Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE3.1.2.3.14
Specification of AXMEDIS Protection Support – Update

Version: 0.3
Date: 16/07/2007
Responsible: UPC (revised and approved by coordinator)

Project Number: IST-2-511299 Project Title: AXMEDIS Deliverable Type: report Visible to User Groups: yes Visible to Affiliated: yes Visible to the Public: yes
Deliverable Number: DE3.1.2.3.14 Contractual Date of Delivery: M34 Actual Date of Delivery: 30/06/2007 Title of Deliverable: Specification of AXMEDIS Protection Support (see also parts 3 and 13) Work-Package contributing to the Deliverable: WP3.1 Task contributing to the Deliverable: WP3, WP2 Nature of the Deliverable: report Author(s): UPC

Abstract: this part includes the specification of components, formats, databases and protocol related to the AXMEDIS Framework area regarding Protection Support including PMS all versions and other protection issues (see also parts 3 and 13).

Keyword List: Protection Management Support, Security, Digital Rights Management

AXMEDIS Copyright Notice

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

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

1. **DEFINITIONS**

   i. "**Acceptance Date**" is the date on which these terms and conditions for entering into possession of the document have been accepted.

   ii. "**Copyright**" stands for any content, document or portion of it that is covered by the copyright disclaimer in a Document.

   iii. "**Licensor**" is AXMEDIS Consortium as a de-facto consortium of the EC project and any of its derivations in terms of companies and/or associations, see www.axmedis.org

   iv. "**Document**" means the information contained in any electronic file, which has been published by the Licensor's as AXMEDIS official document and listed in the web site mentioned above or available by any other means.

   v. "**Works**" means any works created by the licensee, which reproduce a Document or any of its part.

2. **LICENCE**

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

   2. In consideration of the Licensor granting the Licence, licensee agrees to adhere to the following terms and conditions.

3. **TERM AND TERMINATION**

   1. Granted Licence shall commence on Acceptance Date.

   2. Granted Licence will terminate automatically if licensee fails to comply with any of the terms and conditions of this Licence.

   3. Termination of this Licence does not affect either party's accrued rights and obligations as at the date of termination.

   4. Upon termination of this Licence for whatever reason, licensee shall cease to make any use of the accessed Copyright.

   5. All provisions of this Licence, which are necessary for the interpretation or enforcement of a party's rights or obligations, shall survive termination of this Licence and shall continue in full force and effect.

   6. Notwithstanding License termination, confidentiality clauses related to any content, document or part of it as stated in the document itself will remain in force for a period of 5 years after license issue date or the period stated in the document whichever is the longer.

4. **USE**

   1. Licensee shall not breach or denigrate the integrity of the Copyright Notice and in particular shall not:

      i. remove this Copyright Notice on a Document or any of its reproduction in any form in which those may be achieved;

      ii. change or remove the title of a Document;

      iii. use all or any part of a Document as part of a specification or standard not emanating from the Licensor without the prior written consent of the Licensor; or

      iv. do or permit others to do any act or omission in relation to a Document which is contrary to the rights and obligations as stated in the present license and agreed with the Licensor

5. **COPYRIGHT NOTICES**

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

6. **WARRANTY**

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

7. **INFRINGEMENT**

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

8. **GOVERNING LAW AND JURISDICTION**

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

Please note that:

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

Table of Content

1 Executive Summary and Report Scope

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

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

1.1 This document concerns Protection Support inside AXMEDIS project

Several modules provide protection inside AXMEDIS. The most important are Protection Manager Support (PMS) and Protection Tool Engine.

This document describes the updated specification of these tools and modules.

PMS is divided into different levels:

- PMS Client: A module include in the client side tools
- PMS Server: The server providing the full functionality for license management, authorisation of user actions, RDD support, etc.
- PMS Domain Factory: Light version of PMS Server, that has to be installed on content factories to work on a domain basis
- PMS Domain Home: Light version of PMS Domain Factory, which has to be installed at user home or at specific places, like schools or museums, to work on a domain basis.

1.2 List of Modules or Executable Tools Specified in this document

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

Module/tool Name	Module/Tool Description and purpose, state also in which other AXMEDIS area is used	Standards exploited if any
PMS Server	This is the server side providing licensing functionalities, together with authorisation of user actions and communication with the associated AXCS	MPEG-21 REL, MPEG-21 RDD
PMS Client	This is the client side providing secure caching functionalities, basic authorisation functionalities and communication with the rest of PMS from user side tools	MPEG-21 REL, MPEG-21 RDD
PMS Domain Factory	This is the server side providing licensing functionalities, together with authorisation of user actions and communication with the associated PMS Server. It does not have the whole functionality provided by PMS Server	MPEG-21 REL, MPEG-21 RDD
PMS Domain Home	This is the domain server providing basic domain functionality and communication features with associated PMS Server. It does not have the whole functionality provided by PMS Server nor PMS Domain Factory	MPEG-21 REL, MPEG-21 RDD
License Manager	This module provides the functionality for managing licenses associated to a PMS	MPEG-21 REL
License Verificator	This module verifies that the licenses created are correct according to syntactic and semantic rules	MPEG-21 REL
License Generator	This module provides license generation functionalities: distribution licenses, final user licenses and potential available rights	MPEG-21 REL, MPEG-21 RDD
Authorisation support	This module authorises user actions on the basis of the chain of licenses describing the actions granted to a user or group of users	MPEG-21 REL
RDD Server	This module provides functionality for requesting the hierarchy of rights associated to a right defined in an MPEG-21 license	MPEG-21 RDD
Protection Info Manager	This module provides access to the protection information associated to an AXMEDIS object	MPEG-21 IPMP
Key Generator	This module provides security keys to protect AXMEDIS objects	
Domain Manager	This module provides functionality for the management of domains at Home and Factory levels	
Domain Registration Manager	This modules allows the registration of users inside a domain in order to consume contents associated to the domain	
Rights Expression Translator	This module provides translation functionalities to pass from one rights expression language to another	
Secure cache manager	This module provides secure caching functionalities to store specific information related to user, PMS, domain, user context, etc.	

Content consumption status	This module stores user actions in the secure cache when working in an off-line scenario	

1.3 List of Formats Specified in this document

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

Format Name	Format Description and purpose, state also in which other modules is used	Standards exploited if any
License	Expresses the rights a user has over a content, expressed in XML format	MPEG-21 REL, OMA DRM REL

1.4 List of Databases Specified in this document

Database Name	Database Description and purpose, state also in which other AXMEDIS area is using	Standards exploited if any
License Database	Relational database for storing licenses at PMS level	MPEG-21 REL, OMA DRM REL
Secure Cache	Stores information regarding user status, licenses, etc., inside the secure cache	

1.5 List of Protocols Specified in this document

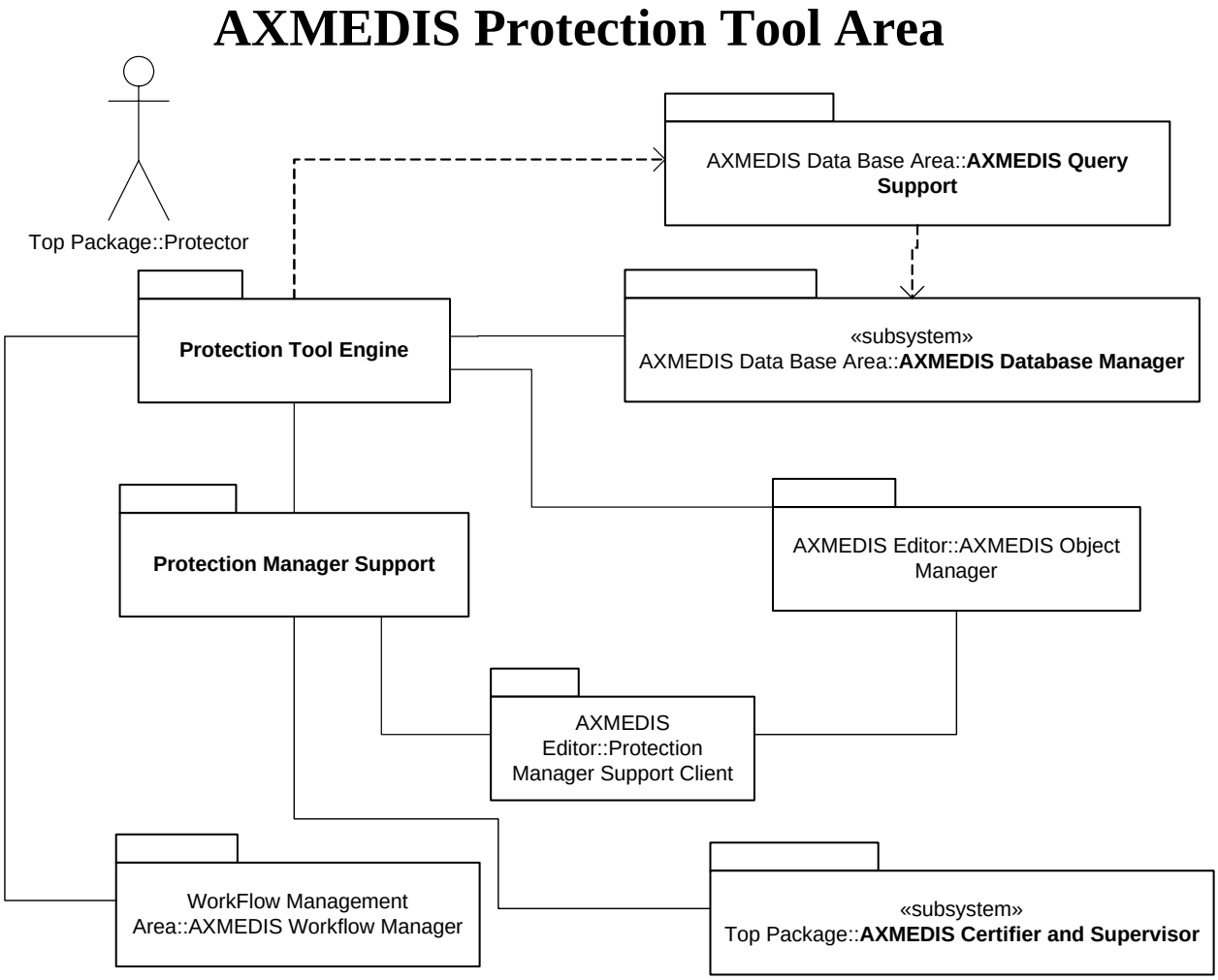
A protocol is a communication modality among distinct processes that can be located or not on different computers.

