoding capabilities, device properties, which include power, storage and data I/O characteristics, and input-output characteristics, which include display and audio output capabilities.
3. **Network characteristics:** These tools specify network characteristics in terms of network capabilities and conditions, including available bandwidth, delay and error characteristics. These descriptions could be used for efficient and robust transmission of resources.
4. **Natural Environment Characteristics:** These tools are used to describe natural environment characteristics including location and time of usage of a Digital Item, as well as characteristics that pertain to audio-visual aspects. For the visual aspects, illumination characteristics that may affect the perceived display of visual information are specified. For the audio aspects, the description of the noise levels and a noise frequency spectrum are specified.

As an example, an instantiation of the codec capabilities of a terminal is given below. In this description instance, the terminal is capable of decoding MP3 and AMR audio formats, the JPEG image format, and the MPEG-4 video format (Simple Profile @ Level 1). It is also able to encode audio in an AMR format and encode video in the MPEG-4 format (Simple Profile @ Level 1).

```
<DIA>
      <Description xsi:type="UsageEnvironmentType">
            <UsageEnvironmentProperty xsi:type="TerminalsType">
                  <Terminal>
                        <TerminalCapability xsi:type="CodecCapabilitiesType">
                              <Decoding xsi:type="AudioCapabilitiesType">
                                    <Format

      href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:4.4">
                                          <mpeg7:Name xml:lang="en">MP3</mpeg7:Name>
                                    </Format>
                                    <Format
href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:6">
                                          <mpeg7:Name xml:lang="en">AMR</mpeg7:Name>
                                    </Format>
                              </Decoding>
                              <Decoding xsi:type="ImageCapabilitiesType">
                                    <Format
href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:4">
                                          <mpeg7:Name xml:lang="en">JPEG</mpeg7:Name>
                                    </Format>
                              </Decoding>
                              <Decoding xsi:type="VideoCapabilitiesType">
                                    <Format

      href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:3.1.2">
                                          <mpeg7:Name xml:lang="en">
```

```
                                                    MPEG-4 Visual Simple Profile @ Level
1
                                              </mpeg7:Name>
                                        </Format>
                                  </Decoding>
                                  <Encoding xsi:type="AudioCapabilitiesType">
                                        <Format
href="urn:mpeg:mpeg7:cs:AudioCodingFormatCS:2001:6">
                                              <mpeg7:Name xml:lang="en">AMR</mpeg7:Name>
                                        </Format>
                                  </Encoding>
                                  <Encoding xsi:type="VideoCapabilitiesType">
                                        <Format

      href="urn:mpeg:mpeg7:cs:VisualCodingFormatCS:2001:3.1.2">
                                              <mpeg7:Name xml:lang="en">
                                                    MPEG-4 Visual Simple Profile @ Level
1
                                              </mpeg7:Name>
                                        </Format>
                                  </Encoding>
                            </TerminalCapability>
                      </Terminal>
                </UsageEnvironmentProperty>
      </Description>
</DIA>
```

15 MPEG-21 DIA (EPFL)

MISSING

16 AXMEDIS Data Model (DSI)

It also includes a description of the AXMEDIS details that make of AXMEDIS of a specialization of MPEG21 format.

16.1 AXMEDIS Objects as MPEG21 Objects

An AXMEDIS Object has to be an MPEG21 digital item but any MPEG21 digital item is not an AXMEDIS Object. This means that an AXMEDIS Object will have a specific structure and will not support all the extremely flexible structuring capabilities of MPEG21 digital items.
In this section will be investigated how AXMEDIS Objects could be represented using the structuring features of MPEG21.

16.1.1 MPEG21 Digital Items

The following figure describes how a MPEG21 Digital Item is structured.

For a complete description see part 2 of the MPEG21 standard describing the Digital Item Description Language. This part of the standard is related to unprotected digital items only.

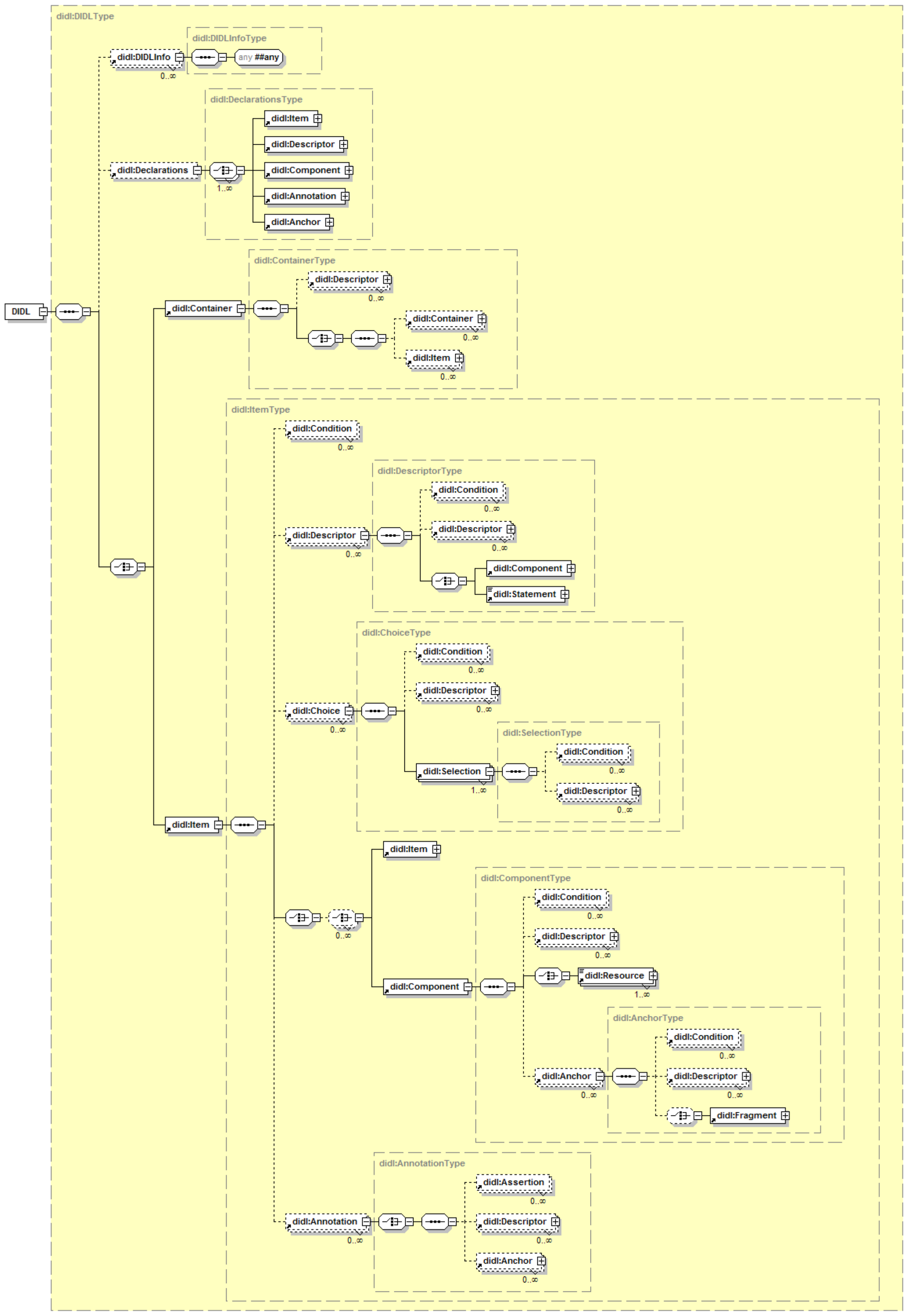
The elements contained in a MPEG21 digital items are:

- ***Container*** – is a container of items or of other containers;
- ***Item*** – represents a digital item, it contains *Descriptors* (metadata of the whole digital item) *Components* (content that builds up the item), and it also contains other secondary elements;
- ***Descriptor*** – contains metadata thought a *Statement* element or a *Component* (e.g. for thumbnails)
- ***Component*** – contains *Resources* and *Descriptors* (metadata of the resource);
- ***Resource*** – contains an external reference to the resource (audio, video, text,…) or it can host it inside the element using base64 encoding;
- ***Annotation*** – contains an annotation
- ***Anchor*** – is a link into the content
- ***Condition, Choice***, ***Selection*** – are used to group sub parts of the item on the basis of end user selections, this to avoid streaming of big items.

For references to other elements the ***xi:include*** elements can be used.

When considering **protected** MPEG21 digital items, the standardization process of this feature, is not currently at level of International Standard but at level of Committee Draft.

Protected content in MPEG21 is obtained by substituting a sub tree of the original XML tree with an element having the same name (but with different namespace) and containing the protected version of the sub tree in binary form and the additional information needed to enable access to the content.


didl:DIDLType
didl:DIDLInfoType
didl:DIDLInfo
0..∞
any ##any
didl:DeclarationsType
didl:Item
didl:Descriptor
didl:Declarations
didl:Component
1..∞
didl:Annotation
didl:Anchor
didl:ContainerType
didl:Descriptor
0..∞
DIDL
didl:Container
didl:Container
0..∞
didl:Item
0..∞
didl:ItemType
didl:Condition
0..∞
didl:DescriptorType
didl:Condition
0..∞
didl:Descriptor
didl:Descriptor
0..∞
0..∞
didl:Component
didl:Statement
didl:ChoiceType
didl:Condition
0..∞
didl:Descriptor
didl:Choice
0..∞
0..∞
didl:SelectionType
didl:Condition
0..∞
didl:Selection
1..∞
didl:Descriptor
0..∞
didl:Item
didl:Item
didl:ComponentType
didl:Condition
0..∞
0..∞
didl:Descriptor
0..∞
didl:Component
didl:Resource
1..∞
didl:AnchorType
didl:Condition
0..∞
didl:Anchor
didl:Descriptor
0..∞
0..∞
didl:Fragment
didl:AnnotationType
didl:Assertion
0..∞
didl:Annotation
didl:Descriptor
0..∞
0..∞
didl:Anchor
0..∞

16.1.2 AXMEDIS Objects

AXMEDIS Objects can be classified as:

Basic AXMEDIS Object: containing one or more digital resources (image, video, document, etc.) and metadata related to the whole object. Resources can be stored inside the object or outside.
Protected Basic AXMEDIS Object: containing one or more protected digital resources and metadata of the whole object (in clear but certified). Protected resources can be stored inside the object or outside.
Composite AXMEDIS Object: containing a set of AXMEDIS Objects (Basic or Composite, protected or not). It has specific metadata for the whole object in addition to metadata related to sub-objects. The sub-objects can be stored inside the object or referenced.
Protected Composite AXMEDIS Object: as the previous but the whole object is protected (the metadata of the whole object and of the sub-objects has to be accessible in clear)
Referred AXMEDIS Object: refers to an AXMEDIS object using a url/urn, it may contain metadata of the object.
Governed AXMEDIS Object: anyone of the previous containing the license to use the object

In the following possible mappings of AXMEDIS Objects as MPEG21 digital items are reported. A tree like structure is used to represent the XML structure.

MPEG21 *Descriptors* are used to contain metadata related to the content. The *Statement* element inside the Descriptor can contain any XML or text, MPEG21 does not fix its content.
The order where *Descriptor* elements are reported is not fixed, however some of them are required (have to be present) and others are optional (may be missing). Some *Descriptors* are specified in the standard for Digital Item Identification:

- *Identifier*, used to identify the object, MPEG21 does not provide a new identification scheme but it allows to host any kind of identification code. A Registration Authority will be set up to register identification schemes to be used in MPEG21 Digital Items. A URI is used as identifier, for Example: <dii:Identifier>*urn:mpegRA:mpeg21:dii:**isrc**:US-ZO3-99-32476*</dii:Identifier> identifies an object using a ISRC code. An Identifier can be used to store the AXMEDIS Object ID.
- *RelatedIdentifier*, used to identify the work with a uri. It can be used to store the AXMEDIS Work ID. Example: <dii:RelatedIdentifier>urn:mpegRA:mpeg21:dii:iscw:T-034.524.680-1</dii:RelatedIdentifier>

The *AXInfo* element is used to contain information specific for AXMEDIS framework. Metadata like title, author, etc. and mpeg7 metadata are not stored inside AXInfo to allow MPEG21 terminals to access to these metadata even if they are not AXMEDIS compliant tools. Other AXMEDIS specific metadata related to the content can be defined (e.g. for technical information), and hosted in specific *Descriptor* elements, if a suitable standardized format is not available (e.g mpeg7).

The following example shows an example of an object with multiple descriptors:

```
<?xml version="1.0" encoding="UTF-8"?>
<DIDL xmlns="urn:mpeg:mpeg21:2002:02-DIDL-NS" xmlns:ax="urn:axmedis:01" xmlns:dii="urn:mpeg:mpeg21:2002:01-DII-
NS" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:r="urn:mpeg:mpeg21:2003:01-REL-R-NS"
xmlns:mx="urn:mpeg:mpeg21:2003:01-REL-MX-NS" xmlns:sx="urn:mpeg:mpeg21:2003:01-REL-SX-NS"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:axmedis:01 AXMEDIS.xsd">
    <Item>
        <!--
            Descriptor containing the AXOID to identify the object (REQUIRED)
         -->
        <Descriptor id="dsc_id">
            <Statement mimeType="text/xml">
                <ax:ObjectIdentifier>
                    <dii:Identifier>urn:axmedis:axoid:A001AGSHDI</dii:Identifier>
                    <Version>0</Version>
                </ax:ObjectIdentifier>
            </Statement>
        </Descriptor>
        <!--
            Descriptor containing the AXInfo containing information regarding the object (REQUIRED)
        -->
        <Descriptor id="public_dsc_ax">
            <Statement mimeType="text/xml">
                <ax:AXInfo>
                        …
                </ax:AXInfo>
            </Statement>
        </Descriptor>
        <!--
          Descriptor containing the Dublin Core information regarding the object (REQUIRED)
        -->
        <Descriptor id="public_dsc_dc">
            <Statement mimeType="text/xml">
                <rdf:Description>
                    <dc:title xml:lang="en">When the Thistle Blooms</dc:title>
                    <dc:title xml:lang="it">Quando il Cardo Sboccia</dc:title>
                    <dc:creator>Always Red</dc:creator>
                    <dc:publisher>PDQ Records</dc:publisher>
                </rdf:Description>
            </Statement>
        </Descriptor>
        <!--
           Descriptor containing the MPEG7 information regarding the object (OPTIONAL)
        -->
        <Descriptor id="public_dsc_mpeg7">
            <Statement mimeType="text/xml">
                <mpeg7:Mpeg7>
                        …
                </mpeg7:Mpeg7>
            </Statement>
        </Descriptor>
        <!--
            Component elements containing the resource (REQUIRED for single object)
        -->
        <Component id="cmp">
            <Resource mimeType="video/mp4v-es" encoding="base64">
                aadsfadsfsyd647dgd78r85hfuv8nbr8fnf985nf9g9gm569gmty9ghmg90hdhd8fhfjd9d9
                dhd8f95mnfk9gfm59fgt95mkt0jhdf8fnj587fjd67n3jf84mf00eedjf8fj58tm58fm58emds9o
                ...
            </Resource>
        </Component>
    </Item>
</DIDL>
```

References in an object to other objects can be done using the AXOID. This allows to reconstruct objects relations in any other place. An additional complexity is due to the use of temporary AXOIDs which are forbidden to be used outside the AXMEDIS Factory.

References to resources (audio, document, video, … files) can be done using a path. However have to be noted that a resource have not a unique ID this means that sharing a resource among objects is not possible.