Protocol Name	Protocol Description and purpose, state also in which other modules is used	Who is the master and who is the slave	Standards exploited if any
Authorisation	Request authorisation to perform an action based on the chain of licenses	Master is PMS Server, slave is the PMS Client	MPEG-21 REL
License creation	Create a license for content distribution or fruition	Master is PMS Server, slave is the PMS Client	MPEG-21 REL
Key generation	Request a key for protecting an AXMEDIS Content	Master is PMS Server, slave is the PMS Client	

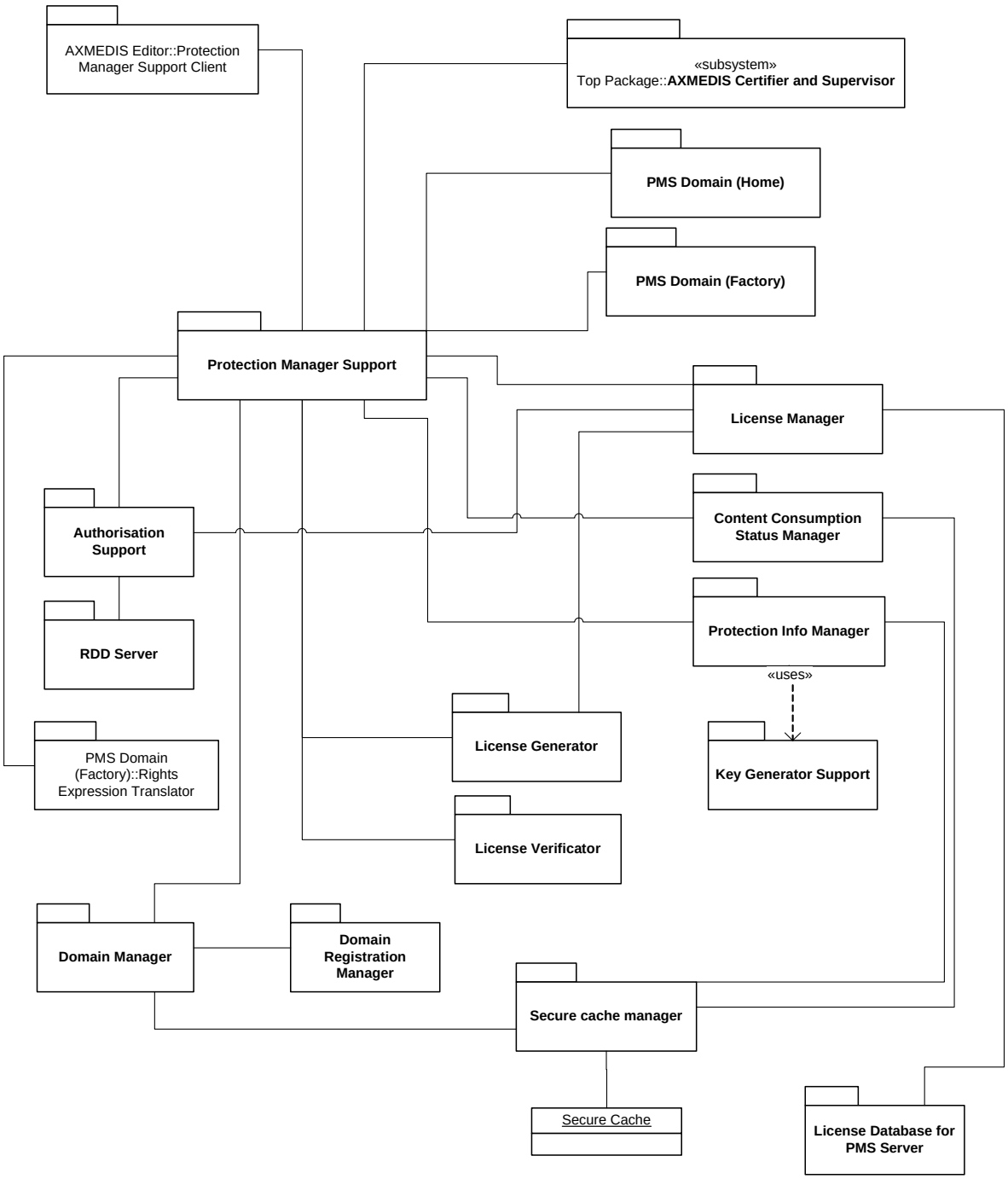
2 General architecture and relationships among the modules produced

The whole AXMEDIS system has been decomposed in subsystems and tools. The decomposition has been performed on the basis of structural aspects, the diagrams are reported in UML files in visio.

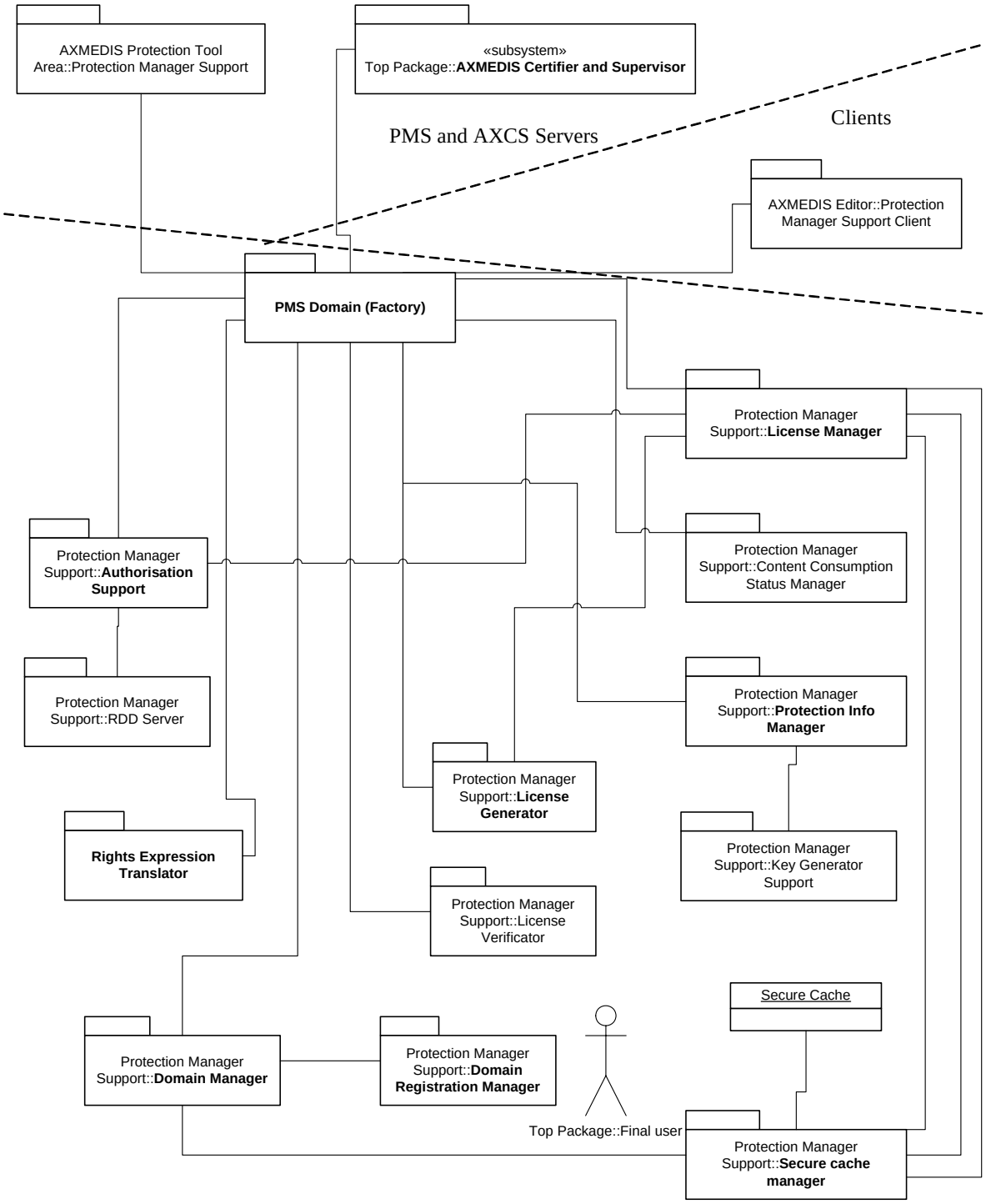
The following figures show the general structure of the AXMEDIS Protection Tool Area, Protection Management Support Server, Client, Domain Home and Domain Factory. This modules make use of (or are used by) several other modules inside the AXMEDIS project.

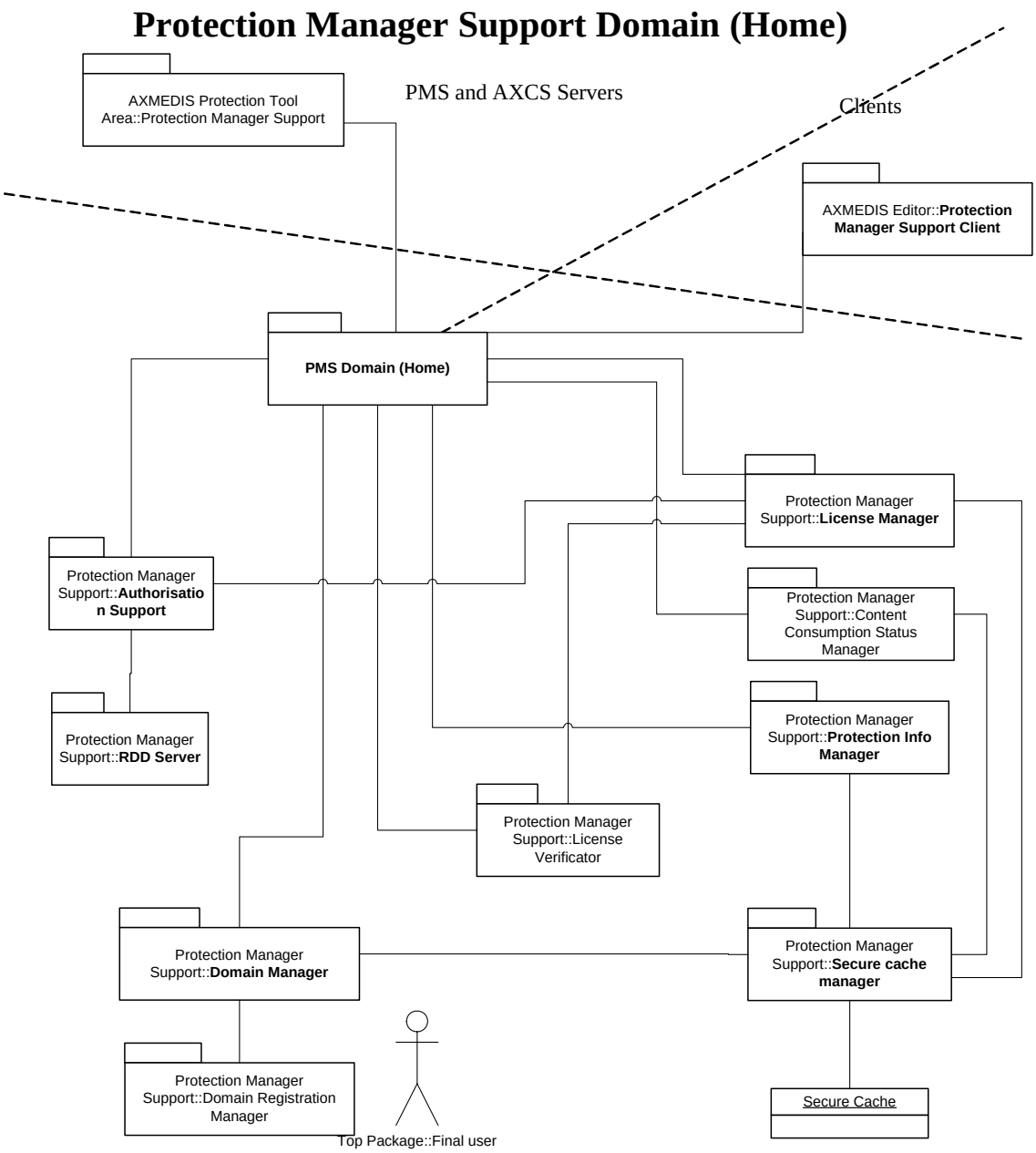


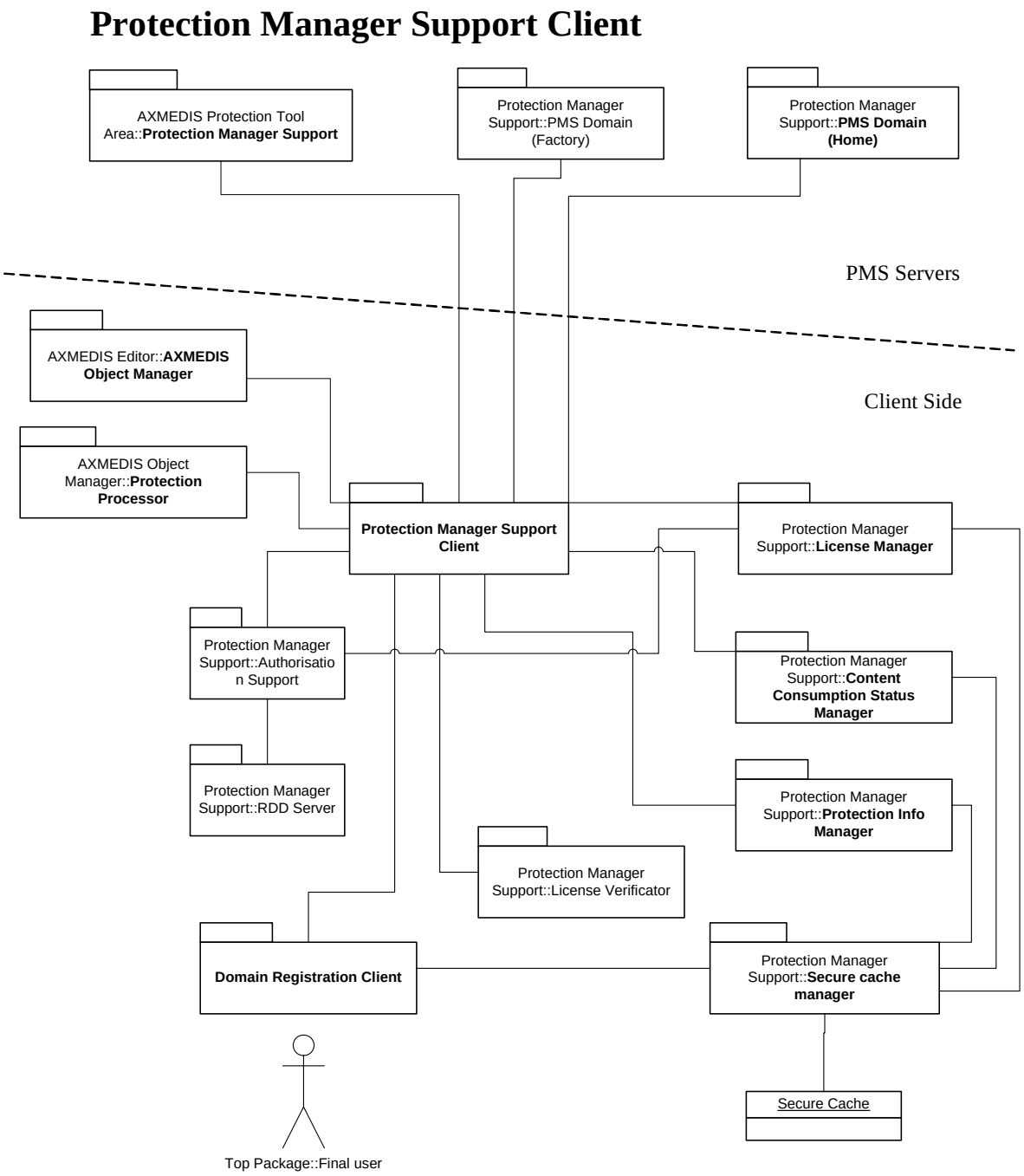
Protection Manager Support (Server, Home, Client)



Protection Manager Support Domain (Factory)




Protection Manager Support Domain (Home)
AXMEDIS Protection Tool Area::Protection Manager Support
PMS and AXCS Servers
Clients
AXMEDIS Editor::Protection Manager Support Client
PMS Domain (Home)
Protection Manager Support::License Manager
Protection Manager Support::Authorisation Support
Protection Manager Support::Content Consumption Status Manager
Protection Manager Support::RDD Server
Protection Manager Support::Protection Info Manager
Protection Manager Support::License Verificator
Protection Manager Support::Domain Manager
Protection Manager Support::Secure cache manager
Protection Manager Support::Domain Registration Manager
Top Package::Final user
Secure Cache



In the next sections, these tools are described in detail.

3 Protection Manager Support Server (UPC)

Module/Tool Profile		
Protection Manager Support Server (PMS Server)		
Responsible Name	Rubén Barrio	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First version	
Executable or Library/module (Support)	Executable, Web service	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/WebServices/PMSWs	
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	http://193.145.45.173:8502/PMS	
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	N/A	
Major pending requirements	N/A	
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.)
PMSClient		
AXCS		
Formats Used	Shared with	format name or reference to a section
XML		
Protocol Used	Shared with	Protocol name or reference to a

		section
SOAP		
Used Database name		
AXMEDIS		
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
Gsoap		
WxWidgets		
OpenSSL		
Mysql++		

3.1 General Description of the Module

PMS Server module is implemented as a C++ Web Service executable, which provides the protection needed for a set of PMS Domain Factory, Domain Home and Clients. It is connected to AXMEDIS Certifier and Supervisor, in order to check that users only perform the actions they are allowed to.

The PMS server module is the interface of the protection tools with all the other Axmedis remote modules. The PMS Server is called by other PMS's, and offers functionalities such as: creation of licenses, authorisation of actions, verification and certification of users an tools, and other functions described below.

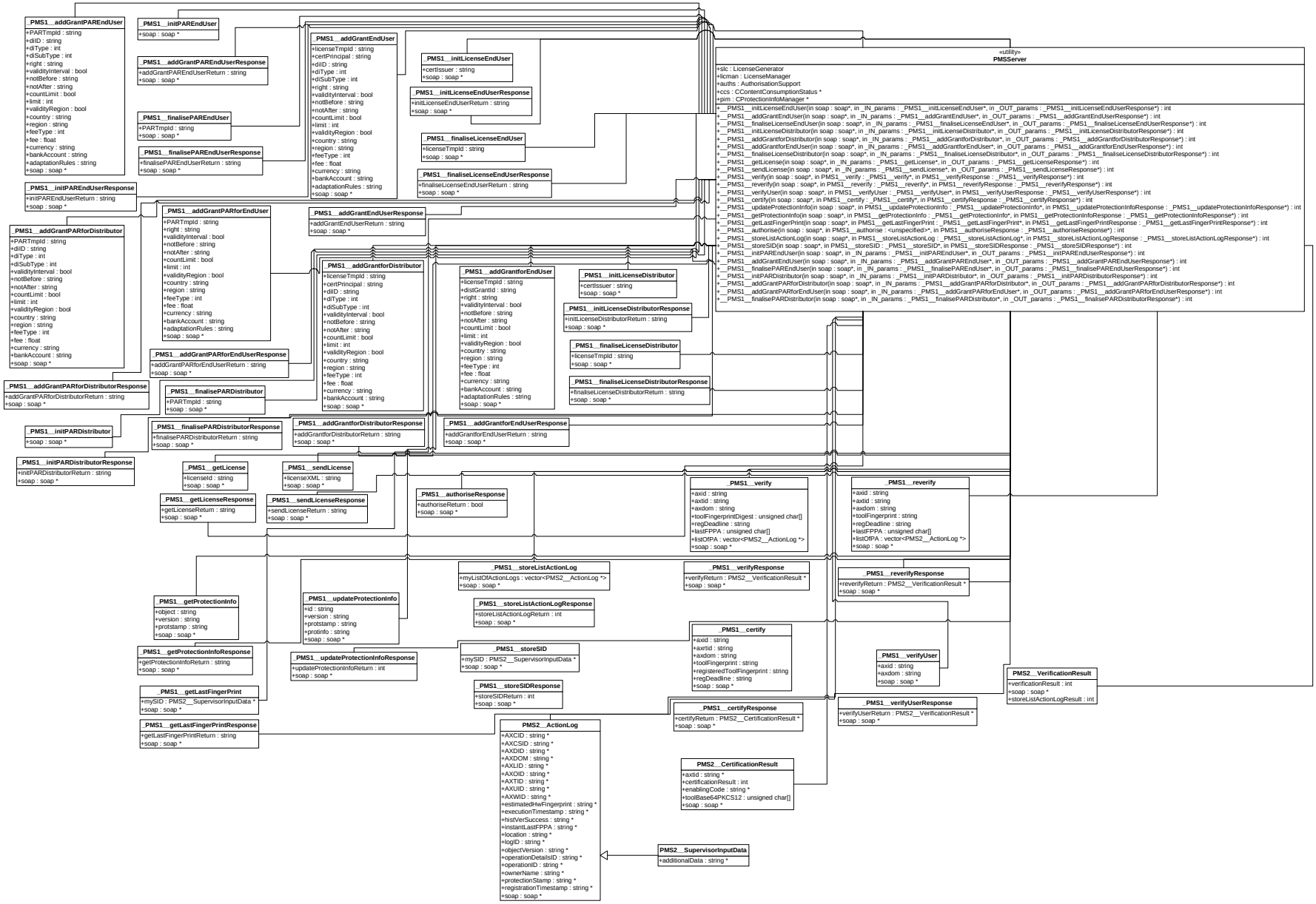
This module needs the configuration file licman.ini, containing the following fields:

host="IP of the Mysql LicenseDB"
licdatabase=”DatabaseName of LicenseDB”
pardatabase=”DatabaseName of PARDB”par
domaindatabase=”DatabaseName of DomainDB”
user="User to connect to LicenseDB"
password="Password to connect to LicenseDB"
bindaddress="Your own IP address"
AXCV="URL of AXCV"
AXS="URL of AXS"
SCDsn="Name of ODBC Connector for SecureCache"
SCuser="User of SecureCache"
SCpass="Password of SecureCache"
RDDDsn=AXRDDServer
secure=(true or false) if the server runs over SSL
PMSServCert=”Filename of server certificate”
paswPMSServCert=password for PMSServCert

AXCSClientCert=”Filename of client certificate for AXCS”
paswAXCSClientCert=”password for AXCSClientCert”
CACert=Certification Authority Certificate

The implemented module is supported on different platforms, as Windows OS specific libraries are not used (we use wxWindows instead), so it is only needed to recompile the source code. There is no need for user interface Multilanguage support, as this module does not have GUI.