16.1.3 Basic AXMEDIS Object:

```
DIDL
   Item
      Descriptor
         Statement
            dii:Identifier          (contains the AXOID, REQUIRED)
               urn:axmedis:obj:id:AXOID1
      Descriptor
         Statement
            ax:FingerprintAlgID     (contains the fingerprint algorithm id, REQUIRED for published obj)
      Descriptor
         Statement
            dsig:Signature   (contains the object signature, REQUIRED for published obj)
               …
      Descriptor
         Statement
      dii:Identifier          (contains any other identifier, OPTIONAL)
  Descriptor
    Statement
      dii:RelatedIdentifier   (contains the WorkID, OPTIONAL)
  Descriptor
    Statement
      dii:Type                (contains the type of object, OPTIONAL)
  Descriptor
    Statement
      ax:AXInfo               (contains AXMEDIS specific information, REQUIRED)
  Descriptor
    Descriptor
      Statement
        ax:MetadataStatus     (contains information on the status of the DC metadata, OPTIONAL)
    Descriptor
      Statement
        ax:MetadataVisibility (contains the visibility of the DC metadata, OPTIONAL, public if not
                               present)
    Descriptor
      Statement
        dsig:Signature        (contains the signature of the DC metadata, OPTIONAL)
    Statement
      rdf:Description         (contains Dublin Core metadata, REQUIRED)
  Descriptor
    Statement
      mpeg7:Mpeg7             (contains MPEG7 metadata, OPTIONAL)
  Descriptor
    Statement
      ???:XXX                 (contains any other metadata in XML, OPTIONAL)
  Component
    Resource                  (contains/refers the resource, REQUIRED)
  Component                   (another component, OPTIONAL)
    Resource                  (contains/refers the resource, REQUIRED)
  …
```

Thus a *Basic AXMEDIS Object* is structured in the following way:

```
DIDL
  Item
    OBJECT_AXOID
    OBJECT_METADATA
    CONTENT
```

where:

- *OBJECT_AXOID* is a Descriptor containing the ax:ObjectIdentifier element with the AXOID of the basic object;
- *OBJECT_METADATA* is a sequence of Descriptors containing the metadata of the basic object;
- *CONTENT* is a sequence of Components

Each Descriptor element can have inside other Descriptor elements with information on the metadata itself. This descriptors can contain:

- an *ax:MetadataStatus* element with information on the status of metadata
- an *ax:MetadataVisisbility* element with information on the visibility of the metadata when the object is protected: *public* (the default one) when the metadata has to be accessible in clear, *private* when the metadata has to be accessible only after unprotection.

Note: Multiple components will be used for HTML documents containing images inside. Normally only one component is present.

16.1.4 Protected Basic AXMEDIS Object:

A Protected AXMEDIS Object is obtained by protecting the root Item.

The *dii:Identifier* with the AXOID, the *ax:FingerprintAlgID* and the *dsig:Signature* for the protected object are stored in a AXMEDIS specific element (ax:ObjectIdentification) inside the *ipmpdidl:Identifier*.

The descriptors of the protected object are accessible in clear in the *ipmpdidl:ContentInfo* element, it should not contain another *ax:FingerprintAlgID* element or a different *dsig:Signature* element.

```
DIDL
   ipmpdidl:Item
      ipmpdidl:Identifier                (contains the AXOID, REQUIRED)
         ax:ObjectIdentification
            dii:Identifier
            ax:FingerprintAlgID
            dsig:Signature
      ipmpdidl:Info                            (contains protection information)
         ipmpinfo:IPMPInfoDescriptor
            ipmpinfo:Tool
               …
      ipmpdidl:ContentInfo               (contains metadata of the object in clear)
         didl:Item
            Descriptor
               Statement
                  dii:Identifier         (contains the AXOID, REQUIRED)
            Descriptor
               Statement
                  dii:Identifier         (contains any other identifier, OPTIONAL)
            Descriptor
               Statement
                  dii:RelatedIdentifier  (contains the WorkID, OPTIONAL)
            Descriptor
               Statement
                  dii:Type               (contains the type of object, OPTIONAL)
            Descriptor
               Statement
                  ax:AXInfo              (contains AXMEDIS specific information, REQUIRED)
            Descriptor
               Statement
                  rdf:Description        (contains Dublin Core metadata, REQUIRED)
            Descriptor
               Statement
                  mpeg7:Mpeg7            (contains MPEG7 metadata, OPTIONAL)
            Descriptor
               Statement
                  ???:XXX                (contains any other metadata in XML, OPTIONAL)
      ipmpdidl:Contents
         XXXXXXXXXXXXXXXXXX
         …
         XXXXXXXXXXXXXXXXXX
```

16.1.5 Composite AXMEDIS Object:

A composite object obtained by composing objects O_1, O_2, … O_n is structured as follows:

```
DIDL
   Item
      OBJECT_AXOID
      OBJECT_METADATA
      FIRST_ITEM[O1]
      FIRST_ITEM[O2]
      …
      FIRST_ITEM[On]
```

where:

- *OBJECT_AXOID* is a Descriptor containing the AXOID of the composed object;
- *OBJECT_METADATA* is a sequence of Descriptors containing the metadata of the whole object
- FIRST_ITEM[*O*] is a function to get the first child item of the object, it is used to skip the DIDL tag.

the following is an example of double composition:

AXOID1 = COMPOSE(AXOID2, AXOID3 = COMPOSE(AXOID4, AXOID5))

AXOID1 – a composite object
- AXOID2 – a basic object
- AXOID3 – a composite object
    - AXOID4 – a basic object
    - AXOID5 – a basic object

```
DIDL
   Item
      Descriptor              (contains AXOID1)
      Descriptor              (contains AXinfo)
      Descriptor              (contains Dublin Core metadata)
      Item
         Descriptor           (contains AXOID2)
         Descriptor           (contains AXinfo)
         Descriptor           (contains Dublin Core metadata)
         Component
      Item
         Descriptor           (contains AXOID3)
         Descriptor           (contains AXinfo)
         Descriptor           (contains Dublin Core metadata)
         Item
            Descriptor        (contains AXOID4)
            Descriptor        (contains AXinfo)
            Descriptor        (contains Dublin Core metadata)
            Component
         Item
            Descriptor        (contains AXOID5)
            Descriptor        (contains AXinfo)
            Descriptor        (contains Dublin Core metadata)
            Component
```

16.1.6 Protected Composite AXMEDIS Object:

A Protected Composite AXMEDIS Object is obtained, as for basic objects, protecting the root Item.

```
DIDL
   ipmpdidl:Item
      ipmpdidl:Identifier
         ax:ObjectIdentification
            dii:Identifier           (contains AXOID1)
            FingerprintAlgID         (contains the fingerprint algorithm used)
            dsig:Signature           (contains the signature for the protected data)
               …
      ipmpdidl:Info                          (contains protection information)
      ipmpdidl:ContentInfo           (contains metadata of the objectin clear)
         didl:Item                           (contains the metadata of the object)
            Descriptor                       (contains AXOID1)
            Descriptor                       (contains AXinfo)
            Descriptor                       (contains Dublin Core metadata)
            Item
               Descriptor                    (contains AXOID2)
               Descriptor                    (contains AXinfo)
               Descriptor                    (contains Dublin Core metadata)
            Item
               Descriptor                    (contains AXOID3)
               Descriptor                    (contains AXinfo)
               Descriptor                    (contains Dublin Core metadata)
               Item
                  Descriptor                 (contains AXOID4)
                  Descriptor                 (contains AXinfo)
                  Descriptor                 (contains Dublin Core metadata)
               Item
            Descriptor                  (contains AXOID5)
            Descriptor                  (contains AXinfo)
            Descriptor                  (contains Dublin Core metadata)
    ipmpdidl:Contents
        XXXXXXXXXXXXXXXXXX
        …
        XXXXXXXXXXXXXXXXXX
```

16.1.7 Referred AXMEDIS Object

A referred object may contain metadata of the referred object and it references to the real AXMEDIS object.
A referred object my be used in places where should be any other kind of AXMEDIS object (basic, composite, basic protected, composite protected)
Referred objects can be used to produce query/promotional objects.
The following is an example.

```
DIDL
  xi:include ref=…AXOID1…
    xi:fallback
      didl:Item
        Descriptor                      (contains AXOID1)
        Descriptor                      (contains AXinfo)
        Descriptor                      (contains Dublin Core metadata)
        Item
          Descriptor                    (contains AXOID2)
          Descriptor                    (contains AXinfo)
          Descriptor                    (contains Dublin Core metadata)
        Item
          Descriptor                    (contains AXOID3)
          Descriptor                    (contains AXinfo)
          Descriptor                    (contains Dublin Core metadata)
          Item
            Descriptor                  (contains AXOID4)
            Descriptor                  (contains AXinfo)
            Descriptor                  (contains Dublin Core metadata)
          Item
            Descriptor                  (contains AXOID5)
            Descriptor                  (contains AXinfo)
            Descriptor                  (contains Dublin Core metadata)
```

16.1.8 Governed AXMEDIS Object:

A Governed AXMEDIS Object contains the licence inside a descriptor like in the following example:

```
DIDL
  Item
    Descriptor
      Statement
        dii:Identifier          (contains the AXOID, REQUIRED)
    Descriptor
      Statement
        dii:RelatedIdentifier   (contains the WorkID, OPTIONAL)
    Descriptor
      Statement
        ax:AXInfo               (contains AXMEDIS specific information, REQUIRED)
    Descriptor
      Statement
        rdf:Description         (contains Dublin Core metadata, REQUIRED)
    Descriptor
      Statement
        mpeg7:Mpeg7             (contains MPEG7 metadata, OPTIONAL)
    Descriptor
      Statement
        r:license               (contains the license for the object, OPTIONAL)
    …                           (any other of the previous structures)
```

16.1.9 AXMEDIS Metadata Model (DSI, EPFL, …..)

Metadata information related to an object, as seen in previous section, is split among various Descriptors. These Descriptors can contain:

- identification information (as standardized by MPEG21) for AXOID, AXWID and Type
- AXMEDIS specific information regarding object life-cycle (in AXInfo)
- Dublin Core metadata
- MPEG 7 metadata
- any other metadata represented in XML

These descriptors may contain other Descriptors with information on the metadata itself like:

- metadata status explaining the current status of the metadata provided (e.g. "to be revised", "minimal");
- metadata visibility (public/private) telling if the metadata should be provided in clear when the object is protected (default if not stated) or should be accessible only after deprotection;
- metadata certification containing a signature for the statement contained in the descriptor.

Thus the structure for a descriptor is:

```
Descriptor
   Descriptor
      Statement
         ax:MetadataStatus
            "to be revised"
   Descriptor
      Statement
         ax:MetadataVisibility
            "private"
   Descriptor
      Statement
         dsig:Signature
   Statement
      mpeg7:Mpeg7
```

The descriptors can be found in any order and are all optional.

16.1.9.1 Dublin Core Metadata

The Dublin Core Metadata Initiative produced RDF schemas and XML schemas to allow the representation of Dublin Core metadata (for details see http://dublincore.org/) AXMEDIS will use this schemas to represent basic metadata.

The 15 basic metadata terms defined in Dublin Core are:

- contributor
- coverage
- creator
- date
- description
- format
- identifier
- language
- publisher
- relation
- rights
- source
- subject
- title
- type

each term may be repeated more than one time meaning that all of them applies to the resource described. Terms may be written in a different language and the language used is identified by a xml:lang attribute.
A resource with:

```
<dc:creator>J. Doe<dc:creator>
<dc:creator>M. White<dc:creator>
<dc:title xml:lang=”en”>A title<dc:title>
<dc:title xml:lang=”it”>Un titolo<dc:title>
```

has two authors (J. Doe and M. White) and a title expressed in English and Italian.

In the following table is reported the definition for the DC terms as found in (http://dublincore.org/documents/dcmi-terms/).

contributor	
URI:	http://purl.org/dc/elements/1.1/contributor
Definition:	An entity responsible for making contributions to the content of the resource.
Comment:	Examples of a Contributor include a person, an organisation, or a service. Typically, the name of a Contributor should be used to indicate the entity.
coverage	
URI:	http://purl.org/dc/elements/1.1/coverage
Definition:	The extent or scope of the content of the resource.
Comment:	Coverage will typically include spatial location (a place name or geographic coordinates), temporal period (a period label, date, or date range) or jurisdiction (such as a named administrative entity). Recommended best practice is to select a value from a controlled vocabulary (for example, the Thesaurus of Geographic Names [TGN]) and that, where appropriate, named places or time periods be used in preference to numeric identifiers such as sets of coordinates or date ranges.
References:	[TGN] http://www.getty.edu/research/tools/vocabulary/tgn/index.html
creator	
URI:	http://purl.org/dc/elements/1.1/creator
Definition:	An entity primarily responsible for making the content of the resource.
Comment:	Examples of a Creator include a person, an organisation, or a service. Typically, the name of a Creator should be used to indicate the entity.
date	
URI:	http://purl.org/dc/elements/1.1/date
Definition:	A date associated with an event in the life cycle of the resource.
Comment:	Typically, Date will be associated with the creation or availability of the resource. Recommended best practice for encoding the date value is defined in a profile of ISO 8601 [W3CDTF] and follows the YYYY-MM-DD format.
References:	[W3CDTF] http://www.w3.org/TR/NOTE-datetime
description	
URI:	http://purl.org/dc/elements/1.1/description
Definition:	An account of the content of the resource.
Comment:	Description may include but is not limited to: an abstract, table of contents, reference to a graphical representation of content or a free-text account of the content.
format	
URI:	http://purl.org/dc/elements/1.1/format
Definition:	The physical or digital manifestation of the resource.
Comment:	Typically, Format may include the media-type or dimensions of the resource. Format may be used to determine the software, hardware or other equipment needed to display or operate the resource. Examples of dimensions include size and duration. Recommended best practice is to select a value from a controlled vocabulary (for example, the list of Internet Media Types [MIME] defining computer media formats).

References:	[MIME] http://www.isi.edu/in-notes/iana/assignments/media-types/media-types
identifier	
URI:	http://purl.org/dc/elements/1.1/identifier
Definition:	An unambiguous reference to the resource within a given context.
Comment:	Recommended best practice is to identify the resource by means of a string or number conforming to a formal identification system. Example formal identification systems include the Uniform Resource Identifier (URI) (including the Uniform Resource Locator (URL)), the Digital Object Identifier (DOI) and the International Standard Book Number (ISBN).
language	
URI:	http://purl.org/dc/elements/1.1/language
Definition:	A language of the intellectual content of the resource.
Comment:	Recommended best practice is to use RFC 3066 [RFC3066], which, in conjunction with ISO 639 [ISO639], defines two- and three-letter primary language tags with optional subtags. Examples include "en" or "eng" for English, "akk" for Akkadian, and "en-GB" for English used in the United Kingdom.
References:	[RFC3066] http://www.ietf.org/rfc/rfc3066.txt
References:	[ISO639] http://www.loc.gov/standards/iso639-2/
publisher	
URI:	http://purl.org/dc/elements/1.1/publisher
Definition:	An entity responsible for making the resource available
Comment:	Examples of a Publisher include a person, an organisation, or a service. Typically, the name of a Publisher should be used to indicate the entity.
relation	
URI:	http://purl.org/dc/elements/1.1/relation
Definition:	A reference to a related resource.
Comment:	Recommended best practice is to reference the resource by means of a string or number conforming to a formal identification system.
rights	
URI:	http://purl.org/dc/elements/1.1/rights
Definition:	Information about rights held in and over the resource.
Comment:	Typically, a Rights element will contain a rights management statement for the resource, or reference a service providing such information. Rights information often encompasses Intellectual Property Rights (IPR), Copyright, and various Property Rights. If the Rights element is absent, no assumptions can be made about the status of these and other rights with respect to the resource.
source	
URI:	http://purl.org/dc/elements/1.1/source
Definition:	A reference to a resource from which the present resource is derived.
Comment:	The present resource may be derived from the Source resource in whole or in part. Recommended best practice is to reference the resource by means of a string or number conforming to a formal identification system.
subject	
URI:	http://purl.org/dc/elements/1.1/subject
Definition:	The topic of the content of the resource.
Comment:	Typically, a Subject will be expressed as keywords, key phrases or classification codes that describe a topic of the resource. Recommended best practice is to select a value from a controlled vocabulary or formal classification scheme.
title	
URI:	http://purl.org/dc/elements/1.1/title
Definition:	A name given to the resource.

Comment:	Typically, a Title will be a name by which the resource is formally known.
Multiplicity:	*0..many (title in many languages)*
type	
URI:	http://purl.org/dc/elements/1.1/type
Definition:	The nature or genre of the content of the resource.
Comment:	Type includes terms describing general categories, functions, genres, or aggregation levels for content. Recommended best practice is to select a value from a controlled vocabulary (for example, the DCMI Type Vocabulary [DCMITYPE]). To describe the physical or digital manifestation of the resource, use the Format element.
References:	[DCMITYPE] http://dublincore.org/documents/dcmi-type-vocabulary/

Many other terms has been introduced as refinements of these basic terms like:

- *abstract* as refinement of *description*
- *alternative* as refinement of *title*
- *comformsTo* as refinemet of *relation*
- etc.