3.2 Module Design in terms of Classes


«utility»
PMSServer
+slc : LicenseGenerator
+licman : LicenseManager
+auths : AuthorisationSupport
+ccs : CContentConsumptionStatus *
+pim : CProtectionInfoManager *
+__PMS1__initLicenseEndUser(in soap : soap*, in _IN_params : _PMS1__initLicenseEndUser*, in _OUT_params : _PMS1__initLicenseEndUserResponse*) : int
+__PMS1__addGrantEndUser(in soap : soap*, in _IN_params : _PMS1__addGrantEndUser*, in _OUT_params : _PMS1__addGrantEndUserResponse*) : int
+__PMS1__finaliseLicenseEndUser(in soap : soap*, in _IN_params : _PMS1__finaliseLicenseEndUser*, in _OUT_params : _PMS1__finaliseLicenseEndUserResponse*) : int
+__PMS1__initLicenseDistributor(in soap : soap*, in _IN_params : _PMS1__initLicenseDistributor*, in _OUT_params : _PMS1__initLicenseDistributorResponse*) : int
+__PMS1__addGrantforDistributor(in soap : soap*, in _IN_params : _PMS1__addGrantforDistributor*, in _OUT_params : _PMS1__addGrantforDistributorResponse*) : int
+__PMS1__addGrantforEndUser(in soap : soap*, in _IN_params : _PMS1__addGrantforEndUser*, in _OUT_params : _PMS1__addGrantforEndUserResponse*) : int
+__PMS1__finaliseLicenseDistributor(in soap : soap*, in _IN_params : _PMS1__finaliseLicenseDistributor*, in _OUT_params : _PMS1__finaliseLicenseDistributorResponse*) : int
+__PMS1__getLicense(in soap : soap*, in _IN_params : _PMS1__getLicense*, in _OUT_params : _PMS1__getLicenseResponse*) : int
+__PMS1__sendLicense(in soap : soap*, in _IN_params : _PMS1__sendLicense*, in _OUT_params : _PMS1__sendLicenseResponse*) : int
+__PMS1__verify(in soap : soap*, in PMS1__verify : _PMS1__verify*, in PMS1__verifyResponse : _PMS1__verifyResponse*) : int
+__PMS1__reverify(in soap : soap*, in PMS1__reverify : _PMS1__reverify*, in PMS1__reverifyResponse : _PMS1__reverifyResponse*) : int
+__PMS1__verifyUser(in soap : soap*, in PMS1__verifyUser : _PMS1__verifyUser*, in PMS1__verifyUserResponse : _PMS1__verifyUserResponse*) : int
+__PMS1__certify(in soap : soap*, in PMS1__certify : _PMS1__certify*, in PMS1__certifyResponse : _PMS1__certifyResponse*) : int
+__PMS1__updateProtectionInfo(in soap : soap*, in PMS1__updateProtectionInfo : _PMS1__updateProtectionInfo*, in PMS1__updateProtectionInfoResponse : _PMS1__updateProtectionInfoResponse*) : int
+__PMS1__getProtectionInfo(in soap : soap*, in PMS1__getProtectionInfo : _PMS1__getProtectionInfo*, in PMS1__getProtectionInfoResponse : _PMS1__getProtectionInfoResponse*) : int
+__PMS1__getLastFingerPrint(in soap : soap*, in PMS1__getLastFingerPrint : _PMS1__getLastFingerPrint*, in PMS1__getLastFingerPrintResponse : _PMS1__getLastFingerPrintResponse*) : int
+__PMS1__authorise(in soap : soap*, in PMS1__authorise : <unspecified>*, in PMS1__authoriseResponse : _PMS1__authoriseResponse*) : int
+__PMS1__storeListActionLog(in soap : soap*, in PMS1__storeListActionLog : _PMS1__storeListActionLog*, in PMS1__storeListActionLogResponse : _PMS1__storeListActionLogResponse*) : int
+__PMS1__storeSID(in soap : soap*, in PMS1__storeSID : _PMS1__storeSID*, in PMS1__storeSIDResponse : _PMS1__storeSIDResponse*) : int
+__PMS1__initPAREndUser(in soap : soap*, in _IN_params : _PMS1__initPAREndUser*, in _OUT_params : _PMS1__initPAREndUserResponse*) : int
+__PMS1__addGrantPARforDistributor(in soap : soap*, in _IN_params : _PMS1__addGrantPARforDistributor*, in _OUT_params : _PMS1__addGrantPARforDistributorResponse*) : int
+__PMS1__addGrantPARforEndUser(in soap : soap*, in _IN_params : _PMS1__addGrantPARforEndUser*, in _OUT_params : _PMS1__addGrantPARforEndUserResponse*) : int
+__PMS1__finalisePARDistributor(in soap : soap*, in _IN_params : _PMS1__finalisePARDistributor*, in _OUT_params : _PMS1__finalisePARDistributorResponse*) : int
_PMS1__addGrantPAREndUser
+PARTmpId : string
+dliD : string
+diType : int
+diSubType : int
+right : string
+validityInterval : bool
+notBefore : string
+notAfter : string
+countLimit : bool
+limit : int
+validityRegion : bool
+country : string
+region : string
+feeType : int
+fee : float
+currency : string
+bankAccount : string
+adaptationRules : string
+soap : soap *
_PMS1__initPAREndUser
+soap : soap *
_PMS1__addGrantPAREndUserResponse
+addGrantPAREndUserReturn : string
+soap : soap *
_PMS1__finalisePAREndUser
+PARTmpId : string
+soap : soap *
_PMS1__finalisePAREndUserResponse
+finalisePAREndUserReturn : string
+soap : soap *
_PMS1__initPAREndUserResponse
+initPAREndUserReturn : string
+soap : soap *
_PMS1__addGrantPARforDistributor
+PARTmpId : string
+dliD : string
+diType : int
+diSubType : int
+validityInterval : bool
+notBefore : string
+notAfter : string
+countLimit : bool
+limit : int
+validityRegion : bool
+country : string
+region : string
+feeType : int
+fee : float
+currency : string
+bankAccount : string
+soap : soap *
_PMS1__addGrantPARforDistributorResponse
+addGrantPARforDistributorReturn : string
+soap : soap *
_PMS1__addGrantPARforEndUser
+PARTmpId : string
+right : string
+validityInterval : bool
+notBefore : string
+notAfter : string
+countLimit : bool
+limit : int
+validityRegion : bool
+country : string
+region : string
+feeType : int
+fee : float
+currency : string
+bankAccount : string
+adaptationRules : string
+soap : soap *
_PMS1__addGrantPARforEndUserResponse
+addGrantPARforEndUserReturn : string
+soap : soap *
_PMS1__finalisePARDistributor
+PARTmpId : string
+soap : soap *
_PMS1__addGrantEndUser
+licenseTmpId : string
+certPrincipal : string
+diID : string
+diType : int
+diSubType : int
+right : string
+validityInterval : bool
+notBefore : string
+notAfter : string
+countLimit : bool
+limit : int
+validityRegion : bool
+country : string
+region : string
+feeType : int
+fee : float
+currency : string
+bankAccount : string
+adaptationRules : string
+soap : soap *
_PMS1__addGrantEndUserResponse
+addGrantEndUserReturn : string
+soap : soap *
_PMS1__addGrantforDistributor
+licenseTmpId : string
+certPrincipal : string
+diID : string
+diType : int
+diSubType : int
+validityInterval : bool
+notBefore : string
+notAfter : string
+countLimit : bool
+limit : int
+validityRegion : bool
+country : string
+region : string
+feeType : int
+fee : float
+currency : string
+bankAccount : string
+soap : soap *
_PMS1__addGrantforDistributorResponse
+addGrantforDistributorReturn : string
+soap : soap *
_PMS1__addGrantforEndUser
+licenseTmpId : string
+distGrantId : string
+right : string
+validityInterval : bool
+notBefore : string
+notAfter : string
+countLimit : bool
+limit : int
+validityRegion : bool
+country : string
+region : string
+feeType : int
+fee : float
+currency : string
+bankAccount : string
+adaptationRules : string
+soap : soap *
_PMS1__addGrantforEndUserResponse
+addGrantforEndUserReturn : string
+soap : soap *
_PMS1__initLicenseEndUser
+certIssuer : string
+soap : soap *
_PMS1__initLicenseEndUserResponse
+initLicenseEndUserReturn : string
+soap : soap *
_PMS1__finaliseLicenseEndUser
+licenseTmpId : string
+soap : soap *
_PMS1__finaliseLicenseEndUserResponse
+finaliseLicenseEndUserReturn : string
+soap : soap *
_PMS1__initLicenseDistributor
+certIssuer : string
+soap : soap *
_PMS1__initLicenseDistributorResponse
+initLicenseDistributorReturn : string
+soap : soap *
_PMS1__finaliseLicenseDistributor
+licenseTmpId : string
+soap : soap *
_PMS1__finaliseLicenseDistributorResponse
+finaliseLicenseDistributorReturn : string
+soap : soap *
_PMS1__initPARDistributor
+soap : soap *
_PMS1__finalisePARDistributorResponse
+finalisePARDistributorReturn : string
+soap : soap *
_PMS1__initPARDistributorResponse
+initPARDistributorReturn : string
+soap : soap *
_PMS1__getLicense
+licenseId : string
+soap : soap *
_PMS1__getLicenseResponse
+getLicenseReturn : string
+soap : soap *
_PMS1__sendLicense
+licenseXML : string
+soap : soap *
_PMS1__sendLicenseResponse
+sendLicenseReturn : string
+soap : soap *
_PMS1__authoriseResponse
+authoriseReturn : bool
+soap : soap *
_PMS1__verify
+axid : string
+axtid : string
+axdom : string
+toolFingerprintDigest : unsigned char[]
+regDeadline : string
+lastFPPA : unsigned char[]
+listOfPA : vector<PMS2__ActionLog *>
+soap : soap *
_PMS1__reverify
+axid : string
+axtid : string
+axdom : string
+toolFingerprint : string
+regDeadline : string
+lastFPPA : unsigned char[]
+listOfPA : vector<PMS2__ActionLog *>
+soap : soap *
_PMS1__storeListActionLog
+myListOfActionLogs : vector<PMS2__ActionLog *>
+soap : soap *
_PMS1__verifyResponse
+verifyReturn : PMS2__VerificationResult *
+soap : soap *
_PMS1__reverifyResponse
+reverifyReturn : PMS2__VerificationResult *
+soap : soap *
_PMS1__getProtectionInfo
+object : string
+version : string
+protstamp : string
+soap : soap *
_PMS1__updateProtectionInfo
+id : string
+version : string
+protstamp : string
+protInfo : string
+soap : soap *
_PMS1__storeListActionLogResponse
+storeListActionLogReturn : int
+soap : soap *
_PMS1__certify
+axid : string
+axrtid : string
+axdom : string
+toolFingerprint : string
+registeredToolFingerprint : string
+regDeadline : string
+soap : soap *
_PMS1__verifyUser
+axid : string
+axdom : string
+soap : soap *
_PMS1__getProtectionInfoResponse
+getProtectionInfoReturn : string
+soap : soap *
_PMS1__updateProtectionInfoResponse
+updateProtectionInfoReturn : int
+soap : soap *
_PMS1__storeSID
+mySID : PMS2__SupervisorInputData *
+soap : soap *
_PMS1__storeSIDResponse
+storeSIDReturn : int
+soap : soap *
_PMS1__getLastFingerPrint
+mySID : PMS2__SupervisorInputData *
+soap : soap *
_PMS1__getLastFingerPrintResponse
+getLastFingerPrintReturn : string
+soap : soap *
_PMS1__certifyResponse
+certifyReturn : PMS2__CertificationResult *
+soap : soap *
_PMS1__verifyUserResponse
+verifyUserReturn : PMS2__VerificationResult *
+soap : soap *
PMS2__VerificationResult
+verificationResult : int
+soap : soap *
+storeListActionLogResult : int
PMS2__ActionLog
+AXCID : string *
+AXCSID : string *
+AXDID : string *
+AXDOM : string *
+AXLID : string *
+AXOID : string *
+AXTID : string *
+AXUID : string *
+AXWID : string *
+estimatedHwFingerprint : string *
+executionTimestamp : string *
+histVerSuccess : string *
+instantLastFPPA : string *
+location : string *
+logID : string *
+objectVersion : string *
+operationDetailsID : string *
+operationID : string *
+ownerName : string *
+protectionStamp : string *
+registrationTimestamp : string *
+soap : soap *
PMS2__CertificationResult
+axtid : string *
+certificationResult : int
+enablingCode : string *
+toolBase64PKCS12 : unsigned char[]
+soap : soap *
PMS2__SupervisorInputData
+additionalData : string *

3.3 Integration and compilation issues

How to compile

Local Environment variables to be defined

OPENSSL -> Path to OpenSSL library
WXWIN -> Path to WxWidgets
XERCESROOT -> Path to Xerces Library
MYSQLROOT -> Path to Mysql server
MYSQL++ -> Path to Mysql++ Library

Use Requirements

1.- Install Mysql
2.- Install Mysql ODBC Driver
3.- Create a database with the tables defined in the file "SecureCache.sql"
4.- Create a database with the tables defined in the file "LicenseDB.sql"
5.- Grant a user (or two different) to access to these databases
6.- Create a Windows ODBC connector to SecureCache Mysql database
7.- Update licman.ini file to establish the application parameters

host="IP of the Mysql LicenseDB"
database="DatabaseName of LicecenseDB"
user="User to connect to LicenseDB"
password="Password to connect to LicenseDB"
bindaddress="Your own IP address"
AXCV="URL of AXCV"
AXS="URL of AXS"
SCDsn="Name of ODBC Connector for SecureCache"
SCuser="User of SecureCache"
SCpass="Password of SecureCache"

3.4 Configuration Parameters

The following table shows possible values for the configuration parameters stored in file “licman.ini”