The full list is reported in the following and additional information can be found in (http://dublincore.org/documents/dcmi-terms/).

abstract	
Definition:	A summary of the content of the resource.
Refines:	http://purl.org/dc/elements/1.1/description
accessRights	
Definition:	Information about who can access the resource or an indication of its security status.
Comment:	Access Rights may include information regarding access or restrictions based on privacy, security or other regulations.
Refines:	http://purl.org/dc/elements/1.1/rights
alternative	
Definition:	Any form of the title used as a substitute or alternative to the formal title of the resource.
Comment:	This qualifier can include Title abbreviations as well as translations.
Refines:	http://purl.org/dc/elements/1.1/title
audience	
Definition:	A class of entity for whom the resource is intended or useful.
Comment:	A class of entity may be determined by the creator or the publisher or by a third party.
available	
Definition:	Date (often a range) that the resource will become or did become available.
Refines:	http://purl.org/dc/elements/1.1/date
bibliographicCitation	
Definition:	A bibliographic reference for the resource.
Comment:	Recommended practice is to include sufficient bibliographic detail to identify the resource as unambiguously as possible, whether or not the citation is in a standard form.
Refines:	http://purl.org/dc/elements/1.1/identifier
conformsTo	
Definition:	A reference to an established standard to which the resource conforms.
Refines:	http://purl.org/dc/elements/1.1/relation
created	
Definition:	Date of creation of the resource.

Refines:	http://purl.org/dc/elements/1.1/date
dateAccepted	
Definition:	Date of acceptance of the resource (e.g. of thesis by university department, of article by journal, etc.).
Refines:	http://purl.org/dc/elements/1.1/date
dateCopyrighted	
Definition:	Date of a statement of copyright.
Refines:	http://purl.org/dc/elements/1.1/date
dateSubmitted	
Definition:	Date of submission of the resource (e.g. thesis, articles, etc.).
Refines:	http://purl.org/dc/elements/1.1/date
educationLevel	
Definition:	A general statement describing the education or training context. Alternatively, a more specific statement of the location of the audience in terms of its progression through an education or training context.
Refines:	http://purl.org/dc/terms/audience
extent	
Definition:	The size or duration of the resource.
Refines:	http://purl.org/dc/elements/1.1/format
hasFormat	
Definition:	The described resource pre-existed the referenced resource, which is essentially the same intellectual content presented in another format.
Refines:	http://purl.org/dc/elements/1.1/relation
hasPart	
Definition:	The described resource includes the referenced resource either physically or logically.
Refines:	http://purl.org/dc/elements/1.1/relation
hasVersion	
Definition:	The described resource has a version, edition, or adaptation, namely, the referenced resource.
Refines:	http://purl.org/dc/elements/1.1/relation
isFormatOf	
Definition:	The described resource is the same intellectual content of the referenced resource, but presented in another format.
Refines:	http://purl.org/dc/elements/1.1/relation
isPartOf	
Definition:	The described resource is a physical or logical part of the referenced resource.
Refines:	http://purl.org/dc/elements/1.1/relation
isReferencedBy	
Definition:	The described resource is referenced, cited, or otherwise pointed to by the referenced resource.
Refines:	http://purl.org/dc/elements/1.1/relation
isReplacedBy	
Definition:	The described resource is supplanted, displaced, or superseded by the referenced resource.
Refines:	http://purl.org/dc/elements/1.1/relation
isRequiredBy	
Definition:	The described resource is required by the referenced resource, either physically or logically.
Refines:	http://purl.org/dc/elements/1.1/relation
issued	

Definition:	Date of formal issuance (e.g., publication) of the resource.
Refines:	http://purl.org/dc/elements/1.1/date
isVersionOf	
Definition:	The described resource is a version, edition, or adaptation of the referenced resource. Changes in version imply substantive changes in content rather than differences in format.
Refines:	http://purl.org/dc/elements/1.1/relation
license	
Definition:	A legal document giving official permission to do something with the resource.
Comment:	Recommended best practice is to identify the license using a URI. Examples of such licenses can be found at http://creativecommons.org/licenses/.
Refines:	http://purl.org/dc/elements/1.1/rights
mediator	
Definition:	A class of entity that mediates access to the resource and for whom the resource is intended or useful.
Comment:	The audiences for a resource are of two basic classes: (1) an ultimate beneficiary of the resource, and (2) frequently, an entity that mediates access to the resource. The mediator element refinement represents the second of these two classes.
Refines:	http://purl.org/dc/terms/audience
medium	
Definition:	The material or physical carrier of the resource.
Refines:	http://purl.org/dc/elements/1.1/format
modified	
Definition:	Date on which the resource was changed.
Refines:	http://purl.org/dc/elements/1.1/date
provenance	
Definition:	A statement of any changes in ownership and custody of the resource since its creation that are significant for its authenticity, integrity and interpretation.
Comment:	The statement may include a description of any changes successive custodians made to the resource.
references	
Definition:	The described resource references, cites, or otherwise points to the referenced resource.
Refines:	http://purl.org/dc/elements/1.1/relation
replaces	
Definition:	The described resource supplants, displaces, or supersedes the referenced resource.
Refines:	http://purl.org/dc/elements/1.1/relation
requires	
Definition:	The described resource requires the referenced resource to support its function, delivery, or coherence of content.
Refines:	http://purl.org/dc/elements/1.1/relation
rightsHolder	
Definition:	A person or organization owning or managing rights over the resource.
Comment:	Recommended best practice is to use the URI or name of the Rights Holder to indicate the entity.
spatial	
Definition:	Spatial characteristics of the intellectual content of the resource.
Refines:	http://purl.org/dc/elements/1.1/coverage
tableOfContents	
Definition:	A list of subunits of the content of the resource.

Refines:	http://purl.org/dc/elements/1.1/description
temporal	
Definition:	Temporal characteristics of the intellectual content of the resource.
Refines:	http://purl.org/dc/elements/1.1/coverage
valid	
Definition:	Date (often a range) of validity of a resource.
Refines:	http://purl.org/dc/elements/1.1/date

AXMEDIS will support all these metadata (basic and refined), however in case of collision with information stored in other descriptors like in *AXInfo* or *Identifier*s, these ones are considered valid and the DC ones are dependent. Meaning that in case of inconsistency between these information the AXInfo and the Identifiers have a higher priority and can be used to fix the DC values (under user control).

Have to be noted that not all refined elements may have sense in the AXMEDIS context, thus some of them may be not considered by some applications (e.g. DB may not index some metadata).

16.1.10 Examples of AXMEDIS Objects

Basic AXMEDIS Object
The following is an example of a Basic AXMEDIS Object

```
<?xml version="1.0" encoding="UTF-8"?>
<DIDL xmlns="urn:mpeg:mpeg21:2002:02-DIDL-NS" xmlns:ax="urn:axmedis:01" xmlns:dii="urn:mpeg:mpeg21:2002:01-DII-NS"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:r="urn:mpeg:mpeg21:2003:01-REL-R-NS" xmlns:mx="urn:mpeg:mpeg21:2003:01-
REL-MX-NS" xmlns:sx="urn:mpeg:mpeg21:2003:01-REL-SX-NS" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:axmedis:01 AXMEDIS-new.xsd">
    <Item>
        <!--
            Descriptor containing the AXOID to identify the object (REQUIRED)
         -->
        <Descriptor id="dsc_id">
            <Statement mimeType="text/xml">
                <dii:Identifier>urn:axmedis:axoid:OBJ_A001AGSHDI</dii:Identifier>
            </Statement>
        </Descriptor>
        <Descriptor id="dsc_fingprt">
            <Statement mimeType="text/xml">
                <ax:FingerprintAlgID>axobjFingerprint</ax:FingerprintAlgID>
            </Statement>
        </Descriptor>
        <Descriptor id="dsc_sign">
            <Statement mimeType="text/xml">
                <dsig:Signature>
                    <dsig:SignedInfo>
                        <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-20000710"/>
                        <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa"/>
                        <dsig:Reference>
                            <dsig:Transforms>
                                <dsig:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
                                <dsig:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#null"/>
                            </dsig:Transforms>
                            <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                            <dsig:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</dsig:DigestValue>
                        </dsig:Reference>
                    </dsig:SignedInfo>

    <dsig:SignatureValue>MC0CFFrVLtRlkMc3Daon4BqqnkhCOlEaAhUAk8pH1iRNK+q1I+sisDTz2TFEALE=</dsig:SignatureValue>
                </dsig:Signature>
            </Statement>
        </Descriptor>
        <!--
            Descriptor containing the RelatedIdentifier to identify the work (OPTIONAL)
        -->
        <Descriptor id="dsc_rel_id">
```

```
            <Statement mimeType="text/xml">
                <dii:RelatedIdentifier>urn:mpegRA:mpeg21:dii:iscw:T-034.524.680-1</dii:RelatedIdentifier>
            </Statement>
        </Descriptor>
        <!--
          Descriptor containing the Type to identify the type of object (OPTIONAL)
        -->
        <Descriptor id="dsc_type">
            <Statement mimeType="text/xml">
                <dii:Type>Music</dii:Type>
            </Statement>
        </Descriptor>
        <!--
          Descriptor containing the AXInfo containing information regarding the object (REQUIRED)
        -->
        <Descriptor id="dsc_axinfo">
            <Statement mimeType="text/xml">
                <ax:AXInfo>
                    <ax:ObjectCreator>
                        <ax:AXCID>... a creator ID ...</ax:AXCID>
                        <ax:ObjectCreatorName>J. Doe</ax:ObjectCreatorName>
                        <ax:ObjectCreatorURL>mailto:jdoe@invideo.com</ax:ObjectCreatorURL>
                        <ax:ObjectCreatorCompany>InVideo</ax:ObjectCreatorCompany>
                        <ax:ObjectCreatorCompanyURL>http://www.invideo.com</ax:ObjectCreatorCompanyURL>
                        <ax:ObjectCreatorNationality>US</ax:ObjectCreatorNationality>
                    </ax:ObjectCreator>
                    <ax:AccessMode>read_write</ax:AccessMode>
                    <ax:CreationDate>2004-12-27T15:00:00</ax:CreationDate>
                    <ax:LastModificationDate>2004-12-27T16:43:00</ax:LastModificationDate>
                    <ax:Version>1</ax:Version>
                    <ax:Revision>1</ax:Revision>
                    <ax:ObjectStatus>production</ax:ObjectStatus>
                    <ax:ObjectType>BASIC</ax:ObjectType>
                    <!--
                      History of the object
                    -->
                    <ax:History>
                        <ax:ObjVersion number="1">
                            <ax:When>2004-12-27T16:27:00</ax:When>
                            <ax:Who>J. Doe</ax:Who>
                            <ax:Where>
                                <ax:Organization>InVideo</ax:Organization>
                                <ax:Site>Atlanta</ax:Site>
                                <ax:Machine>JDOE_01</ax:Machine>
                            </ax:Where>
                            <ax:What>
                                <ax:Description>First version</ax:Description>
                            </ax:What>
                        </ax:ObjVersion>
                        <ax:ObjVersion number="2">
                            <ax:When>2004-12-27T16:27:00</ax:When>
                            <ax:Who>J. Doe</ax:Who>
                            <ax:Where>
                                <ax:Organization>InVideo</ax:Organization>
                                <ax:Site>Atlanta</ax:Site>
                                <ax:Machine>JDOE_05</ax:Machine>
                            </ax:Where>
                            <ax:What>
                                <ax:Commands>
                                    <ax:Cmd>
                                        <ax:AXTID>.... a Tool ID ...</ax:AXTID>
                                        <ax:AXTTID>... a Tool Type ID...</ax:AXTTID>
                                        <ax:AXRTID>... a Real Tool ID ...</ax:AXRTID>
                                        <ax:Operation>
                                            <ax:Name>Add</ax:Name>
                                        </ax:Operation>
                                        <ax:Revision>1</ax:Revision>
                                    </ax:Cmd>
                                    <!-- To be completed -->
                                </ax:Commands>
                            </ax:What>
                        </ax:ObjVersion>
                    </ax:History>
                    <!--
                      Workflow information
                    -->
          <ax:Workflow>
            <ax:WorkItemID>... a work item ID... </ax:WorkItemID>
            <ax:WorkspaceInstanceID>.... a workspace instance ID ...</ax:WorkspaceInstanceID>
          </ax:Workflow>
          <!--
           Rights potentially available on the object
          -->
          <ax:PotentialAvailableRights>
            <ax:LicensingURL>http://www.axmedis.org</ax:LicensingURL>
            <ax:PARStatus/>
            <r:license>
              <r:grantGroup>
                <r:grant>
                  <mx:play/>
                  <r:allConditions>
                    <sx:validityIntervalFloating>
                      <sx:duration>P1M</sx:duration>
                    </sx:validityIntervalFloating>
                    <r:validityInterval>
                      <r:notAfter>2010-01-01T00:00:00</r:notAfter>
                    </r:validityInterval>
                  </r:allConditions>
                </r:grant>
                <r:grant>
                  <mx:move/>
                </r:grant>
                <r:grant>
                  <mx:delete/>
                </r:grant>
              </r:grantGroup>
            </r:license>
          </ax:PotentialAvailableRights>
        </ax:AXInfo>
      </Statement>
    </Descriptor>
    <!--
     Descriptor containing the Dublin Core information regarding the object (REQUIRED)
    -->
    <Descriptor id="dsc_dc">
      <Descriptor>
        <Statement mimeType="text/xml">
          <ax:MetadataStatus>to be revised</ax:MetadataStatus>
        </Statement>
      </Descriptor>
      <Statement mimeType="text/xml">
        <rdf:Description>
          <dc:title xml:lang="en">When the Thistle Blooms</dc:title>
          <dc:title xml:lang="it">Quando il Cardo Sboccia</dc:title>
          <dc:creator>Always Red</dc:creator>
          <dc:publisher>PDQ Records</dc:publisher>
        </rdf:Description>
      </Statement>
    </Descriptor>
    <!--
     Descriptor containing the MPEG7 information regarding the object (OPTIONAL)
    -->
    <Descriptor id="dsc_mpeg7">
      <Descriptor>
        <Statement mimeType="text/xml">
          <ax:MetadataVisibility>private</ax:MetadataVisibility>
        </Statement>
      </Descriptor>
      <Statement mimeType="text/xml">
        <mpeg7:Mpeg7>
          <mpeg7:DescriptionUnit xsi:type="MediaProfileType">
            <mpeg7:MediaFormat>
              <mpeg7:VisualCodingFormat href="urn:mpeg:mpeg7:cs:MPEG7VisualCodingFormatCS:3.1.2"/>
              <mpeg7:BitRate>64000</mpeg7:BitRate>
            </mpeg7:MediaFormat>
            <mpeg7:MediaQuality>
              <mpeg7:QualityRating ratingType="objective">
                <mpeg7:RatingValue>35.6</mpeg7:RatingValue>
                <mpeg7:RatingMetric>
                  <mpeg7:QualityRatingScheme href="urn:mpeg:mpeg7:cs:MPEG-
7QualityRatingSchemeCS:2.3"/>
                  <mpeg7:RatingStyle>higherBetter</mpeg7:RatingStyle>
```

```
                                        </mpeg7:RatingMetric>
                                    </mpeg7:QualityRating>
                                </mpeg7:MediaQuality>
                            </mpeg7:DescriptionUnit>
                        </mpeg7:Mpeg7>
                    </Statement>
                </Descriptor>
                <!--
                    Component elements containing the resource (REQUIRED for single object)
                -->
                <Component id="cmp">
                    <Resource mimeType="video/mp4v-es" encoding="base64">
                        aadsfadsfsyd647dgd78r85hfuv8nbr8fnf985nf9g9gm569gmty9ghmg90hdhd8fhfjd9d9
                        dhd8f95mnfk9gfm59fgt95mkt0jhdf8fnj587fjd67n3jf84mf00eedjf8fj58tm58fm58emds9o
                        ...
                    </Resource>
                </Component>
            </Item>
</DIDL>
```

Protected Basic AXMEDIS Object

```
<DIDL xmlns="urn:mpeg:mpeg21:2002:02-DIDL-NS" xmlns:ipmpdidl="urn:mpeg:mpeg21:2004:01-IPMPDIDL-NS"
xmlns:ipmp="urn:mpeg:mpeg21:2004:01-IPMP-NS" xmlns:ax="urn:axmedis:01" xmlns:dii="urn:mpeg:mpeg21:2002:01-DII-NS"
xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:r="urn:mpeg:mpeg21:2003:01-REL-R-NS" xmlns:mx="urn:mpeg:mpeg21:2003:01-
REL-MX-NS" xmlns:sx="urn:mpeg:mpeg21:2003:01-REL-SX-NS" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:axmedis:01 AXMEDIS-new.xsd">
    <ipmpdidl:Item>
        <ipmpdidl:Identifier>
            <ax:ObjectIdentification>
                <dii:Identifier>urn:axmedis:obj:id:9d00eda5-cb23-48c3-8675-972fce3e0a22</dii:Identifier>
                <ax:FingerprintAlgID>axobjFingerprint</ax:FingerprintAlgID>
                <dsig:Signature>
                    <dsig:SignedInfo>
                        <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-20000710"/>
                        <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa"/>
                        <dsig:Reference>
                            <dsig:Transforms>
                                <dsig:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
                                <dsig:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#null"/>
                            </dsig:Transforms>
                            <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                            <dsig:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</dsig:DigestValue>
                        </dsig:Reference>
                    </dsig:SignedInfo>

    <dsig:SignatureValue>MC0CFFrVLtRlkMc3Daon4BqqnkhCOlEaAhUAk8pH1iRNK+q1I+sisDTz2TFEALE=</dsig:SignatureValue>
                </dsig:Signature>
            </ax:ObjectIdentification>
        </ipmpdidl:Identifier>
        <ipmpdidl:Info>
            <ipmp:IPMPInfoDescriptor>
                <ipmp:Tool>
                    <ipmp:ToolBaseDescription>
                        <ipmp:IPMPToolID>urn:mpegRA:mpeg21:IPMP:ABC005:77:29</ipmp:IPMPToolID>
                        <ipmp:Remote ref="urn:IPMPToolsServer:ToolEnc005-3484"/>
                    </ipmp:ToolBaseDescription>
                </ipmp:Tool>
                <ipmp:Tool>
                    <ipmp:ToolBaseDescription>
                        <ipmp:IPMPToolID>urn:mpegRA:mpeg21:IPMP:ABC064:55:86</ipmp:IPMPToolID>
                        <ipmp:Remote ref="urn:IPMPToolsServer:ToolWat005-6393"/>
                    </ipmp:ToolBaseDescription>
                </ipmp:Tool>
            </ipmp:IPMPInfoDescriptor>
        </ipmpdidl:Info>
        <ipmpdidl:ContentInfo>
            <Item>
                <!--
                Descriptor containing the AXOID to identify the object (REQUIRED)
                -->
                <Descriptor id="dsc_id">
                    <Statement mimeType="text/xml">
                        <dii:Identifier>urn:axmedis:obj:id:9d00eda5-cb23-48c3-8675-972fce3e0a22</dii:Identifier>
                    </Statement>
                </Descriptor>
                <!--
                Descriptor containing the RelatedIdentifier to identify the work (OPTIONAL)
                -->
                <Descriptor id="dsc_rel_id">
                    <Statement mimeType="text/xml">
                        <dii:RelatedIdentifier>urn:mpegRA:mpeg21:dii:iscw:T-034.524.680-1</dii:RelatedIdentifier>
                    </Statement>
                </Descriptor>
                <!--
                Descriptor containing the Type to identify the type of object (OPTIONAL)
                -->
                <Descriptor id="dsc_type">
                    <Statement mimeType="text/xml">
                        <dii:Type>Music</dii:Type>
                    </Statement>
                </Descriptor>
                <!--
                Descriptor containing the AXInfo containing information regarding the object (REQUIRED)
                -->
                <Descriptor id="dsc_axinfo">
                    <Statement mimeType="text/xml">
                        <ax:AXInfo>
                            <ax:ObjectCreator>
                                <ax:AXCID>... a creator ID ...</ax:AXCID>
                                <ax:ObjectCreatorName>J. Doe</ax:ObjectCreatorName>
                                <ax:ObjectCreatorURL>mailto:jdoe@invideo.com</ax:ObjectCreatorURL>
                                <ax:ObjectCreatorCompany>InVideo</ax:ObjectCreatorCompany>
                                <ax:ObjectCreatorCompanyURL>http://www.invideo.com</ax:ObjectCreatorCompanyURL>
                                <ax:ObjectCreatorNationality>US</ax:ObjectCreatorNationality>
                            </ax:ObjectCreator>
                            <ax:AccessMode>read_write</ax:AccessMode>
                            <ax:CreationDate>2004-12-27T15:00:00</ax:CreationDate>
                            <ax:LastModificationDate>2004-12-27T16:43:00</ax:LastModificationDate>
                            <ax:Version>1</ax:Version>
                            <ax:Revision>1</ax:Revision>
                            <ax:ObjectStatus>production</ax:ObjectStatus>
                            <ax:ObjectType>BASIC</ax:ObjectType>
                            <!--
                            History of the object
                            -->
                            <ax:History>
                                <ax:ObjVersion number="1">
                                    <ax:When>2004-12-27T16:27:00</ax:When>
                                    <ax:Who>J. Doe</ax:Who>
                                    <ax:Where>
                                        <ax:Organization>InVideo</ax:Organization>
                                        <ax:Site>Atlanta</ax:Site>
                                        <ax:Machine>JDOE_01</ax:Machine>
                                    </ax:Where>
                                    <ax:What>
                                        <ax:Description>First version</ax:Description>
                                    </ax:What>
                                </ax:ObjVersion>
                                <ax:ObjVersion number="2">
                                    <ax:When>2004-12-27T16:27:00</ax:When>
                                    <ax:Who>J. Doe</ax:Who>
                                    <ax:Where>
                                        <ax:Organization>InVideo</ax:Organization>
                                        <ax:Site>Atlanta</ax:Site>
                                        <ax:Machine>JDOE_05</ax:Machine>
                                    </ax:Where>
                                    <ax:What>
                                        <ax:Commands>
                                            <ax:Cmd>
                                                <ax:AXTID>.... a Tool ID ...</ax:AXTID>
                                                <ax:AXTTID>... a Tool Type ID...</ax:AXTTID>
                                                <ax:AXRTID>... a Real Tool ID ...</ax:AXRTID>
                                                <ax:Operation>
                                                    <ax:Name>Add</ax:Name>
                                                </ax:Operation>
                                                <ax:Revision>1</ax:Revision>
                                            </ax:Cmd>
```

```xml
                      <!-- To be completed -->
                    </ax:Commands>
                  </ax:What>
                </ax:ObjVersion>
              </ax:History>
              <!--
              Workflow information
              -->
              <ax:Workflow>
                <ax:WorkItemID>... a work item ID... </ax:WorkItemID>
                <ax:WorkspaceInstanceID>.... a workspace instance ID ...</ax:WorkspaceInstanceID>
              </ax:Workflow>
              <!--
              Rights potentially available on the object
              -->
              <ax:PotentialAvailableRights>
                <ax:LicensingURL>http://www.axmedis.org</ax:LicensingURL>
                <ax:PARStatus/>
                <r:license>
                  <r:grantGroup>
                    <r:grant>
                      <mx:play/>
                      <r:allConditions>
                        <sx:validityIntervalFloating>
                          <sx:duration>P1M</sx:duration>
                        </sx:validityIntervalFloating>
                        <r:validityInterval>
                          <r:notAfter>2010-01-01T00:00:00</r:notAfter>
                        </r:validityInterval>
                      </r:allConditions>
                    </r:grant>
                    <r:grant>
                      <mx:move/>
                    </r:grant>
                    <r:grant>
                      <mx:delete/>
                    </r:grant>
                  </r:grantGroup>
                </r:license>
              </ax:PotentialAvailableRights>
            </ax:AXInfo>
          </Statement>
        </Descriptor>
        <!--
          Descriptor containing the Dublin Core information regarding the object (REQUIRED)
        -->
        <Descriptor id="dsc_dc">
          <Descriptor>
            <Statement mimeType="text/xml">
              <ax:MetadataStatus>to be revised</ax:MetadataStatus>
            </Statement>
          </Descriptor>
          <Descriptor>
            <Statement mimeType="text/xml">
              <dsig:Signature>
                <dsig:SignedInfo>
                  <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-
20000710"/>
                  <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa"/>
                  <dsig:Reference>
                    <dsig:Transforms>
                      <dsig:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
                      <dsig:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#null"/>
                    </dsig:Transforms>
                    <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                    <dsig:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</dsig:DigestValue>
                  </dsig:Reference>
                </dsig:SignedInfo>

  <dsig:SignatureValue>MC0CFFrVLtRlkMc3Daon4BqqnkhCOlEaAhUAk8pH1iRNK+q1I+sisDTz2TFEALE=</dsig:SignatureValue>
              </dsig:Signature>
            </Statement>
          </Descriptor>
          <Statement mimeType="text/xml">
            <rdf:Description>
              <dc:title xml:lang="en">When the Thistle Blooms</dc:title>
```

```
                                        <dc:title xml:lang="it">Quando il Cardo Sboccia</dc:title>
                                        <dc:creator>Always Red</dc:creator>
                                        <dc:publisher>PDQ Records</dc:publisher>
                                    </rdf:Description>
                                </Statement>
                            </Descriptor>
                            <!--
                            Descriptor containing the MPEG7 information regarding the object removed because it is private
                            -->
                        </Item>
                    </ipmpdidl:ContentInfo>
                    <ipmpdidl:Contents>agsdhsjdddjfhf945734md9v784nf.... 7283udfhjdf94jdbnhcysd8e</ipmpdidl:Contents>
                </ipmpdidl:Item>
</DIDL>
```

17 AXInfo (DSI)

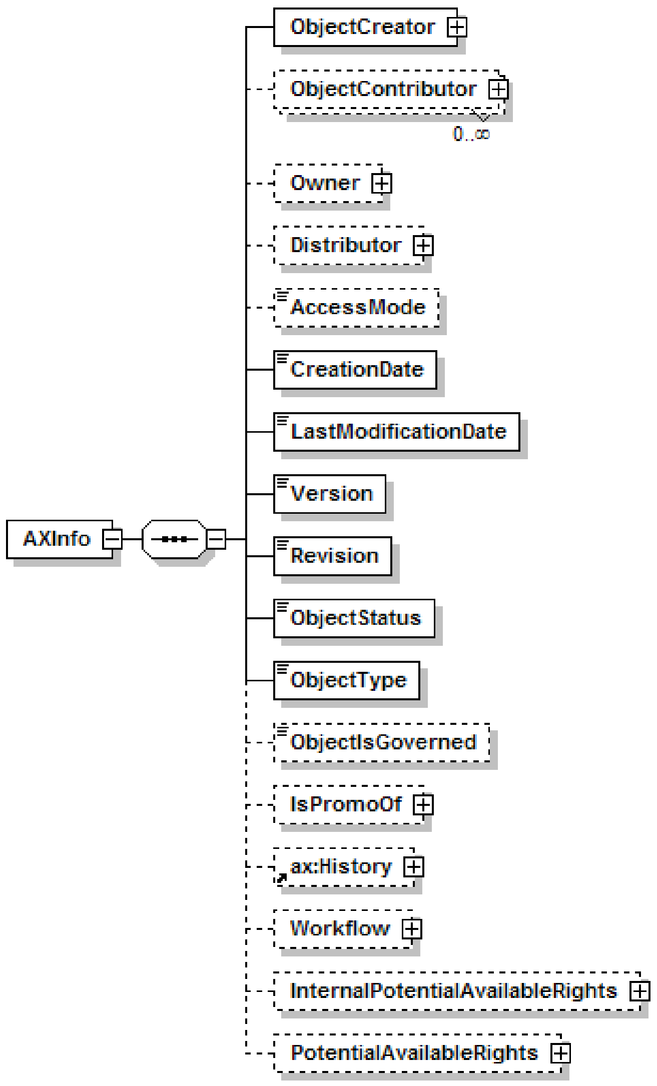
The AXInfo contains information to manage the object in its entire life-cycle, it contains

- Creator information (AXCID, Name, Company, URLs, …)
- Contributor information (AXCID, Name, Company, URLs, …)
- Distributor information (AXDID, Name, URLs, …)
- Access information (read only or read/write)
- Creation and modification times
- The History of the object (version/revision, commands performed on the object)
- The Workflow information, etc.
- Potential Available Rights (PAR) for the object and licensing information

In the following documentation of AXInfo schema is reported.

element **AXInfo**

diagram

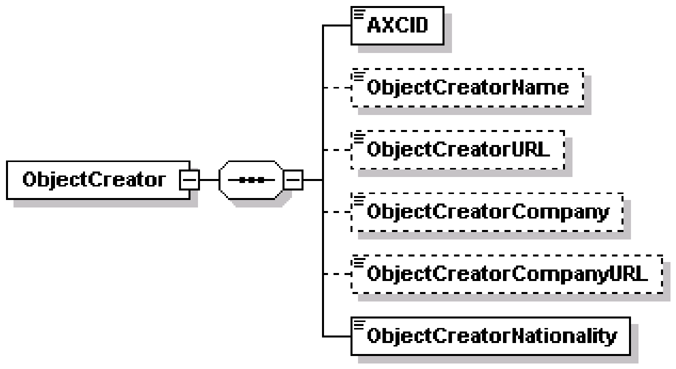


namespace	urn:axmedis:01
children	ObjectCreator ObjectContributor Owner Distributor AccessMode CreationDate LastModificationDate Version Revision ObjectStatus ObjectType ObjectIsGoverned IsPromoOf ax:History Workflow InternalPotentialAvailableRights PotentialAvailableRights

description	
constraints	if the AccessMode is missing it should be considered as readOnly

element **AXInfo/ObjectCreator**

diagram



namespace urn:axmedis:01

children [AXCID](#) [ObjectCreatorName](#) [ObjectCreatorURL](#) [ObjectCreatorCompany](#) [ObjectCreatorCompanyURL](#) [ObjectCreatorNationality](#)

description It contains information regarding the person who created the object

example
```
<ObjectCreator>
   <AXCID>938372671615274854959487 3723</AXCID>
   <ObjectCreatorName>John Doe</ObjectCreatorName>
   <ObjectCreatorURL>mailto:j.doe@video2.org</ObjectCreatorURL>
   <ObjectCreatorCompany>VIDEO2</ObjectCreatorCompany>
   <ObjectCreatorCompanyURL>http://www.video2.com</ObjectCreatorCompanyURL>
   <ObjectCreatorNationality>US</ObjectCreatorNationality>
</ObjectCreator>
```

element **AXInfo/ObjectCreator/AXCID**

diagram

namespace urn:axmedis:01

type **xs:string**

description It contains the AXMEDIS Creator Identifier

element **AXInfo/ObjectCreator/ObjectCreatorName**

diagram

namespace urn:axmedis:01

type **xs:string**

description personal name of the creator

constraints This tag should be removed when published on the P2P or on B2C

element **AXInfo/ObjectCreator/ObjectCreatorURL**

diagram

namespace urn:axmedis:01

type **xs:anyURI**

description URL associated to the object creator, it could be the email address

constraints	This tag should be removed when published on the P2P or on B2C

element **AXInfo/ObjectCreator/ObjectCreatorCompany**

diagram

namespace	urn:axmedis:01
type	**xs:string**
description	name of the company of the creator

element **AXInfo/ObjectCreator/ObjectCreatorCompanyURL**

diagram

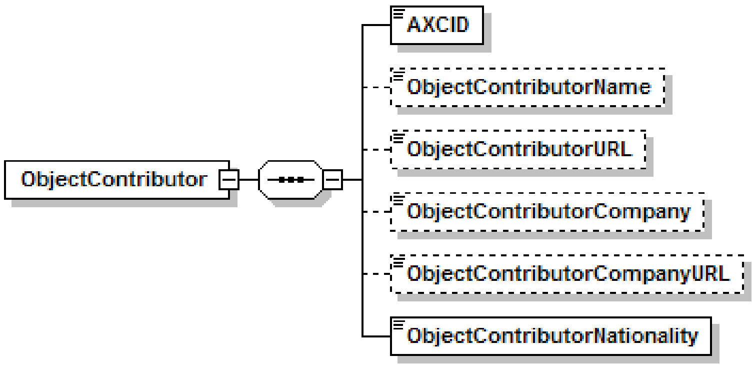
namespace	urn:axmedis:01
type	**xs:anyURI**
description	URL of the company of the creator

element **AXInfo/ObjectCreator/ObjectCreatorNationality**

diagram

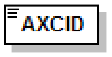
namespace	urn:axmedis:01
type	**xs:string**
description	nationality of the creator company using the ISO 3166 two letters code

element **AXInfo/ObjectContributor**

diagram


namespace	urn:axmedis:01
children	AXCID ObjectContributorName ObjectContributorURL ObjectContributorCompany ObjectContributorCompanyURL ObjectContributorNationality
description	**It contains information regarding the person contributing to the object realization**

element **AXInfo/ObjectContributor/AXCID**

diagram


namespace	urn:axmedis:01
type	**xs:string**
description	**It contains the AXMEDIS Creator Identifier**