Config parameter	Possible values
host	193.145.45.173
database	axmedis
user	axmedis
password	axmedis
bindaddress	193.145.45.173
AXCV	http:// 193.145.45.173:8080/axis/AXCV
AXS	http:// 193.145.45.173:8080/axis/AXS

3.5 Formal description of PMS Server operations

Authorise	
Method	authorise
Description	This function authorises AXMEDIS users to perform actions over AXMEDIS objects. It proves that a user has the appropriate license that grants him to exercise a right over a resource

	if the conditions are met based on the execution context of the client.
Input parameters	String userID: User id to be authorised String action: action to be authorised String resource: resource to be authorised contextData context: context of the client to be authorised ActionLog constructingAL: Actionlog of the authorisation with the “client side” parameters fulfilled
Output parameters	Integer result

getLicense	
Method	getLicense
Description	This function retrieves the licenses stored in the license database. It retrieves the license with the licenseID set as a parameter, or the licenses associated to an AXOID.
Input parameters	String licenseId: License Id
Output parameters	String, the license in XML

sendLicense	
Method	sendLicense
Description	This function stores a license in the license database.
Input parameters	String licenseXML: the license in XML format
Output parameters	String: result of the operation

InitLicenseEndUser	
Method	InitLicenseEndUser
Description	InitLicenseEndUser initialises the creation of a license. This is the first web service to be called in the process of an End User License creation. The service initLicenseEndUser returns the temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	IssuerAXUID String with the Issuer AXUID (creator of the license).
Output parameters	The temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored, this identifier is not used any more

AddGrantEndUser	
Method	AddGrantEndUser
Description	AddGrantEndUser is the web service that adds (one each time) the rights granted in a license. This service has to be called as many times as rights granted by the license. The different parameters allow introducing: the right, the resource over which the right will be exercised, the user who will obtain the right, and finally, the different conditions to be accomplished.
Input parameters	**licenseTmpId** Temporal license identifier, returned by initLicenseEndUser. **AXUIDPrincipal** This is the AXUID of the user (user of the license). **diResource** Establishes that the resource will be referenced by an URI. f.e. http://www.musicserver.org/track1.mp3 If this parameter is TRUE, diReference has to be FALSE diType Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource).

	(urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **right** The right that will be granted in the license. Can take the following values: adapt, delete, diminish, embed, enhance, enlarge, execute, install, modify, move, play, print, reduce, uninstall that correspond to rights described in "MPEG-21 multimedia extension". **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. fee If feeType is not 0, this parameter corresponds to the fee. currency If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different condition adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created and added to the license. If the right has not been created, the returned value is 4XX:Error causes. If the right has been correctly created, it returns 200:OK

finaliseLicenseEndUser	
Method	finaliseLicenseEndUser
Description	finaliseLicenseEndUser finalises the license. This is the last service to be invoked in a license creation process. The service builds the license and, if it is correct, then stores it in the database.
Input parameters	licenseTmpId String with the Temporal license ID returned by initLicenseEndUser.
Output parameters	A String with the license identifier. This is unique identifier of the license and can be used to retrieve a copy of the license

InitLicenseDistributor	
Method	InitLicenseDistributor
Description	InitLicenseDistributor initialises the creation of a license.

	This is the first web service to be called in the process of a Distributor License creation. This service receives information about the creator of the license. The service initLicenseEndUser returns the temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	IssuerAXUID String with the Issuer AXUID (normally creator of the content or rights owner).
Output parameters	The temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored, this identifier is not used any more

addGrantForDistributor	
Method	addGrantforDistributor
Description	addGrantforDistributor is the service that adds (one each time) the different rights for distributors and the distribution conditions for each one. The parameters established in this service affect only to the issue right (the one defining distribution). This service has to be called as many times as distributors the license has. The different parameters allow introducing: the distribution conditions, the content that will be distributed and the identification of the distributor.
Input parameters	**licenseTmpId** Temporal license identifier, returned by initLicenseDistributor. **AXUIDPrincipal** This is the AXUID of the principal (the distributor user). **diType** Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised. region If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the

	payment will be done.
Output parameters	a String with the temporal distributor grant ID. This identifier is usable while the license is being created, and it will be used to assign the different distributable rights to the distributor with AddGrantforEndUser.

addGrantforEndUser	
Method	addGrantforEndUser
Description	addGrantforEndUser is the service that adds (one each time) the rights that a distributor can distribute. In other words, this function adds the rights that can be included in an EndUser License created by a specific distributor. This service has to be called as many times as different rights will be available in the future EndUser licenses. The different parameters allow introducing: right and the different conditions to be accomplished. The resource is established before in the addGrantforDistributor service.
Input parameters	**licenseTmpId** Temporal license identifier, returned by initLicenseDistributor. **distGrantId** Temporal grant identifier, returned by AddGrantforDistributor. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBeforeIf** validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. country If validityRegion is TRUE, this parameter corresponds to the **country** where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created into the license. If the right has not been created, the returned value is 4XX:Error causes. If the right has been created normally, it returns 200:OK.

finaliseLicenseDistributor	
Method	finaliseLicenseDistributor
Description	finaliseLicenseDistributor finalises the license. This is the last service to be invoked in a license creation process. The service builds the licenses and, if it is correct, then stores it in the database.
Input	licenseTmpId Temporal license identifier, returned by initLicenseDistributor.

parameters	
Output parameters	String with the license identifier. This is a unique identifier of the license and can be used to retrieve a copy of the license

InitPAREndUser	
Method	InitPAREndUser
Description	InitPAREndUser initialises the creation of a PAR. This is the first function to be called in the process of an End User PAR creation. The service initPAREndUser returns the temporal identifier of the PAR. This identifier is usable while the PAR is being created. When the PAR is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	
Output parameters	The temporal identifier of the PAR. This identifier is usable while the PAR is being created. When the PAR is finished and stored, this identifier is not used any more

AddGrantPAREndUser	
Method	AddGrantPAREndUser
Description	AddGrantPAREndUser is the function that adds (one each time) the rights granted in a PAR. This service has to be called as many times as rights granted by the PAR. The different parameters allow introducing: the right, the resource over which the right will be exercised, the user who will obtain the right, and finally, the different conditions to be accomplished.
Input parameters	**PARTmpId** Temporal PAR identifier, returned by initPAREndUser. **diResource** Establishes that the resource will be referenced by an URI. f.e. http://www.musicserver.org/track1.mp3 If this parameter is TRUE, diReference has to be FALSE diType Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **right** The right that will be granted in the PAR. Can take the following values: adapt, delete, diminish, embed, enhance, enlarge, execute, install, modify, move, play, print, reduce, uninstall that correspond to rights described in "MPEG-21 multimedia extension". **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the

	country where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. fee If feeType is not 0, this parameter corresponds to the fee. currency If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different condition adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created and added to the PAR. If the right has not been created, the returned value is 4XX:Error causes. If the right has been correctly created, it returns 200:OK

finalisePAREndUser	
Method	finalisePAREndUser
Description	finalisePAREndUser finalises the PAR. This is the function to be invoked in a PAR reation process. The function builds the PAR and, if it is correct, then stores it in the database.
Input parameters	PARTmpId String with the Temporal PAR ID returned by initPAREndUser.
Output parameters	A String with the PAR identifier. This is unique identifier of the PAR and can be used to retrieve a copy of the PAR

InitPARDistributor	
Method	InitPARDistributor
Description	InitPARDistributor initialises the creation of a PAR. This is the first function to be called in the process of a Distributor PAR creation. This service receives information about the creator of the PAR. The function initPAREndUser returns the temporal identifier of the PAR. This identifier is usable while the PAR is being created. When the PAR is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	
Output parameters	The temporal identifier of the PAR. This identifier is usable while the PAR is being created. When the PAR is finished and stored, this identifier is not used any more

addGrantPARforDistributor	
Method	addGrantPARforDistributor
Description	addGrantPARforDistributor is the function that adds (one each time) the different rights for distributors and the distribution conditions for each one. The parameters established in this function affect only to the issue right (the one defining distribution).

	This function has to be called as many times as distributors the PAR has. The different parameters allow introducing: the distribution conditions, the content that will be distributed and the identification of the distributor.
Input parameters	**PARTmpId** Temporal PAR identifier, returned by initPARDistributor. **diType** Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised. region If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done.
Output parameters	a String with the temporal distributor grant ID. This identifier is usable while the PAR is being created, and it will be used to assign the different distributable rights to the distributor with AddGrantforEndUser.

addGrantPARforEndUser	
Method	addGrantPARforEndUser
Description	addGrantPARforEndUser is the function that adds (one each time) the rights that a distributor can distribute. In other words, this function adds the rights that can be included in an EndUser PAR created by a specific distributor. This function has to be called as many times as different rights will be available in the future EndUser PARs. The different parameters allow introducing: right and the different conditions to be accomplished. The resource is established before in the addGrantforDistributor service.
Input parameters	**PARTmpId** Temporal PAR identifier, returned by initPARDistributor. **distGrantId** Temporal grant identifier, returned by AddGrantforDistributor. **validityInterval** If this parameter is TRUE the right can be exercised within a time

	period. If it is FALSE it could be exercised always. **notBeforeIf** validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. country If validityRegion is TRUE, this parameter corresponds to the **country** where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created into the PAR. If the right has not been created, the returned value is 4XX:Error causes. If the right has been created normally, it returns 200:OK.

finalisePARDistributor	
Method	finalisePARDistributor
Description	finalisePARDistributor finalises the PAR. This is the last function to be invoked in a PAR creation process. The service builds the PARs and, if it is correct, then stores it in the database.
Input parameters	PARTmpId Temporal PAR identifier, returned by initPARDistributor.
Output parameters	String with the PAR identifier. This is a unique identifier of the PAR and can be used to retrieve a copy of the PAR

verifyUser	
Method	verifyUser
Description	This method is called by PMS Client and reaches AXCV through PMS Server. It can be used to verify the status of a user, optionally inside a domain. It verifies if the user is registered in the specified domain (if present) and checks that the user status and registration deadline are valid, so that the user can still use the AXMEDIS tools and the AXMEDIS framework.
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axdom**: AXMEDIS domain of certified user (if any)
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: Verification OK -1: invalid AXID

	-2: user is not registered -3: user is blocked -4: user domain mismatch -5: user registration deadline expired When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur, but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the deadline has expired or not).

certify	
Method	certify
Description	This method is called by PMS Client and reaches AXCV through PMS Server. It is used to certify that the original tool has not been modified and to activate it. It creates a new entry in the CerTools table of the AXCS database which associates the user, tool and device and returns to the Protection Processor an activation code, a tool identifier and a PKCS12 structure with the tool certificate and private key issued by AXMEDIS CA (EJBCA).
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axrtid**: identifier of the registered AXMEDIS tool xsd:string **axdom**: domain where the user is registered. xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool xsd:string **regDeadline**: registration deadline of the installed tool.
Output parameters	CertificationResult complex type formed by sequence of: xsd:string **axtid**, the identifier of the installed tool associated to a user and device. xsd:int **certificationResult**, which indicates the result of the certification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: tool not registered (RegTools table) -7: registered tool is blocked -8: received tool deadline exceeds registered tool deadline (user and tool have been blocked) -9: received tool deadline has expired -10: registered tool fingerprint mismatch. Tool has been manipulated (user and tool have been blocked) -11: user-tool-device had already been certified. New tool certificate should be created -20: error updating user status in database -21: error inserting new entry in CerTools table -22: error in AXSupervisor when communicating with database -30: internal AXCV error xsd:string **enablingCode**, the tool activation code sent to the Protection Processor. byte[] **toolBase64PKCS12**, PKCS12 structure bytes encoded in Base 64. It includes the tool certificate signed by the AXCS CA Root Certificate and tool private key together and protected with a password. If the unrestricted policy files for Sun JCE were available at the server (default configuration), the password will be the full AXMEDIS AXID. Otherwise, the password will be the first 8 characters of the AXMEDIS AXID. It proves that an AXMEDIS tool has been certified and can be used in the AXMEDIS framework When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur,

	but all possible errors $y<x$ have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the registered tool is blocked or not, or if the tool fingerprint did match or not).