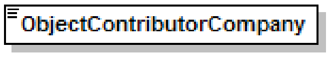
element **AXInfo/ObjectContributor/ObjectContributorName**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	personal name of the contributor
constraints	This tag should be removed when published on the P2P or on B2C

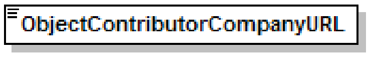
element **AXInfo/ObjectContributor/ObjectContributorURL**

diagram	
namespace	urn:axmedis:01
type	**xs:anyURI**
description	URL associated to the object contributor, it could be the email address
constraints	This tag should be removed when published on the P2P or on B2C

element **AXInfo/ObjectContributor/ObjectContributorCompany**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	name of the company of the contributor

element **AXInfo/ObjectContributor/ObjectContributorCompanyURL**

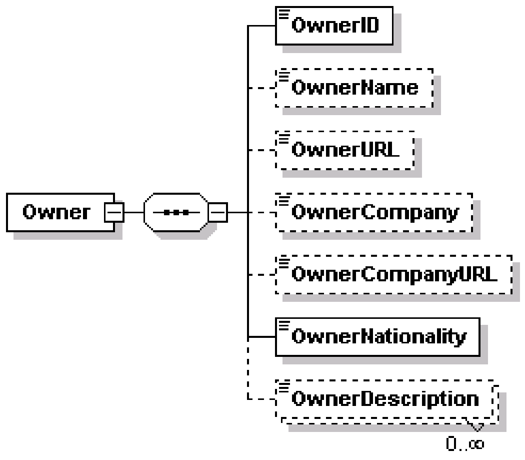
diagram	
namespace	urn:axmedis:01
type	**xs:anyURI**
description	URL of the company of the creator

element **AXInfo/ObjectContributor/ObjectContributorNationality**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	nationality of the contributor company using the ISO 3166 two letters code

element **AXInfo/Owner**

diagram



namespace	urn:axmedis:01
children	OwnerID OwnerName OwnerURL OwnerCompany OwnerCompanyURL OwnerNationality OwnerDescription
description	It contains information regarding the owner of the content, if not present the creator is the owner

example
```
<Owner>
  <OwnerID coding="SIAE">0038367292-292893-202383</OwnerID>
  <OwnerCompany>VIDEO Production</OwnerCompany>
  <OwnerCompanyURL>http://www.videoproduction.com</OwnerCompanyURL>
  <OwnerNationality>US</OwnerNationality>
</Owner>
```

element **AXInfo/Owner/OwnerID**

diagram

namespace urn:axmedis:01

type extension of **xs:string**

attributes

Name	Type	Use	Default	Fixed
coding	xs:string	required		

description identification code to identify the content owner, the coding attribute is used to state which coding scheme is used

example <OwnerID coding="**SIAE**">**10293834-236272-353**</OwnerID>

element **AXInfo/Owner/OwnerName**

diagram

namespace urn:axmedis:01

type **xs:string**

description name of the content owner

element **AXInfo/Owner/OwnerURL**

diagram

namespace urn:axmedis:01

type **xs:anyURI**

description the URL of the owner (website or email)

element **AXInfo/Owner/OwnerCompany**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	company name owning the content

element **AXInfo/Owner/OwnerCompanyURL**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	URL of the company owning the content (web site)

element **AXInfo/Owner/OwnerNationality**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	Nationality of the content owner encoded using ISO 3166 two letters code

element **AXInfo/Owner/OwnerDescription**

diagram	
namespace	urn:axmedis:01
type	extension of **xs:string**

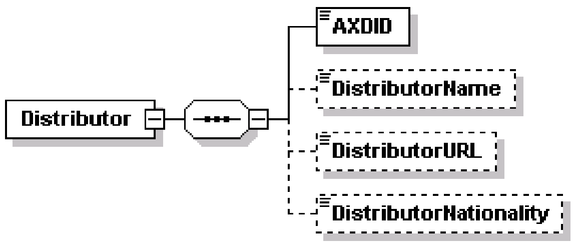
attributes

Name	Type	Use	Default	Fixed
lang	xs:string	required		

description A description of the owner, the *lang* attribute states the language used for the description

element **AXInfo/Distributor**

diagram



namespace	urn:axmedis:01
children	**AXDID** **DistributorName** **DistributorURL** **DistributorNationality**
description	It contains information about the Distributor that distributed the object, it will be present only in the B2C phase

element **AXInfo/Distributor/AXDID**

diagram	
namespace	urn:axmedis:01

type	**xs:string**
description	is the AXMEDIS Distributor Identifier

element **AXInfo/Distributor/DistributorName**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	name of the distributor

element **AXInfo/Distributor/DistributorURL**

diagram	
namespace	urn:axmedis:01
type	**xs:anyURI**
description	URL of the distributor (web site)

element **AXInfo/Distributor/DistributorNationality**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	Nationality of the distributor encoded using ISO 3166 two letters code

element **AXInfo/AccessMode**

diagram	
namespace	urn:axmedis:01
type	restriction of **xs:string**
facets	enumeration readOnly enumeration read_write
description	states if the object can be changed (read_write) or not (readOnly)
constraints	The AccessMode should be the same in all the AXInfos of a composite object, however in case they are missing or contradictory the one at the top level should be considered valid.

element **AXInfo/CreationDate**

diagram	
namespace	urn:axmedis:01
type	**xs:dateTime**
description	date and time of object creation

element **AXInfo/LastModificationDate**

diagram	
namespace	urn:axmedis:01
type	**xs:dateTime**

description	date and time of object modification

element **AXInfo/Version**

diagram	
namespace	urn:axmedis:01
type	**xs:nonNegativeInteger**
description	number of version of the object, it should be incremented each time the object is uploaded in the AXDB

element **AXInfo/Revision**

diagram	
namespace	urn:axmedis:01
type	**xs:nonNegativeInteger**
description	number of revision of the object, it should be incremented each time the object is saved to disk and it should return to 0 when uploaded in the AXDB

element **AXInfo/ObjectStatus**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	status of the object (e.g. in production, published, …)

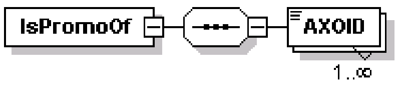
element **AXInfo/ObjectType**

diagram	
namespace	urn:axmedis:01
type	restriction of **xs:string**
facets	enumeration BASIC enumeration COMPOSITE
description	it states if the object is BASIC or COMPOSITE
constraints	in case the object is BASIC it should have the structure of an AXMEDIS Basic Object, and the structure of an AXMEDIS Composite Object for a COMPOSITE one. The value for this tag can be also derived from the object structure.

element **AXInfo/ObjectIsGoverned**

diagram	
namespace	urn:axmedis:01
type	**xs:boolean**
description	states if the object has a licence inside (true) or not (false)/

element **AXInfo/IsPromoOf**

diagram	
namespace	urn:axmedis:01

children	**AXOID**
description	contains a sequence of AXOIDs referring to objects for which this object is a promotional version

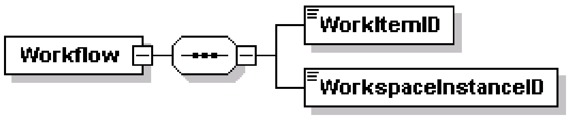
element **AXInfo/IsPromoOf/AXOID**

diagram

namespace	urn:axmedis:01
type	**xs:token**
description	an identifier of an AXMEDIS object for which the whole object is a promotional version

element **AXInfo/Workflow**

diagram



namespace	urn:axmedis:01
children	**WorkItemID** **WorkspaceInstanceID**
description	it contains information for the workflow management of the object

element **AXInfo/Workflow/WorkItemID**

diagram

namespace	urn:axmedis:01
type	**xs:string**
description	the identifier of the workitem

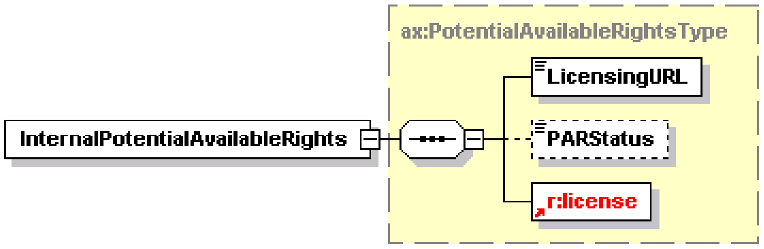
element **AXInfo/Workflow/WorkspaceInstanceID**

diagram

namespace	urn:axmedis:01
type	**xs:string**
description	Identifies the workspace instance

element **AXInfo/InternalPotentialAvailableRights**

diagram


ax:PotentialAvailableRightsType
InternalPotentialAvailableRights
LicensingURL
PARStatus
r:license

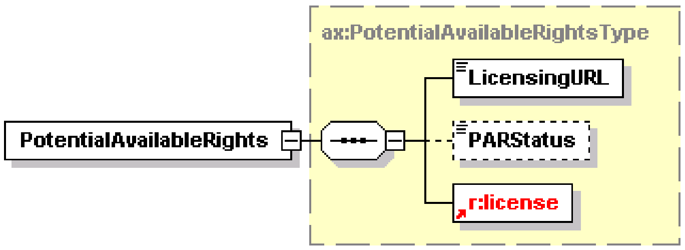
namespace	urn:axmedis:01
type	**ax:PotentialAvailableRightsType**
children	**LicensingURL** **PARStatus** **r:license**
description	contains all the rights really available on the object, not all these rights can be exploited by end users or distributors.
	the LicensingURL contains the URL to acquire a license for the object and PARStatus contains the status like to be

verified, verified, …

constraints the license is not a complete licence, it is used to contain only the grants but without the principal and the resource elements.

element **AXInfo/PotentialAvailableRights**

diagram



namespace urn:axmedis:01

type **ax:PotentialAvailableRightsType**

children **LicensingURL** **PARStatus** **r:license**

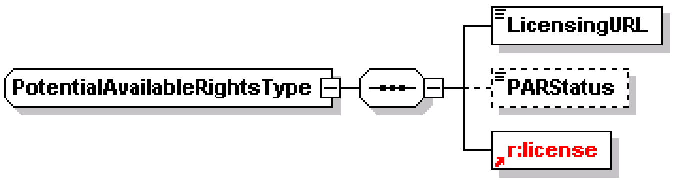
description contains the rights available outside the AXMEDIS Factory usually it is a subset of the InternalPotentialAvailableRights.

the LicensingURL contains the URL to acquire a license for the object and PARStatus contains the status like to be verified, verified, …

constraints the license is not a complete licence, it is used to contain only the grants but without the principal and the resource elements.

complexType **PotentialAvailableRightsType**

diagram



namespace urn:axmedis:01

children **LicensingURL** **PARStatus** **r:license**

description this type contains the information on the rights potentially available on the object, its status and the url to be used to acquire a real license

element **PotentialAvailableRightsType/LicensingURL**

diagram

namespace urn:axmedis:01

type **xs:anyURI**

description contains the URL to be used to acquire a licence for the object

element **PotentialAvailableRightsType/PARStatus**

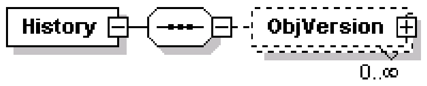
diagram

namespace urn:axmedis:01

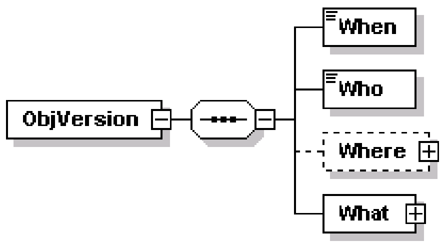
type **xs:string**

description contains the current status of the PAR like: to be verified, verified, …

element **History**

diagram	
namespace	urn:axmedis:01
children	ObjVersion
description	contains the history of the object

element **History/ObjVersion**

diagram	
namespace	urn:axmedis:01
children	When Who Where What
description	contains information on the history of a specific version of the object, the *number* attribute indicates the version number

attributes

Name	Type	Use	Default	Fixed
number	xs:nonNegativeInteger			

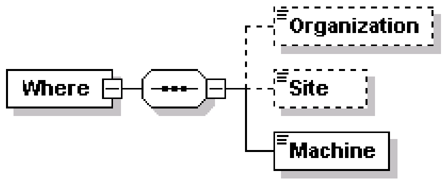
element **History/ObjVersion/When**

diagram	
namespace	urn:axmedis:01
type	**xs:dateTime**
description	contains the date & time when the version was uploaded on the AXDB

element **History/ObjVersion/Who**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	contains the name of the person who uploaded he object on the AXDB

element **History/ObjVersion/Where**

diagram	
namespace	urn:axmedis:01
children	Organization Site Machine
description	contains the indication of the location where the upload was performed

element **History/ObjVersion/Where/Organization**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	contains the indication of the Organization where the upload was performed

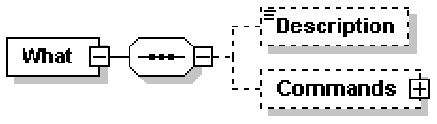
element **History/ObjVersion/Where/Site**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	contains the indication of the site where the upload was performed

element **History/ObjVersion/Where/Machine**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	contains the indication of the machine where the upload was performed

element **History/ObjVersion/What**

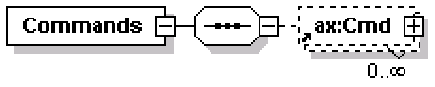
diagram	
namespace	urn:axmedis:01
children	**Description** **Commands**
description	contains what have been performed on the object as a textual description and as the list of commands performed.

element **History/ObjVersion/What/Description**

diagram	
namespace	urn:axmedis:01
type	extension of **xs:string**
attributes	
description	contains textual description of what have been done on the object for the specific object version

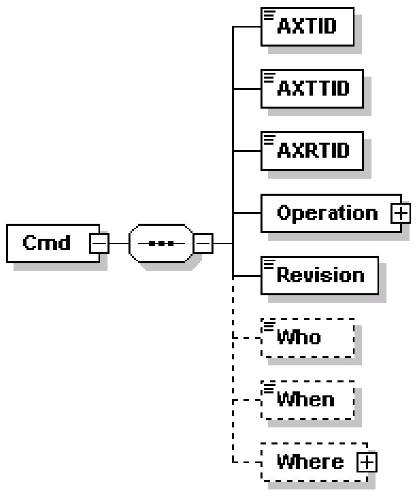
Name	Type	Use	Default	Fixed
lang	xs:string	optional		

element **History/ObjVersion/What/Commands**

diagram	
namespace	urn:axmedis:01
children	**ax:Cmd**
description	contains the commands performed on the object.

element **Cmd**

diagram



namespace	urn:axmedis:01
children	**AXTID AXTTID AXRTID Operation Revision Who When Where**
description	contains information regarding a command performed on the object

element **Cmd/AXTID**

diagram

namespace	urn:axmedis:01
type	**xs:string**
description	contains the AXMEDIS Tool ID identifying the tool used to perform the command

element **Cmd/AXTTID**

diagram

namespace	urn:axmedis:01
type	**xs:string**
description	contains the AXMEDIS Tool Type ID identifying the type of tool used to perform the command

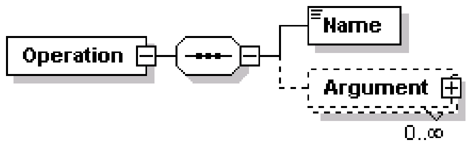
element **Cmd/AXRTID**

diagram

namespace	urn:axmedis:01
type	**xs:string**
description	contains the AXMEDIS Real Tool ID identifying the tool instance used to produce the object

element **Cmd/Operation**

diagram



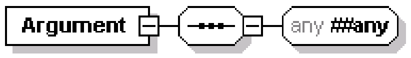
namespace	urn:axmedis:01
children	**Name** **Argument**
description	contains the operation performed to the object

element **Cmd/Operation/Name**

diagram

namespace	urn:axmedis:01
type	**xs:string**
description	contains the name of the operation performed on the object

element **Cmd/Operation/Argument**

diagram 

namespace	urn:axmedis:01
description	contains an argument for the operation, it can be any xml tag.

element **Cmd/Revision**

diagram

namespace	urn:axmedis:01
type	**xs:nonNegativeInteger**
description	contains the revision number to which the command contributes

element **Cmd/Who**

diagram

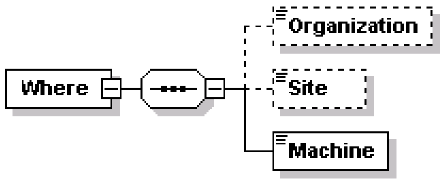
namespace	urn:axmedis:01
type	**xs:string**
description	contains information regarding who performed the operation

element **Cmd/When**

diagram

namespace	urn:axmedis:01
type	**xs:dateTime**
description	contains when (date & time) the operation was performed

element **Cmd/Where**

diagram 

namespace	urn:axmedis:01

children	**Organization** **Site** **Machine**
description	contains the location where the operation was performed

element **Cmd/Where/Organization**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	contains the Organization where the operation was performed

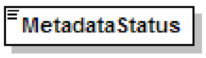
element **Cmd/Where/Site**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	contains the site where the operation was performed

element **Cmd/Where/Machine**

diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	contains the identifier of the machine where the operation was performed

element **MetadataStatus**

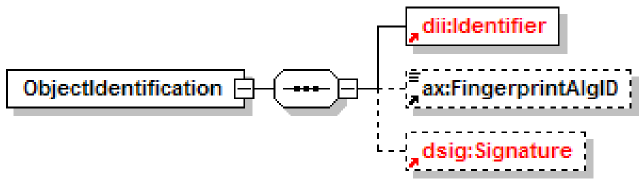
diagram	
namespace	urn:axmedis:01
type	**xs:string**
description	contains the editorial status of the metadata descriptor (e.g. to be completed, verified, …)

element **MetadataVisibility**

diagram	
namespace	urn:axmedis:01
type	restriction of **xs:string**
facets	enumeration public enumeration private
description	contains the visibility of metadata when the object is protected, private means that the metadata should not be accessible in clear, public otherwise.

element **ObjectIdentification**

diagram



namespace	urn:axmedis:01
children	**dii:Identifier** **ax:FingerprintAlgID** **dsig:Signature**
description	contains the Identifier element with the AXOID, the fingerprint algorithm to be used for object recognition and the signature for the whole object (protected)

The following is the complete textual description of the AXInfo Schema:

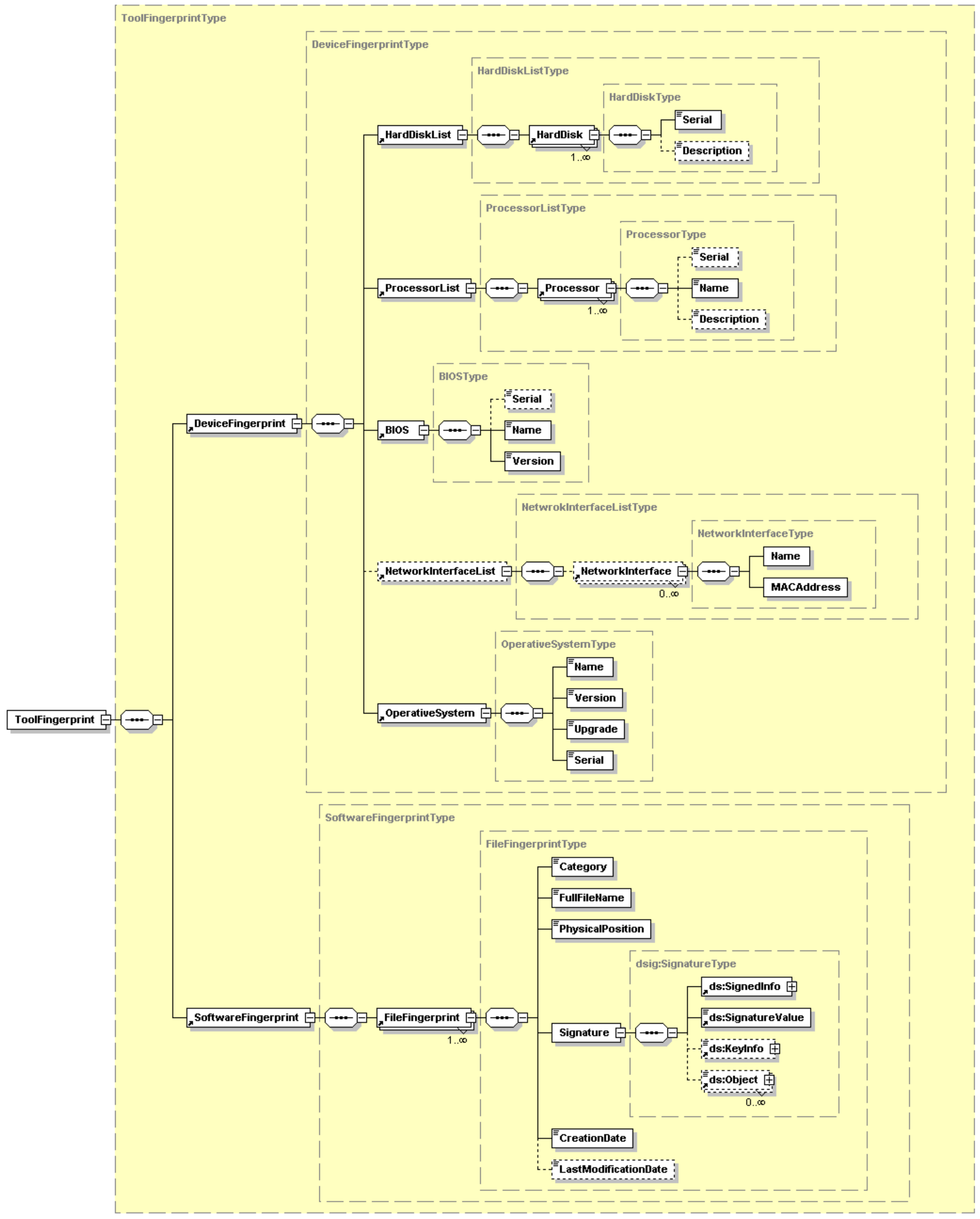
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:axmedis:01" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:dii="urn:mpeg:mpeg21:2002:01-DII-NS" xmlns:r="urn:mpeg:mpeg21:2003:01-REL-R-NS"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:ax="urn:axmedis:01" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:import namespace="urn:mpeg:mpeg21:2002:02-DIDL-NS" schemaLocation="mpeg21xmlschemas\DIDL.xsd"/>
    <xs:import namespace="urn:mpeg:mpeg21:2004:01-IPMPDIDL-NS" schemaLocation="mpeg21xmlschemas\ipmpDIDL.xsd"/>
    <xs:import namespace="urn:mpeg:mpeg21:2002:01-DII-NS" schemaLocation="mpeg21xmlschemas\dii.xsd"/>
    <xs:import namespace="urn:mpeg:mpeg21:2003:01-REL-R-NS" schemaLocation="mpeg21xmlschemas\rel-r.xsd"/>
    <xs:import namespace="urn:mpeg:mpeg21:2003:01-REL-MX-NS" schemaLocation="mpeg21xmlschemas\rel-mx.xsd"/>
    <xs:import namespace="urn:mpeg:mpeg21:2003:01-REL-SX-NS" schemaLocation="mpeg21xmlschemas\rel-sx.xsd"/>
    <xs:import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="mpeg21xmlschemas\xmldsig-core-
schema.xsd"/>
    <xs:element name="AXInfo">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="ObjectCreator">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="AXCID" type="xs:string"/>
                            <xs:element name="ObjectCreatorName" type="xs:string" minOccurs="0"/>
                            <xs:element name="ObjectCreatorURL" type="xs:anyURI" minOccurs="0"/>
                            <xs:element name="ObjectCreatorCompany" type="xs:string" minOccurs="0"/>
                            <xs:element name="ObjectCreatorCompanyURL" type="xs:anyURI" minOccurs="0"/>
                            <xs:element name="ObjectCreatorNationality" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="ObjectContributor" minOccurs="0" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="AXCID" type="xs:string"/>
                            <xs:element name="ObjectContributorName" type="xs:string" minOccurs="0"/>
                            <xs:element name="ObjectContributorURL" type="xs:anyURI" minOccurs="0"/>
                            <xs:element name="ObjectContributorCompany" type="xs:string" minOccurs="0"/>
                            <xs:element name="ObjectContributorCompanyURL" type="xs:anyURI" minOccurs="0"/>
                            <xs:element name="ObjectContributorNationality" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="Owner" minOccurs="0">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="OwnerID">
                                <xs:complexType>
                                    <xs:simpleContent>
                                        <xs:extension base="xs:string">
                                            <xs:attribute name="coding" type="xs:string" use="required"/>
                                        </xs:extension>
                                    </xs:simpleContent>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="OwnerName" type="xs:string" minOccurs="0"/>
            <xs:element name="OwnerURL" type="xs:anyURI" minOccurs="0"/>
            <xs:element name="OwnerCompany" type="xs:string" minOccurs="0"/>
            <xs:element name="OwnerCompanyURL" type="xs:string" minOccurs="0"/>
            <xs:element name="OwnerNationality" type="xs:string"/>
            <xs:element name="OwnerDescription" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:string">
                    <xs:attribute name="lang" type="xs:string" use="required"/>
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Distributor" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="AXDID" type="xs:string"/>
            <xs:element name="DistributorName" type="xs:string" minOccurs="0"/>
            <xs:element name="DistributorURL" type="xs:anyURI" minOccurs="0"/>
            <xs:element name="DistributorNationality" type="xs:string" minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="AccessMode" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="readOnly"/>
            <xs:enumeration value="read_write"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="CreationDate" type="xs:dateTime"/>
      <xs:element name="LastModificationDate" type="xs:dateTime"/>
      <xs:element name="Version" type="xs:nonNegativeInteger"/>
      <xs:element name="Revision" type="xs:nonNegativeInteger"/>
      <xs:element name="ObjectStatus" type="xs:string"/>
      <xs:element name="ObjectType">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="BASIC"/>
            <xs:enumeration value="COMPOSITE"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="ObjectIsGoverned" type="xs:boolean" minOccurs="0"/>
      <xs:element name="IsPromoOf" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="AXOID" type="xs:token" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element ref="ax:History" minOccurs="0"/>
      <xs:element name="Workflow" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="WorkItemID" type="xs:string"/>
            <xs:element name="WorkspaceInstanceID" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="InternalPotentialAvailableRights" type="ax:PotentialAvailableRightsType" minOccurs="0"/>
      <xs:element name="PotentialAvailableRights" type="ax:PotentialAvailableRightsType" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Cmd">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="AXTID" type="xs:string"/>
      <xs:element name="AXTTID" type="xs:string"/>
      <xs:element name="AXRTID" type="xs:string"/>
      <xs:element name="Operation">
```

```
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Name" type="xs:string"/>
                        <xs:element name="Argument" minOccurs="0" maxOccurs="unbounded">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:any namespace="##any"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Revision" type="xs:nonNegativeInteger"/>
            <xs:element name="Who" type="xs:string" minOccurs="0"/>
            <xs:element name="When" type="xs:dateTime" minOccurs="0"/>
            <xs:element name="Where" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Organization" type="xs:string" minOccurs="0"/>
                        <xs:element name="Site" type="xs:string" minOccurs="0"/>
                        <xs:element name="Machine" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="History">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="ObjVersion" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="When" type="xs:dateTime"/>
                        <xs:element name="Who" type="xs:string"/>
                        <xs:element name="Where" minOccurs="0">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="Organization" type="xs:string" minOccurs="0"/>
                                    <xs:element name="Site" type="xs:string" minOccurs="0"/>
                                    <xs:element name="Machine" type="xs:string"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="What">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="Description" minOccurs="0">
                                        <xs:complexType>
                                            <xs:simpleContent>
                                                <xs:extension base="xs:string">
                                                    <xs:attribute name="lang" type="xs:string" use="optional"/>
                                                </xs:extension>
                                            </xs:simpleContent>
                                        </xs:complexType>
                                    </xs:element>
                                    <xs:element name="Commands" minOccurs="0">
                                        <xs:complexType>
                                            <xs:sequence>
                                                <xs:element ref="ax:Cmd" minOccurs="0" maxOccurs="unbounded"/>
                                            </xs:sequence>
                                        </xs:complexType>
                                    </xs:element>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                    <xs:attribute name="number" type="xs:nonNegativeInteger"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:complexType name="PotentialAvailableRightsType">
    <xs:sequence>
```