certifyForMobile	
Method	certifyForMobile
Description	This method is called by PMS Client in Mobile and reaches AXCV through PMS Server. It is used to certify that the original tool has not been modified and to activate it. It creates a new entry in the CerTools table of the AXCS database which associates the user, tool and device and returns to the Protection Processor an activation code, a tool identifier and a PKCS12 structure with the tool certificate and private key issued by AXMEDIS CA (EJBCA).
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axrtid**: identifier of the registered AXMEDIS tool xsd:string **axdom**: domain where the user is registered. xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool xsd:string **regDeadline**: registration deadline of the installed tool.
Output parameters	CertificationResult complex type formed by sequence of: xsd:string **axtid**, the identifier of the installed tool associated to a user and device. xsd:int **certificationResult**, which indicates the result of the certification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: tool not registered (RegTools table) -7: registered tool is blocked -8: received tool deadline exceeds registered tool deadline (user and tool have been blocked) -9: received tool deadline has expired -10: registered tool fingerprint mismatch. Tool has been manipulated (user and tool have been blocked) -11: user-tool-device had already been certified. New tool certificate should be created -20: error updating user status in database -21: error inserting new entry in CerTools table -22: error in AXSupervisor when communicating with database -30: internal AXCV error When an error code x is returned, it means that all the possible errors y, $x<y<0$ did not occur, but all possible errors $y<x$ have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the registered tool is blocked or not, or if the tool fingerprint did match or not).

verify	
Method	verify
Description	This method is called by PMS Client and reaches AXCV through PMS Server. It is used to verify that the tool installed on a device has neither been modified nor blocked, that the user is not blocked and that the registered tool is not blocked. It is also responsible for resynchronizing the offline tool operation through AXMEDIS Supervisor (AXS).
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID)

	xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axdom**: domain where the user is registered. byte[] **toolFingerprintDigest**: SHA1 hash of the relevant data of hash of the full fingerprint (software and hardware parts) of the installed tool. byte[] **LastFPPA**: fingerprint of the history of the operations performed during the offline operation. tns2:ActionLog **listOfPA**: Array of ActionLogs, which is a complex type defined in AXMEDIS Supervisor, including the actions performed during the offline operation.
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: AXTID does not exist -7: installed (and certified) tool is blocked -8: tool deadline has expired -9: toolFingerprintDigest (toolFingerprint hash) mismatch -10: toolFingerprint mismatch (user and tool have been blocked) -11: registered tool is blocked -12: user has been blocked and installed tool has been blocked again -13: tool has been blocked -20: error updating user status in database -21: error updating tool status in database -22: error updating LastFPPA in database -23: error retrieving regtool data from database -24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog -25: error in AXSupervisor when communicating with AXCS accounting database in storeSID -30: internal AXCV error xsd:int **storeListActionLogResult**, which indicates the result of the storage of the action logs, according to the following numeration: 0: ActionLog(s) has been stored: it includes the case of empty list -1: ActionLog(s) has been stored: tool should have been already blocked -2: ActionLog(s) has been stored: tool operation history hash (LastFPPA) is not consistent -3: ActionLog(s) has not been stored: error in AXSupervisor when communicating with AXCS database -4: ActionLog(s) has not been stored: input actionLog(s) do not refer to the same AXTID -5: ActionLog(s) has not been stored: input actionLog(s) have some non-nillable null fields -6: ActionLog(s) has not been stored: user or tool data unsuccessfully verified by AXCV When an error code *x* is returned, it means that all the possible errors *y*, $x<y<0$ did not occur, but all possible errors $y<x$ have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the received tool deadline has expired or not).

verifyForMobile	
Method	verifyForMobile
Description	This method is called by PMS Client in Mobiles and reaches AXCV through PMS Server. It is used to verify that the tool installed on a device has neither been modified nor blocked, that the user is not blocked and that the registered tool is not blocked. It is also responsible for resynchronizing the offline tool operation through AXMEDIS Supervisor (AXS).
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axdom**: domain where the user is registered. String (base64) **toolFingerprintDigest**: SHA1 hash of the relevant data of hash of the full fingerprint (software and hardware parts) of the installed tool. String (base 64) **LastFPPA**: fingerprint of the history of the operations performed during the offline operation.
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: AXTID does not exist -7: installed (and certified) tool is blocked -8: tool deadline has expired -9: toolFingerprintDigest (toolFingerprint hash) mismatch -10: toolFingerprint mismatch (user and tool have been blocked) -11: registered tool is blocked -12: user has been blocked and installed tool has been blocked again -13: tool has been blocked -20: error updating user status in database -21: error updating tool status in database -22: error updating LastFPPA in database -23: error retrieving regtool data from database -24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog -25: error in AXSupervisor when communicating with AXCS accounting database in storeSID -30: internal AXCV error When an error code *x* is returned, it means that all the possible errors *y*, $x<y<0$ did not occur, but all possible errors $y<x$ have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the received tool deadline has expired or not).

reverify	
Method	reverify
Description	This method is similar to verify method (see previous). It must be called when the verify method fails because of the tool fingerprint hash doesn't match (error code: –9) to perform a new verification with the full fingerprint. Thus, the reverify method has the same input

	parameters as the verify method except the full tool fingerprint, which has to be sent instead of the hash.
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string axtid: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axdom**: domain where the user is registered. xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool. byte[] **LastFPPA**: fingerprint of the history of the operations performed during the offline operation. tns2:ActionLog **listOfPA**: Array of ActionLogs, which is a complex type defined in AXMEDIS Supervisor, including the actions performed during the offline operation.
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: AXTID does not exist -7: installed (and certified) tool is blocked -8: tool deadline has expired -9: toolFingerprintDigest (toolFingerprint hash) mismatch -10: toolFingerprint mismatch (user and tool have been blocked) -11: registered tool is blocked -12: user has been blocked and installed tool has been blocked again -13: tool has been blocked -20: error updating user status in database -21: error updating tool status in database -22: error updating LastFPPA in database -23: error retrieving regtool data from database -24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog -25: error in AXSupervisor when communicating with AXCS accounting database in storeSID -30: internal AXCV error xsd:int **storeListActionLogResult**, which indicates the result of the storage of the action logs, according to the following numeration: 0: ActionLog(s) has been stored: it includes the case of empty list -1: ActionLog(s) has been stored: tool should have been already blocked -2: ActionLog(s) has been stored: tool operation history hash (LastFPPA) is not consistent -3: ActionLog(s) has not been stored: error in AXSupervisor when communicating with AXCS database -4: ActionLog(s) has not been stored: input actionLog(s) do not refer to the same AXTID -5: ActionLog(s) has not been stored: input actionLog(s) have some non-nillable null fields -6: ActionLog(s) has not been stored: user or tool data unsuccessfully verified by AXCV

	When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur, but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the received tool deadline has expired or not).

reverifyForMobile	
Method	reverifyForMobile
Description	This method is similar to verify method (see previous). It must be called when the verify method fails because of the tool fingerprint hash doesn't match (error code: –9) to perform a new verification with the full fingerprint. Thus, the reverify method has the same input parameters as the verify method except the full tool fingerprint, which has to be sent instead of the hash.
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string axtid: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axdom**: domain where the user is registered. xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool. String (base64)**LastFPPA**: fingerprint of the history of the operations performed during the offline operation.
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: AXTID does not exist -7: installed (and certified) tool is blocked -8: tool deadline has expired -9: toolFingerprintDigest (toolFingerprint hash) mismatch -10: toolFingerprint mismatch (user and tool have been blocked) -11: registered tool is blocked -12: user has been blocked and installed tool has been blocked again -13: tool has been blocked -20: error updating user status in database -21: error updating tool status in database -22: error updating LastFPPA in database -23: error retrieving regtool data from database -24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog -25: error in AXSupervisor when communicating with AXCS accounting database in storeSID -30: internal AXCV error When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur, but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the received tool deadline has expired or not).

getProtectionInfo	
Method	getProtectionInfo
Description	This method is called by PMS Client and is used to retrieve the protection information related to an object from the Objects Table of the AXCS Objects ID Database through AXCS.
Input parameters	The following fields of the Objects table in the AXCS Objects ID database: type="xsd:string" **AXOID**, AXMEDIS object identifier type="xsd:string" **ObjectVersion**, object version type="xsd:string" **ProtectionStamp**, protection stamp
Output parameters	type="xsd:string" **ProtectionInfo**, protection information associated to the object or "wrong_object" result if there is no ProtectionInfo for the requested object

UpdateProtectionInfo	
Method	UpdateProtectionInfo
Description	This method is called by PMS Client and is used to insert or update the protection information related to an AXMEDIS object in the Objects Table of the AXCS Objects ID Database.
Input parameters	The following fields of the Objects table in the AXCS Objects ID database: type="xsd:string" **AXOID**, AXMEDIS object identifier type="xsd:string" **ObjectVersion**, object version type="xsd:string" **ProtectionStamp**, protection stamp type="xsd:string" **ProtectionInfo**, protection information to be updated type="xsd:int" **Update**, denotes if the protection info must be inserted (0) or updated (1)
Output parameters	type="xsd:int" **updateProtectionInfoReturn**, which indicates the result of this request, according to the following numeration: 0: OK -1: there is not any entry in AXCS Objects database that matches the input information -2: error in AXSupervisor when updating ProtectionInfo in AXCS Objects database

4 Protection Manager Support Client (UPC)

Module/Tool Profile		
Protection Manager Support Client (PMS Client)		
Responsible Name	Rubén Barrio	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First version	
Executable or Library/module (Support)	Library	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/pmsclient	
Reference to the AXFW location of the demonstrator executable tool for internal download	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.)
PMS Server	Gsoap	
Formats Used	Shared with	format name or reference to a section
XML		
Protocol Used	Shared with	Protocol name or reference to a

		section
SOAP		
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
Gsoap		
WxWidgets		
OpenSSL		

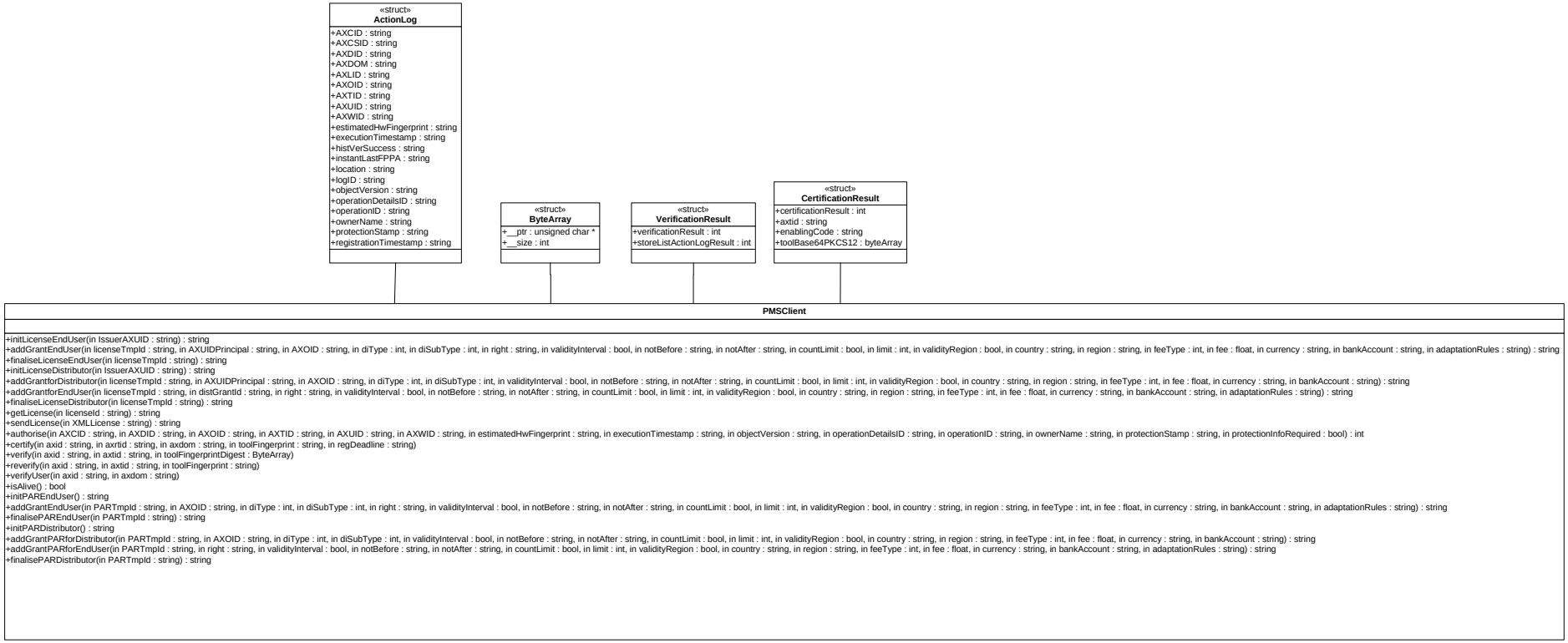
4.1 General Description of the Module

PMS Client module is implemented as a C++ class, which provides to an AXMEDIS tool the access to PMS and AXCS security and protection mechanisms. The PMS Client can work on a connected environment and talk with PMS Server, and can work on an unconnected environment, tracking user operations locally.