```
            <xs:element name="LicensingURL" type="xs:anyURI"/>
            <xs:element name="PARStatus" type="xs:string" minOccurs="0"/>
            <xs:element ref="r:license"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="ObjectIdentification">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="dii:Identifier"/>
                <xs:element ref="ax:FingerprintAlgID" minOccurs="0"/>
                <xs:element ref="dsig:Signature" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="MetadataStatus" type="xs:string"/>
    <xs:element name="MetadataVisibility">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="public"/>
                <xs:enumeration value="private"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="FingerprintAlgID" type="xs:string"/>
</xs:schema>
```

18 AXMEDIS Tool Fingerprint (DSI, FUPF)

Tool fingerprint is decribed in the sections 8.1.1.1, it contains information on the hardware (i.e. the personal computer) and on the software. It has been created to uniquely identify an installation of a given AXMEDIS-complaiant application on a given device and to allow detection of software/hardware changes. Fingerprint is stored and transmitted as XML file/message with the following schema:



Toolfingerprint XML schema.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.axmedis.org/tool-fp" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.axmedis.org/tool-fp" elementFormDefault="qualified"
attributeFormDefault="unqualified">
	<xs:import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="xmldsig-core-schema.xsd"/>
	<xs:element name="ToolFingerprint" type="ToolFingerprintType"/>
	<xs:complexType name="ToolFingerprintType">
		<xs:sequence>
			<xs:element ref="DeviceFingerprint"/>
			<xs:element ref="SoftwareFingerprint"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="DeviceFingerprint" type="DeviceFingerprintType"/>
	<xs:complexType name="DeviceFingerprintType">
		<xs:sequence>
			<xs:element ref="HardDiskList"/>
			<xs:element ref="ProcessorList"/>
			<xs:element ref="BIOS"/>
			<xs:element ref="NetworkInterfaceList" minOccurs="0"/>
			<xs:element ref="OperativeSystem"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="HardDiskList" type="HardDiskListType"/>
	<xs:complexType name="HardDiskListType">
		<xs:sequence>
			<xs:element ref="HardDisk" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="HardDisk" type="HardDiskType"/>
	<xs:complexType name="HardDiskType">
		<xs:sequence>
			<xs:element name="Serial" type="xs:string"/>
			<xs:element name="Description" type="xs:string" minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="ProcessorList" type="ProcessorListType"/>
	<xs:complexType name="ProcessorListType">
		<xs:sequence>
			<xs:element ref="Processor" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="Processor">
		<xs:complexType>
			<xs:complexContent>
				<xs:extension base="ProcessorType"/>
			</xs:complexContent>
		</xs:complexType>
	</xs:element>
	<xs:complexType name="ProcessorType">
		<xs:sequence>
			<xs:element name="Serial" type="xs:string" minOccurs="0"/>
			<xs:element name="Name" type="xs:string"/>
			<xs:element name="Description" type="xs:string" minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="BIOS" type="BIOSType"/>
	<xs:complexType name="BIOSType">
		<xs:sequence>
			<xs:element name="Serial" type="xs:string" minOccurs="0"/>
			<xs:element name="Name" type="xs:string"/>
			<xs:element name="Version" type="xs:string"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="NetworkInterfaceList" type="NetwrokInterfaceListType"/>
	<xs:complexType name="NetwrokInterfaceListType">
		<xs:sequence>
			<xs:element ref="NetworkInterface" minOccurs="0" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="NetworkInterface" type="NetworkInterfaceType"/>
```

```
    <xs:complexType name="NetworkInterfaceType">
        <xs:sequence>
            <xs:element name="Name"/>
            <xs:element name="MACAddress"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="OperativeSystem" type="OperativeSystemType"/>
    <xs:complexType name="OperativeSystemType">
        <xs:sequence>
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Version" type="xs:string"/>
            <xs:element name="Upgrade" type="xs:string"/>
            <xs:element name="Serial" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="SoftwareFingerprint" type="SoftwareFingerprintType"/>
    <xs:complexType name="SoftwareFingerprintType">
        <xs:sequence>
            <xs:element ref="FileFingerprint" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:element name="FileFingerprint" type="FileFingerprintType"/>
    <xs:complexType name="FileFingerprintType">
        <xs:sequence>
            <xs:element name="Category">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="CONTAIN_AXOM"/>
                        <xs:enumeration value="PLUG_IN"/>
                        <xs:enumeration value="CONFIGURATION"/>
                        <xs:enumeration value="SECURE_CACHE"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="FullFileName" type="xs:anyURI"/>
            <xs:element name="PhysicalPosition" type="xs:string"/>
            <xs:element name="Signature" type="dsig:SignatureType"/>
            <xs:element name="CreationDate" type="xs:dateTime"/>
            <xs:element name="LastModificationDate" type="xs:dateTime" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

As stated before, the information contained in the fingerprint are mainly information for the identification of the tool, however profile-related information can be estimated in asimila way and merged with those above reported. Samples of profile information are the following:

- Which kind of content it may manage or not, e.g. it cannot load PDF, it can load PS
- Resolution of the screen device
- Power of the device
- Presence of some basic adaptation tools or their absence
- Print capabilities or not
- Audio capabilities or not
- Video streaming capabilities or not
- Burning ROM capabilities or not
- Network connection speed
- Network connection type, e.g. permanent or irregular
- etc….

19 AXMEDIS Protection Info (DSI, FHGIGD)

Protection information is formatted as stated in MPEG-21 Part 4 IPMP standard. The syntax and semantics is still under discussion, the actual state of the standard is contained in the output document w7717 of the 74th MPEG meeting (see http://mpeg.nist.gov/).

MPEG-21 Part 4 divides protection information into two XML schemas:

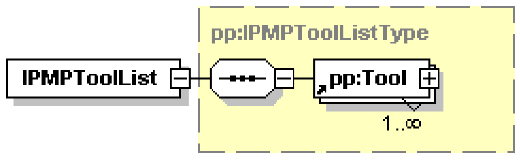
- one is used to declare the list of needed protection tools (or commands as defined in this section) to unprotect the whole digital item;
- the other is used to describe, for each protected element, how to use those tools (e.g. the execution order, keys, initialization parameters, etc…) to unprotect a specific element.

The former part of protection information (i.e. the list of all needed tools) should not only contain the necessary tools to unprotect the "first level" of protected element, it should contain also the required tools to correctly manage all nested levels of protected elements. In that way, looking at the tool list declaration it will be possible to immediately decide whether an AXMEDIS Tool is capable to completely "consume" an object.

20 Protection Tool description (DSI)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.axmedis.org/ipmp-tool-schema" xmlns:prm="http://www.axmedis.org/parameter"
xmlns:ipmpinfo="urn:mpeg:mpeg21:2004:01-IPMPINFO-NS" xmlns:pp="http://www.axmedis.org/ipmp-tool-schema"
xmlns:pin="http://www.axmedis.org/plugin-schema" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified" version="0.1">
    <xs:import namespace="http://www.axmedis.org/plugin-schema" schemaLocation="plugin-schema.xsd"/>
    <xs:import namespace="urn:mpeg:mpeg21:2004:01-IPMPINFO-NS"
schemaLocation="mpeg21xmlschemas/ipmpinfo.xsd"/>
    <xs:import namespace="http://www.axmedis.org/parameter" schemaLocation="param-schema.xsd"/>
    <xs:element name="IPMPToolList" type="pp:IPMPToolListType" substitutionGroup="pin:SpecificDescriptor"/>
    <xs:complexType name="IPMPToolListType">
        <xs:complexContent>
            <xs:extension base="pin:SpecificDescriptorType">
                <xs:sequence>
                    <xs:element ref="pp:Tool" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:element name="Tool" type="pp:ToolType"/>
    <xs:complexType name="ToolType">
        <xs:sequence>
            <xs:element ref="ipmpinfo:IPMPToolID"/>
            <xs:element ref="pp:Name"/>
            <xs:element ref="pp:Description"/>
            <xs:element ref="prm:ParameterList" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute ref="pp:type" use="required"/>
    </xs:complexType>
    <xs:attribute name="type">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="authoring"/>
                <xs:enumeration value="playing"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Description" type="xs:string"/>
    <xs:element name="AllowedValues" type="pp:AllowedValuesType" substitutionGroup="prm:Constraint"/>
    <xs:complexType name="AllowedValuesType">
        <xs:complexContent>
            <xs:restriction base="prm:ConstraintType">
                <xs:sequence>
                    <xs:element ref="pp:Value"/>
                </xs:sequence>
            </xs:restriction>
        </xs:complexContent>
    </xs:complexType>
    <xs:element name="Value" type="xs:anyType"/>
</xs:schema>
```

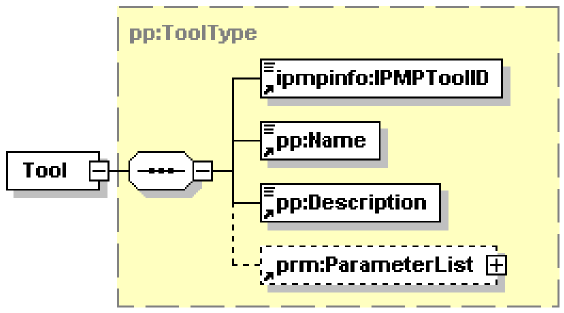
element **IPMPToolList**

diagram	
namespace	http://www.axmedis.org/ipmp-tool-schema
type	**pp:IPMPToolListType**
children	**pp:Tool**
source	`<xs:element name="IPMPToolList" type="pp:IPMPToolListType" substitutionGroup="pin:SpecificDescriptor"/>`
description	This element is the root element for the description of all protection tools exposed by an ipmp plug-in. It is a substitution

of the abstract element *SpecificDescriptor* described in the section “Formal description of format – Content Processing Plug-ins specific description” in DE3.1.2.2.4

element **Tool**

diagram



namespace http://www.axmedis.org/ipmp-tool-schema

type **pp:ToolType**

children **ipmpinfo:IPMPToolID** **pp:Name** **pp:Description** **prm:ParameterList**

used by complexType **IPMPToolListType**

attributes

Name	Type	Use	Default	Fixed	Annotation
pp:type	xs:string	required			

description This element describes a Protection Tool contained in the plug-in. It mainly provides:

- the identifier of the tool using the element *IPMPToolID* which is defined in the MPEG-21 IPMP standard
- a short name for the tool
- a human-readable description of the tool
- a list of parameters in order to initialize the tool

The attribute *type* can be: “authoring”,”playing”. It is “authoring” if both encoding and decoding capabilities are available. Otherwise, it is “playing” in order to warn the only the decoding function is provided.

element **Name**

diagram

namespace http://www.axmedis.org/ipmp-tool-schema

type **xs:string**

used by complexType **ToolType**

description This element conatins a string representing the name of the tool.

element **Description**

diagram

namespace http://www.axmedis.org/ipmp-tool-schema

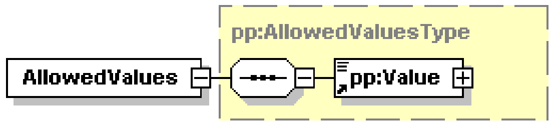
type **xs:string**

used by complexType **ToolType**

description This element conatins a string representing the description of the tool.

element **AllowedValues**

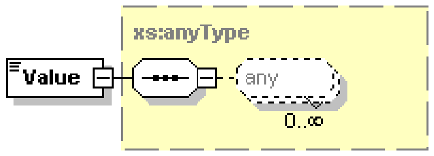
diagram



namespace	http://www.axmedis.org/ipmp-tool-schema
type	**pp:AllowedValuesType**
children	**pp:Value**
source	<xs:element name="AllowedValues" type="pp:AllowedValuesType" substitutionGroup="prm:Constraint"/>
description	This element represent a constraint on the values a given parameter can be set. This element is a substitution group of the *Constrain* element which is explained in the section “Formal description of format – Parameter description” in DE3.1.2.2.4

element **Value**

diagram



namespace	http://www.axmedis.org/ipmp-tool-schema
type	**xs:anyType**
used by	complexType **AllowedValuesType**
description	This element represents an allowed value for a given parameter. Please notice that it can contain any value type since the value type dependes on the type of the parameter itself.

21 Rights and Enforcement (FUPF, DSI, all)

Protection issues not only apply to modifications of the content in order to protect it against non-authorised use, but also imply that, when a content is governed by means of a license, rights are enforced and only authorised actions are allowed.
In this sense, there has been some discussion inside AXMEDIS consortium in order to enforce rights governing a content from a technical point of view.
The result of this discussion is shown in the rest of this section, together with other aspects regarding rights and AXMEDIS.

21.1 Description of typical content manipulation

In this section a relevant selection of content manipulation examples are presented. The scenario based approach has been chosen, for it has provoked the reasoning among action semantics, thus determining the corresponding set of rights to be owned.

21.1.1 General issue on distinguish Adapt and Modify intents

It is worth to point out that, using the MPEG-21 REL standard, some actions can be authorized using different rights. For example, the action of adding an element to a given DI can be authorized using either enlarge or enhance rights. These two rights respectively differ on whether the original object is effectively modified or a new modified object is derived from the original one. The same problem arises with other rights. In order to face this ambiguity, some technical issues have to be considered:

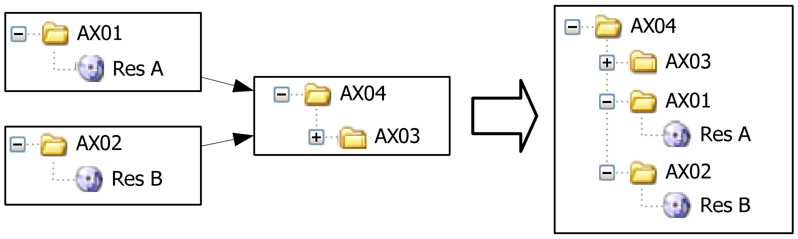
- Whether considering both adapt and modify rights or not. Both are needed, as their semantics are different. Adapt means that an existing resource is changed transiently to derive a new resource and Modify means the resource is altered, not deriving a new one from it, but maintaining modifications inside the resource itself
- Which rights should be requested to the PMS in ambiguous cases
- How to enforce rights in ambiguous cases

A solution in order to verify the correct rights during content manipulation is to ask to the Consumer what he wants to do with the Content, e.g. create a new Object or modify the original one. This question has to be done when requesting a license for manipulating the content.

21.2 Examples of AXMEDIS Object manipulation

This section describes different manipulations done over AXMEDIS objects. These manipulations are related to different MPEG-21 RDD rights. The rights associated are described in each manipulation shown. For some of the scenarios, it is also describe how a rights expression allowing each action should, indicating in each case its structure and the rights granted.

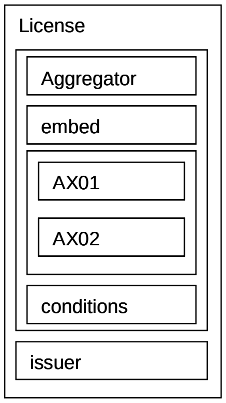
21.2.1 Adding to root level



Adding resources at root level

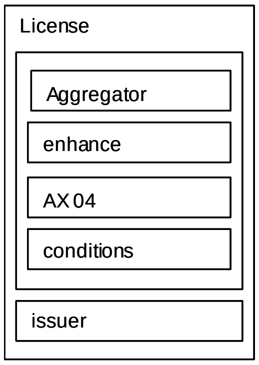
To determine the rights expressions needed in this process, we will consider the use case sketched in the above figure, where a user tries to aggregate two objects, AX01 and AX02, to the digital object AX04. In this scenario, the licenses governing these objects shall grant the user the right to modify or adapt the digital object (AX04) embedding within it AX01 and AX02.
Then, the user shall obtain a license that grants him permissions to embed AX01 and AX02 and to adapt or modify AX04 by adding to it. These licenses will be restricted to certain constraints usually referred to the aggregation process and to the resultant digital object.
Next figure shows the license issued by the owner or distributor of AX01 and AX02 to the aggregator. This license is formed by the following elements: the identification of the aggregator that is the principal of the

license, the **embed** right, the resources Res A and Res B that will be used for aggregation and the conditions that must be fulfilled to perform such aggregation. Finally, the issuer element identifies the manufacturer of both resources.



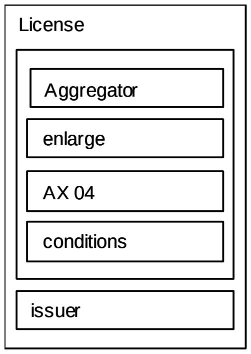
License for allowing objects embedding

This license uses the *embed* right specified in MPEG-21 RDD and defined as the right to put a Resource into another Resource.
On the other hand, the user also must have permissions of modification or adaptation over the digital object AX04.
If the user chooses the adaptation option two different digital objects will exist as result of the addition process, the original one in unchanged form and the newly made. MPEG-21 RDD defines the term *enhance* as the right “to derive a new resource which is larger than its source”. This right permits the licensee to change the original digital object by adding to it, for example embedding other resources. These changes are made temporarily to the original object, but they are not saved in the original object at the end of the process. Then, at the end of the process a new object is generated with such changes.
Next figure shows the MPEG-21 REL license that will grant to the aggregator permissions to enhance the digital object identified as AX04 with some restrictions specified within the conditions element of this license.



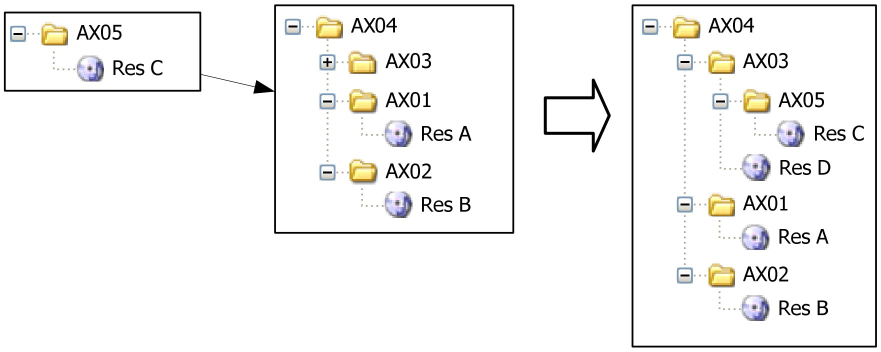
Adaptation choice - AX04 enhancing license

If the user chooses the modification option, only the modified object will be preserved. For this purpose, MPEG-21 RDD defines the term *enlarge* as the right “to Modify a Resource by adding to it”.
Next figure presents the MPEG-21 REL license that will grant to the aggregator permissions to enlarge the digital object identified as AX04 with some restrictions specified within the conditions element of this license. Then, he can add new material, including the embedding of other resources or elements, for example AX01, but not the changing or removal of existing elements of the original digital object.



Modification choice – AX04 enlarging license

21.2.2 Adding to nested level



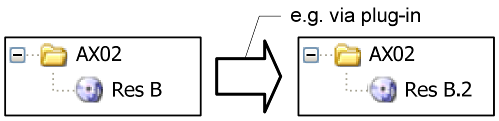
Adding resources at nested level

Standardized RDD terms associated:
Embed: To put a Resource into another Resource.

Scope of Embed
The Resource into which a Resource is Embedded can be pre-existing or can be created by the act of combining the EmbeddedResource with one or more others. Embed refers only to the embedding of an existing Resource in another: if a "copy" of an existing Resource is to be created and Embedded in another, then both Adapt and Embed would be used.

In nested object scenario, protection processor has to request the authorization of modifying all the nested objects involved in the modification.

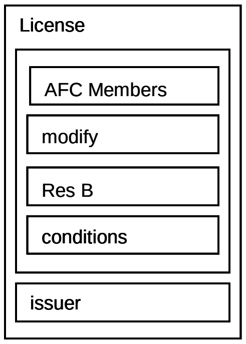
21.2.3 Transformation of basic objects



Modifying a resource, maintaining the changes

Standardized RDD terms associated:
Modify: To Change a Resource, preserving the alterations made.
In this scenario the licensee shall issue a license to the user that grants him the right to transform the asset Res B. MPEG-21 RDD defines the term *modify* as the right "to change a resource preserving the alterations made".

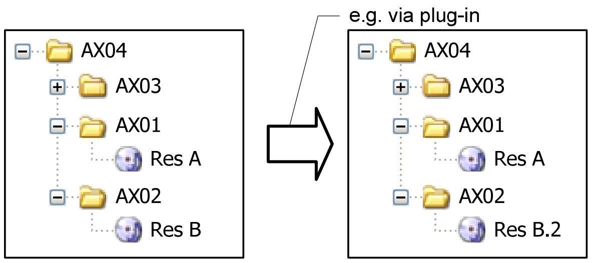
Then, the MPEG-21 license results as follows: the principal contains the identification of the user that can modify (right element) the AX02 (resource element) if certain conditions are previously fulfilled. Next figure shows the structure of the basic transformation license.



Basic transformation license example

In use cases where content is transformed, it is important to determine the restrictions that the licensee will determine regarding to the transformation of its digital content. On the other hand, it is also important to take into account that if the original object will be preserved then the MPEG-21 RDD term used as right in the transformation license shall be the adapt right.
MPEG-21 RDD defines the term *adapt* as the right "To ChangeTransiently an existing Resource to Derive a new Resource".

21.2.4 Transformation of objects in a composition

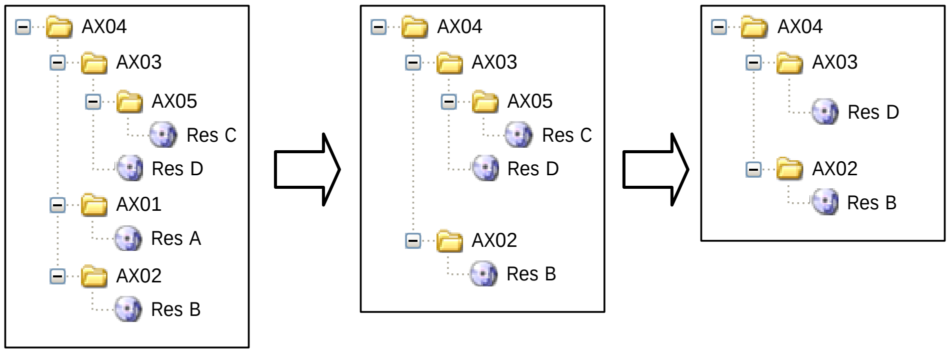


Modifying a composite object

Standardized RDD terms associated:
Modify: To Change a Resource, preserving the alterations made.

In order to modify a Resource the User has to have the rights related to the Resource and the Parent Objects.

21.2.5 Deletion of objects from a composition (from root level or nested)

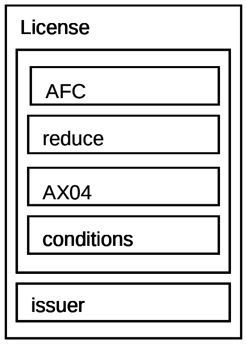


Deleting resources from a composite object

Standardized RDD terms associated:
Reduce: To Modify a Resource by taking away from it.

Scope of Reduce
With Reduce, a single Resource is preserved at the end of the process. Changes can include only the removal of existing elements of the original Resource.

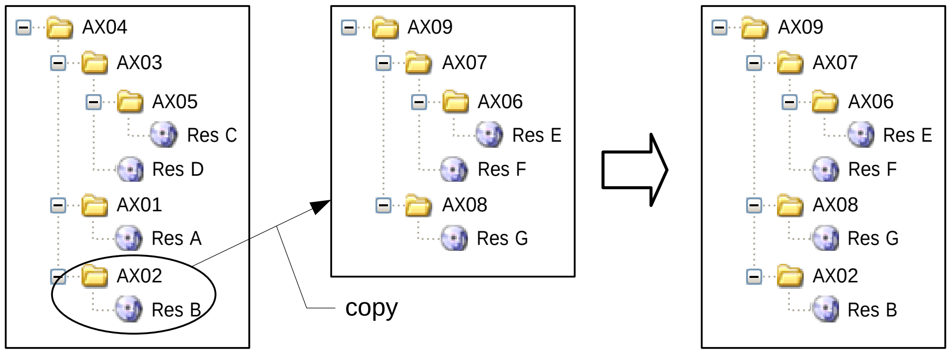
In this use case the user shall obtain a license (see next figure) that grants him permissions to delete digital objects from a composition.
MPEG-21 RDD defines the term *reduce* as the right “to Modify a Resource by taking away from it”. The change that can be performed when exercising reduce right is the removal of existing elements from the original digital object. At the end of the process only the modified resource is preserved.



Deletion license example

When granting *reduce* right, the licensee must consider if he gives permissions to remove all the elements or only a subset of them. If the licensee only grants permissions of deletion of specific elements from the original digital object, these constraints must be specified within the license as conditions.
On the other hand, in some cases the user also wants to destroy the elements taken away from the original digital object. Then, he must posses a license that grants him the right to delete these elements. MPEG-21 RDD defines *delete* term as the right “to Destroy a Digital Resource”. This right forms part of the MPEG-21 REL multimedia extension.

21.2.6 Copying of objects from a composition to another



Copy resources from a composite object to another

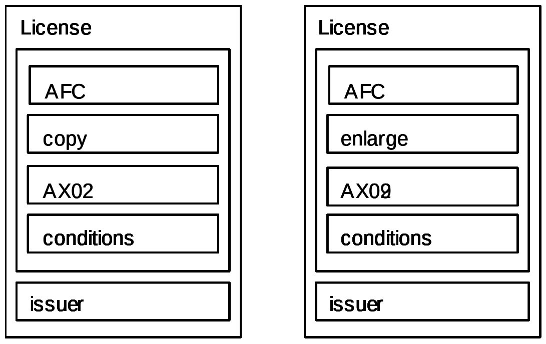
Copy AX02 from Composition 1 and Enlarge Composition2 with AX02.
In this scenario, the user shall have a license that grants him to copy AX02 from Composition 1 and to enlarge Composition2 with AX02.

Standardized RDD terms associated:
Copy is not defined in the RDD, and then a new term shall be added as a specialization of Adapt that is defined in RDD as the action to ChangeTransiently an existing Resource to Derive a new Resource.
Enlarge is defined in the RDD as the action to Modify a Resource by adding to it.

Scope of Enlarge
With Enlarge, a single Resource is preserved at the end of the process. Changes can include the addition of new material, including the Embedding of other Resources, but not the changing or removal of existing elements of the original Resource.

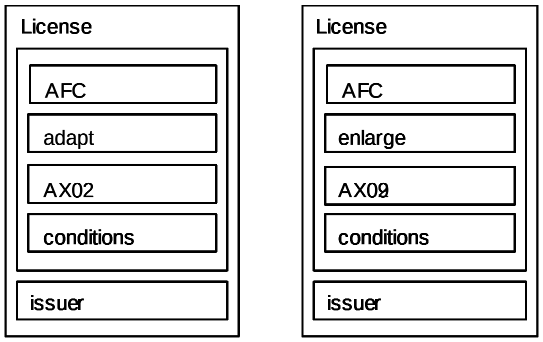
New REL Elements:
In the base profile the element bpx:governedCopy is specified. This element represents the right to copy the resource and at the same time to result in certain rights being associated to the copied resource. The optional

attribute @bpx:governanceRule indicates the name of a governance rule that determines how exactly the copy should be made and what rights should be associated and by whom for the copied resource. When the attribute is not specified, this right allows to make a bit-wise identical copy of the resource and to result in an identical copy of the r:license that this right is specified being made to the copied resource. This right can be used for copying the resource that is either locally available or received from a remote location (e.g., via streaming or broadcasting) during the time this right is being exercised.
In this use case we can use the governedCopy element of the REL base profile and the enlarge term defined in the RDD. Note, that a new extension for the REL shall be defined and must contain the enlarge element, in order to be further used in REL licenses.
Another option is to define the copy right in the RDD as a specialization of the Adapt right and add this element to the REL extension where also is specified the enlarge element.
A third option is to specify a new RDD term that represents the action presented (enlarge with a copied resource) in this section. This new term shall be a specialization of Copy and Enlarge. Then, this term shall be also defined in a new REL extension.
Nevertheless, MPEG-21 REL base profile is not currently supported in AXMEDIS, so the copy right will not be implemented for the moment.
In this scenario, the user shall have a license that grants him to copy AX02 from Composition 1 and to enlarge or enhance Composition2 with AX02.
MPEG-21 RDD does not define the copy right. Then, a new term shall be created under the governance of the RDD Registration Authority. This term will be defined as a specialization of Adapt defined in the RDD as the action “to ChangeTransiently an existing Resource to Derive a new Resource”.
Nevertheless, in the base profile the element *governedCopy* is specified. This element represents the right to copy a resource, in our use case the AX02 element, and at the same time to result in certain rights being associated to the copied resource. The attribute governanceRule of this right indicates the name of a governance rule that determines how exactly the copy should be made and what rights should be associated and by whom for the copied resource.



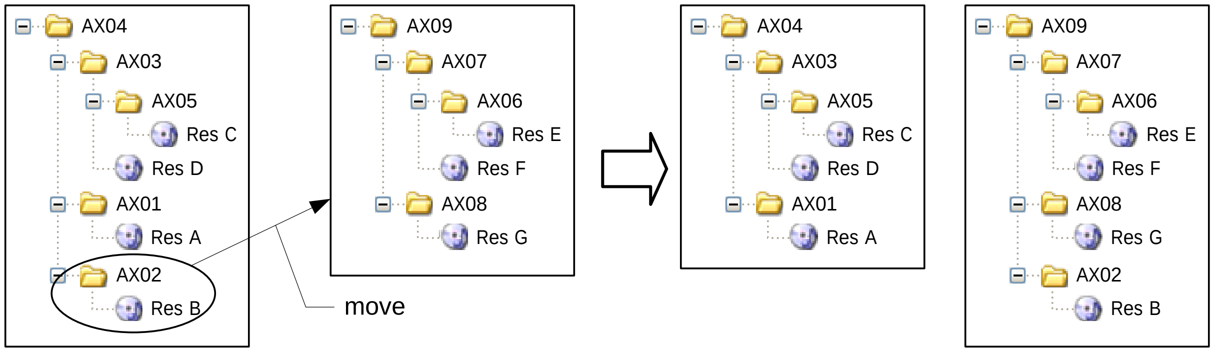
Licenses for copying use case

If we choose the first option, that is to define the copy right, then the licenses that shall be granted to the user in order to perform the copy from a governed composition to another are the presented in the figure.
Another option is the use of the adapt right in order to obtain a copy of AX02. Adapt is defined in the MPEG-21 RDD as the right to change transiently an existing resource to derive a new resource. But, take into account that the new resource can be modified, then it is necessary to define the copy right as an specialization of the adapt right or to add the appropriate conditions in the adaptation license to avoid modifications of the resultant object.



Licenses for copying use case – adapt option

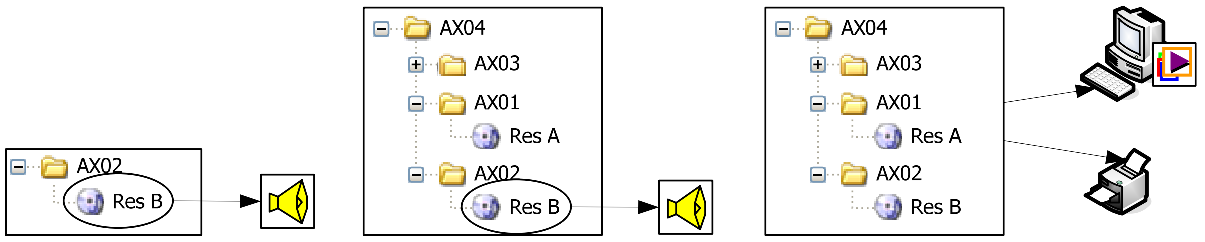
21.2.7 Moving of objects from a composition to another



Move resources from a composite object to another

Standardized RDD terms associated:
Move: To relocate a Resource from one Place to another.

21.2.8 Playing/rendering of objects (basic, composite)

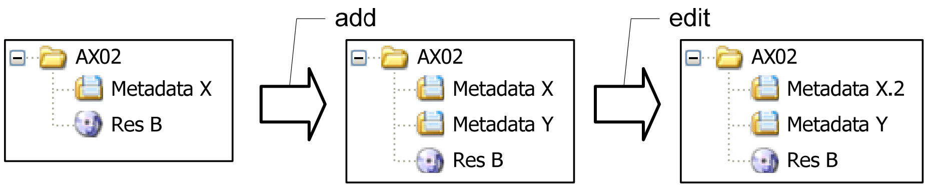


Playing resources

If a user has the right Play for the root object he has also the rights of playing the child objects.

Standardized RDD terms associated:
Play: To Derive a Transient and directly Perceivable representation of a Resource.
Print: To Derive a Fixed and directly Perceivable representation of a Resource.
Render: To Transform an existing Resource into a Perceivable representation of its contents.
Perform: To Express a Transient Resource.
Fix: To Express a Persistent Resource.

21.2.9 Manipulation of object metadata



Manipulating object metadata

Standardized RDD terms associated:
Enlarge: To Modify a Resource by adding to it.
Enlarge RDD term is equivalent to the add term of Figure above.
Translate: To Transform an existing Resource by changing the Language of its Lexical elements without changing their Meaning.

For this manipulation, it should be defined a new term Edit as a specialization of Transform RDD term.

21.3 Mapping Rights on User Action

The following table summarises the rights associated to each scenario described above. The semantics of the operation is also described.

Action	**Right**	**Semantics**
Adding to root level	Enmbed	To put a Resource into another Resource
	Enhance Enlarge	To derive a new resource which is larger than its source To Modify a Resource by adding to it
Adding to nested level	Embed	To put a Resource into another Resource.
	Enhance Enlarge	To derive a new resource which is larger than its source To Modify a Resource by adding to it
Transformation of basic objects	Modify	To Change a Resource, preserving the alterations made.
Transformation of objects in a composition	Modify	To Change a Resource, preserving the alterations made.
Deletion of objects from a composition (from root level or nested)	Reduce	To Modify a Resource by taking away from it.
Copying of objects from a composition to another	Copy Adapt	This right is not defined in the RDD, then a new term shall be added as a specialization of Adapt that is defined in RDD as the action to ChangeTransiently an existing Resource to Derive a new Resource. To change transiently an existing resource to derive a new resource
	governedCopy Enhance Enlarge	Element specified in the MPEG-21 REL base profile. It represents the right to copy the resource and at the same time to result in certain rights being associated to the copied resource. To derive a new resource which is larger than its source Modify a Resource by adding to it.
Moving of objects from a composition to another	Move	To relocate a Resource from one Place to another.

Playing/rendering of objects (basic, composite)	Play	To Derive a Transient and directly Perceivable representation of a Resource.
	Print	To Derive a Fixed and directly Perceivable representation of a Resource.
	Render	To Transform an existing Resource into a Perceivable representation of its contents.
	Perform	To Express a Transient Resource.
	Fix	To Express a Persistent Resource.
Manipulation of object metadata	Edit	It is not defined in the RDD. Define the new term Edit as a specialization of Transform RDD term.
		To Modify a Resource by adding to it.
	Translate	To Transform an existing Resource by changing the Language of its Lexical elements without changing their Meaning.