While the PMS client works offline, it stores the actions in a local Secure Cache, and, when gets connection it synchronizes with the PMS Server.

Basically, when the PMS Server is online and accessible, the PMS Client works as a trustable gateway between the Axmedis Tool and the PMS Server (and AXCS). And when the PMS Server is offline, the PMS Client takes the responsibility of authorising and logging actions.

4.2 Module Design in terms of Classes



4.3 Examples of usage

```
PMSClient *pmsc;

pmsc = new
PMSClient("http://193.145.45.70:8502/PMS","AXSecureCache","axmedis","axmedis");

pmsc->getProtectionInfo("uno","otro","otromas");

templic=pmsc->initLicenseEndUser("Issuer1");
```

4.4 Integration and compilation issues

How to compile

Local Environment variables to be defined

OPENSSL -> Path to OpenSSL library
WXWIN -> Path to WxWidgets

Framework projects needed

PMSClient
ContentConsumptionStatus
ProtectionInfomanager
EncDecSup
SecureCache

Usage Requirements

1.- Install Mysql
2.- Install Mysql ODBC Driver
3.- Create a database with the tables defined in the file "SecureCache.sql" (see securecache module)
4.- Grant a user to access to this database
4.- Create a Windows ODBC connector to this Mysql database

4.5 Errors reported and that may occur

These codes are the possible errors of authorise function in PMS Client. These errors summarize all the possible errors reported in the servers (PMS Server, AXCV).

Error code	Description and rationales
-1200	Prot Info required and no present in Secure Cache
-1201	Prot Info required and database error in Secure Cache
-1001 to -1128	Authorise failed in Offline mode (substract –1000, and see authorise support error table)
-1300	Error storing History Hash in Secure Cache
-1301	Error Storing ActionLog in Secure Cache
-1302	Error storing Number of Executions in Secure Cache
-2001 to -2128	Authorise failed in semi Online mode (substract –2000, and see authorise support error table)
-2200	PMS offline when must be online, reauthorize.
-3000	Pending Action Logs in cache in Online mode, must Verify.
-3001 to -3128	Authorise failed in Online mode (substract –3000, and see authorise support error table)
-3200	PMS offline when must be online, reauthorise

-3201	Prot Info required and no present in AXCV
-3202	Prot Info required and database error in AXCV
-4000	AXCV offline, when must be online, reauthorise
-4xxy	Error Verifing ActionLog in AXCV xx -> Verification Result y -> Store Action Log Result

4.6 Formal description of PMS Client functionality

authorise	
Method	authorise
Description	This function authorises AXMEDIS users to perform actions over AXMEDIS objects. It proves that a user has the appropriate license that grants him to exercise a right over a resource if the conditions are met based on the execution context of the client.
Input parameters	String userID: User id to be authorised String action: action to be authorised String resource: resource to be authorised contextData context: context of the client to be authorised ActionLog constructingAL: Actionlog of the authorisation with the “client side” parameters fulfilled
Output parameters	Integer:

getLicense	
Method	getLicense
Description	This function retrieves the licenses stored in the license database. It retrieves the license with the licenseID set as a parameter.
Input parameters	String licenseId: License Id
Output parameters	String, the license in XML

sendLicense	
Method	sendLicense
Description	This function stores a license in the license database.
Input parameters	String licenseXML: the license in XML format
Output parameters	String: result of the operation

getPAR	
Method	getPAR
Description	This function retrieves the PAR stored in the PAR database. It retrieves the PAR with the PARID set as a parameter.
Input parameters	String licenseId: PAR Id
Output parameters	String, the license in XML

sendPAR	
Method	sendPAR

Description	This function stores a PARin the PARdatabase.
Input parameters	String PARXML: the PARin XML format
Output parameters	String: result of the operation

InitLicenseEndUser	
Method	InitLicenseEndUser
Description	InitLicenseEndUser initialises the creation of a license. This is the first web service to be called in the process of an End User License creation. The service initLicenseEndUser returns the temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	IssuerAXUID String with the Issuer AXUID (creator of the license).
Output parameters	

AddGrantEndUser	
Method	AddGrantEndUser
Description	AddGrantEndUser is the web service that adds (one each time) the rights granted in a license. This service has to be called as many times as rights granted by the license. The different parameters allow introducing: the right, the resource over which the right will be exercised, the user who will obtain the right, and finally, the different conditions to be accomplished.
Input parameters	**licenseTmpId** Temporal license identifier, returned by initLicenseEndUser. **AXUIDPrincipal** This is the AXUID of the user (user of the license). **diResource** Establishes that the resource will be referenced by an URI. f.e. http://www.musicserver.org/track1.mp3 If this parameter is TRUE, diReference has to be FALSE diType Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **right** The right that will be granted in the license. Can take the following values: adapt, delete, diminish, embed, enhance, enlarge, execute, install, modify, move, play, print, reduce, uninstall that correspond to rights described in “MPEG-21 multimedia extension”. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised.

	validityRegion It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. fee If feeType is not 0, this parameter corresponds to the fee. currency If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different condition adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created and added to the license. If the right has not been created, the returned value is 4XX:Error causes. If the right has been correctly created, it returns 200:OK

finaliseLicenseEndUser	
Method	finaliseLicenseEndUser
Description	finaliseLicenseEndUser finalises the license. This is the last service to be invoked in a license reation process. The service builds the license and, if it is correct, then stores it in the database.
Input parameters	licenseTmpId String with the Temporal license ID returned by initLicenseEndUser.
Output parameters	A String with the license identifier. This is unique identifier of the license and can be used to retrieve a copy of the license

InitLicenseDistributor	
Method	InitLicenseDistributor
Description	InitLicenseDistributor initialises the creation of a license. This is the first web service to be called in the process of a Distributor License creation. This service receives information about the creator of the license. The service initLicenseEndUser returns the temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	IssuerAXUID String with the Issuer AXUID (normally creator of the content or rights owner).
Output parameters	The temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored, this identifier is not used any more

addGrantforDistributor	
Method	addGrantforDistributor
Description	addGrantforDistributor is the service that adds (one each time) the different rights for distributors and the distribution conditions for each one. The parameters established in this service affect only to the issue right (the one defining distribution).

	This service has to be called as many times as distributors the license has. The different parameters allow introducing: the distribution conditions, the content that will be distributed and the identification of the distributor.
Input parameters	**licenseTmpId** Temporal license identifier, returned by initLicenseDistributor. **AXUIDPrincipal** This is the AXUID of the principal (the distributor user). **diType** Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised. region If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done.
Output parameters	a String with the temporal distributor grant ID. This identifier is usable while the license is being created, and it will be used to assign the different distributable rights to the distributor with AddGrantforEndUser.

addGrantForEndUser	
Method	addGrantforEndUser
Description	addGrantforEndUser is the service that adds (one each time) the rights that a distributor can distribute. In other words, this function adds the rights that can be included in an EndUser License created by a specific distributor. This service has to be called as many times as different rights will be available in the future EndUser licenses. The different parameters allow introducing: right and the different conditions to be accomplished. The resource is established before in the addGrantforDistributor service.
Input parameters	**licenseTmpId** Temporal license identifier, returned by initLicenseDistributor. **distGrantId** Temporal grant identifier, returned by AddGrantforDistributor. **validityInterval** If this parameter is TRUE the right can be exercised within a time

	period. If it is FALSE it could be exercised always. **notBeforeIf** validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. country If validityRegion is TRUE, this parameter corresponds to the **country** where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created into the license. If the right has not been created, the returned value is 4XX:Error causes. If the right has been created normally, it returns 200:OK.

finaliseLicenseDistributor	
Method	finaliseLicenseDistributor
Description	finaliseLicenseDistributor finalises the license. This is the last service to be invoked in a license creation process. The service builds the licenses and, if it is correct, then stores it in the database.
Input parameters	licenseTmpId Temporal license identifier, returned by initLicenseDistributor.
Output parameters	String with the license identifier. This is a unique identifier of the license and can be used to retrieve a copy of the license

InitPAREndUser	
Method	InitPAREndUser
Description	InitPAREndUser initialises the creation of a PAR. This is the first function to be called in the process of an End User PAR creation. The service initPAREndUser returns the temporal identifier of the PAR. This identifier is usable while the PAR is being created. When the PAR is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	
Output	The temporal identifier of the PAR. This identifier is usable while the PAR is being created.

parameters	When the PAR is finished and stored, this identifier is not used any more

AddGrantPAREndUser	
Method	AddGrantPAREndUser
Description	AddGrantPAREndUser is the function that adds (one each time) the rights granted in a PAR. This service has to be called as many times as rights granted by the PAR. The different parameters allow introducing: the right, the resource over which the right will be exercised, the user who will obtain the right, and finally, the different conditions to be accomplished.
Input parameters	**PARTmpId** Temporal PAR identifier, returned by initPAREndUser. **diResource** Establishes that the resource will be referenced by an URI. f.e. http://www.musicserver.org/track1.mp3 If this parameter is TRUE, diReference has to be FALSE diType Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **right** The right that will be granted in the PAR. Can take the following values: adapt, delete, diminish, embed, enhance, enlarge, execute, install, modify, move, play, print, reduce, uninstall that correspond to rights described in "MPEG-21 multimedia extension". **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. fee If feeType is not 0, this parameter corresponds to the fee. currency If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different condition adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created and added to the PAR. If the right has not been created, the returned value is 4XX:Error causes.

	If the right has been correctly created, it returns 200:OK

finalisePAREndUser	
Method	finalisePAREndUser
Description	finalisePAREndUser finalises the PAR. This is the function to be invoked in a PAR reation process. The function builds the PAR and, if it is correct, then stores it in the database.
Input parameters	PARTmpId String with the Temporal PAR ID returned by initPAREndUser.
Output parameters	A String with the PAR identifier. This is unique identifier of the PAR and can be used to retrieve a copy of the PAR

InitPARDistributor	
Method	InitPARDistributor
Description	InitPARDistributor initialises the creation of a PAR. This is the first function to be called in the process of a Distributor PAR creation. This service receives information about the creator of the PAR. The function initPAREndUser returns the temporal identifier of the PAR. This identifier is usable while the PAR is being created. When the PAR is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	
Output parameters	The temporal identifier of the PAR. This identifier is usable while the PAR is being created. When the PAR is finished and stored, this identifier is not used any more

addGrantPARforDistributor	
Method	addGrantPARforDistributor
Description	addGrantPARforDistributor is the function that adds (one each time) the different rights for distributors and the distribution conditions for each one. The parameters established in this function affect only to the issue right (the one defining distribution). This function has to be called as many times as distributors the PAR has. The different parameters allow introducing: the distribution conditions, the content that will be distributed and the identification of the distributor.
Input parameters	**PARTmpId** Temporal PAR identifier, returned by initPARDistributor. **diType** Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS

	notAfter If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised. region If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done.
Output parameters	a String with the temporal distributor grant ID. This identifier is usable while the PAR is being created, and it will be used to assign the different distributable rights to the distributor with AddGrantforEndUser.

addGrantPARforEndUser	
Method	addGrantPARforEndUser
Description	addGrantPARforEndUser is the function that adds (one each time) the rights that a distributor can distribute. In other words, this function adds the rights that can be included in an EndUser PAR created by a specific distributor. This function has to be called as many times as different rights will be available in the future EndUser PARs. The different parameters allow introducing: right and the different conditions to be accomplished. The resource is established before in the addGrantforDistributor service.
Input parameters	**PARTmpId** Temporal PAR identifier, returned by initPARDistributor. **distGrantId** Temporal grant identifier, returned by AddGrantforDistributor. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBeforeIf** validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. country If validityRegion is TRUE, this parameter corresponds to the **country** where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised.

	feeType This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created into the PAR. If the right has not been created, the returned value is 4XX:Error causes. If the right has been created normally, it returns 200:OK.

finalisePARDistributor	
Method	finalisePARDistributor
Description	finalisePARDistributor finalises the PAR. This is the last function to be invoked in a PAR creation process. The service builds the PARs and, if it is correct, then stores it in the database.
Input parameters	PARTmpId Temporal PAR identifier, returned by initPARDistributor.
Output parameters	String with the PAR identifier. This is a unique identifier of the PAR and can be used to retrieve a copy of the PAR

verifyLicense	
Method	verifyLicense
Description	Verifies a license syntactically against the schemas defined within the license.
Input parameters	xsd:string license: the license to be verified
Output parameters	xsd:boolean: true if the license is correct, false if not.

verifyTemporalLicense	
Method	verifyTemporalLicense
Description	Verifies that the license generated by the user fulfils the initial desirables requirements of the user. For example, the user can verify that with this license he could exercise the desired action over the AXObject.
Input parameters	xsd:string license: the license to be verified xsd:string context: user conditions
Output parameters	xsd: string: Additional conditions that the user must fulfill.

registrationRequest	
Method	registrationRequest
Description	This function is used to a registration of an user in a certain domain
Input parameters	xsd: string domain : name of the domain ax: user
Output parameters	xsd: boolean : 0 means OK

unRegistrationRequest	
Method	unRegistrationRequest
Description	This function is used to an unregistration of an user in a certain domain
Input parameters	xsd: string userID : name of the domain xsd: string domain : name of the domain
Output parameters	xsd: boolean : 0 means OK

getDomainsRegistered	
Method	getDomainsRegistered
Description	This method returns the domain a user is registered to.
Input parameters	String UserId
Output parameters	List of Strings with the domains where the user is registered

insertActionLog	
Method	insertActionLog
Description	Stores the given action log associated to an AXMEDIS object identifier, the object version and the protection stamp.
Input parameters	axoid AXMEDIS identification of the object objectversion Version of the object protectionstamp Protection of the object actionlog ActionLog to be inserted
Output parameters	true on success

retrieveActionLogs	
Method	retrieveActionLogs
Description	This method retrieves all the action logs inside the local cache info when the user connects to the PMS server in order to verify and synchronize the actions performed off-line with the previously performed actions
Input parameters	None
Output parameters	vector with the action logs.

deleteCacheContent	
Method	deleteCacheContent
Description	This method is for deleting the contents of the cache. It can be used when the tool cannot be verified because of illegal manipulation.
Input parameters	None
Output parameters	Integer Value = 0 means all ok, otherwise cache error (database error)

clearActionLogs	
Method	clearActionLogs
Description	Deletes action logs from the cache, after positive authorisation of the user in the connected environment
Input	None

parameters	
Output parameters	Integer Value = 0 means all ok, otherwise cache error (database error)

verifyUser	
Method	verifyUser
Description	This method is called by the Protection Processor and reaches AXCV through PMS Server. It can be used to verify the status of a user, optionally inside a domain. It verifies if the user is registered in the specified domain (if present) and checks that the user status and registration deadline are valid, so that the user can still use the AXMEDIS tools and the AXMEDIS framework.
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID)
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: Verification OK -1: invalid AXID -2: user is not registered -3: user is blocked -4: user domain mismatch -5: user registration deadline expired When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur, but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the deadline has expired or not).

certify	
Method	certify
Description	This method is called by the Protection Processor and reaches AXCV through PMS Server. It is used to certify that the original tool has not been modified and to activate it. It creates a new entry in the CerTools table of the AXCS database, which associates the user, tool and device and returns to the Protection Processor an activation code, a tool identifier and a PKCS12 structure with the tool certificate and private key issued by AXMEDIS CA (EJBCA).
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axrtid**: identifier of the registered AXMEDIS tool xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool xsd:string **regDeadline**: registration deadline of the installed tool.
Output parameters	CertificationResult complex type formed by sequence of: xsd:string **axtid**, the identifier of the installed tool associated to a user and device. xsd:int **certificationResult**, which indicates the result of the certification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: tool not registered (RegTools table) -7: registered tool is blocked -8: received tool deadline exceeds registered tool deadline (user and tool have been blocked)

	-9: received tool deadline has expired -10: registered tool fingerprint mismatch. Tool has been manipulated (user and tool have been blocked) -11: user-tool-device had already been certified. New tool certificate should be created -20: error updating user status in database -21: error inserting new entry in CerTools table -22: error in AXSupervisor when communicating with database -30: internal AXCV error xsd:string **enablingCode**, the tool activation code sent to the Protection Processor. byte[] **toolBase64PKCS12**, PKCS12 structure bytes encoded in Base 64. It includes the tool certificate signed by the AXCS CA Root Certificate and tool private key together and protected with a password. If the unrestricted policy files for Sun JCE were available at the server (default configuration), the password will be the full AXMEDIS AXID. Otherwise, the password will be the first 8 characters of the AXMEDIS AXID. It proves that an AXMEDIS tool has been certified and can be used in the AXMEDIS framework When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur, but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the registered tool is blocked or not, or if the tool fingerprint did match or not).

verify	
Method	verify
Description	This method is called by the Protection Processor and reaches AXCV through PMS Server. It is used to verify that the tool installed on a device has neither been modified nor blocked, that the user is not blocked and that the registered tool is not blocked. It is also responsible for resynchronizing the offline tool operation through AXMEDIS Supervisor (AXS).
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a device). byte[] **toolFingerprintDigest**: md5 hash of the full fingerprint (software and hardware parts) of the installed tool.
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: AXTID does not exist -7: installed (and certified) tool is blocked -8: tool deadline has expired -9: toolFingerprintDigest (toolFingerprint hash) mismatch -10: toolFingerprint mismatch (user and tool have been blocked) -11: registered tool is blocked -12: user has been blocked and installed tool has been blocked again -13: tool has been blocked -20: error updating user status in database -21: error updating tool status in database

	-22: error updating LastFPPA in database -23: error retrieving regtool data from database -24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog -25: error in AXSupervisor when communicating with AXCS accounting database in storeSID -30: internal AXCV error xsd:int **storeListActionLogResult**, which indicates the result of the storage of the action logs, according to the following numeration: 0: ActionLog(s) has been stored: it includes the case of empty list -1: ActionLog(s) has been stored: tool should have been already blocked -2: ActionLog(s) has been stored: tool operation history hash (LastFPPA) is not consistent -3: ActionLog(s) has not been stored: error in AXSupervisor when communicating with AXCS database -4: ActionLog(s) has not been stored: input actionLog(s) do not refer to the same AXTID -5: ActionLog(s) has not been stored: input actionLog(s) have some non-nillable null fields -6: ActionLog(s) has not been stored: user or tool data unsuccessfully verified by AXCV When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur, but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the received tool deadline has expired or not).

reverify	
Method	reverify
Description	This method is similar to verify method (see previous). It must be called when the verify method fails because of the tool fingerprint hash doesn't match (error code: –12) to perform a new verification with the full fingerprint. Thus, the reverify method has the same input parameters as the verify method except the full tool fingerprint, which has to be sent instead of the hash.
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string axtid: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool.
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: AXTID does not exist -7: installed (and certified) tool is blocked -8: tool deadline has expired

	-9: toolFingerprintDigest (toolFingerprint hash) mismatch -10: toolFingerprint mismatch (user and tool have been blocked) -11: registered tool is blocked -12: user has been blocked and installed tool has been blocked again -13: tool has been blocked -20: error updating user status in database -21: error updating tool status in database -22: error updating LastFPPA in database -23: error retrieving regtool data from database -24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog -25: error in AXSupervisor when communicating with AXCS accounting database in storeSID -30: internal AXCV error xsd:int **storeListActionLogResult**, which indicates the result of the storage of the action logs, according to the following numeration: 0: ActionLog(s) has been stored: it includes the case of empty list -1: ActionLog(s) has been stored: tool should have been already blocked -2: ActionLog(s) has been stored: tool operation history hash (LastFPPA) is not consistent -3: ActionLog(s) has not been stored: error in AXSupervisor when communicating with AXCS database -4: ActionLog(s) has not been stored: input actionLog(s) do not refer to the same AXTID -5: ActionLog(s) has not been stored: input actionLog(s) have some non-nillable null fields -6: ActionLog(s) has not been stored: user or tool data unsuccessfully verified by AXCV When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur, but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the received tool deadline has expired or not).

getProtectionInfo	
Method	getProtectionInfo
Description	This method is used to retrieve the protection information related to an object from the Objects Table of the AXCS Objects ID Database.
Input parameters	The following fields of the Objects table in the AXCS Objects ID database: type="xsd:string" **AXOID**, AXMEDIS object identifier type="xsd:string" **ObjectVersion**, object version type="xsd:string" **ProtectionStamp**, protection stamp
Output parameters	type="xsd:string" **ProtectionInfo**, protection information associated to the object or a "wrong_object" result if there is no ProtectionInfo for the requested object

UpdateProtectionInfo	
Method	UpdateProtectionInfo
Description	This method is used to insert or update the protection information related to an AXMEDIS object in the Objects Table of the AXCS Objects ID Database.
Input parameters	The following fields of the Objects table in the AXCS Objects ID database: type="xsd:string" **AXOID**, AXMEDIS object identifier type="xsd:string" **ObjectVersion**, object version

	type="xsd:string" **ProtectionStamp**, protection stamp type="xsd:string" **ProtectionInfo**, protection information to be updated type="xsd:int" **Update**, denotes if the protection info must be inserted (0) or updated (1)
Output parameters	type="xsd:int" **updateProtectionInfoReturn**, which indicates the result of this request, according to the following numeration: 0: OK -1: there is not any entry in AXCS Objects database that matches the input information -2: error in AXSupervisor when updating ProtectionInfo in AXCS Objects database

5 Protection Manager Support Domain Factory (UPC)

Module/Tool Profile		
Protection Manager Support Domain Factory (PMS Domain Factory)		
Responsible Name	Rubén Barrio	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Not Implemented	
Status of the implementation		
Executable or Library/module (Support)	Executable, Web service	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	N/A	
Reference to the AXFW location of the demonstrator executable tool for internal download	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	N/A	
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

5.1 General Description of the Module

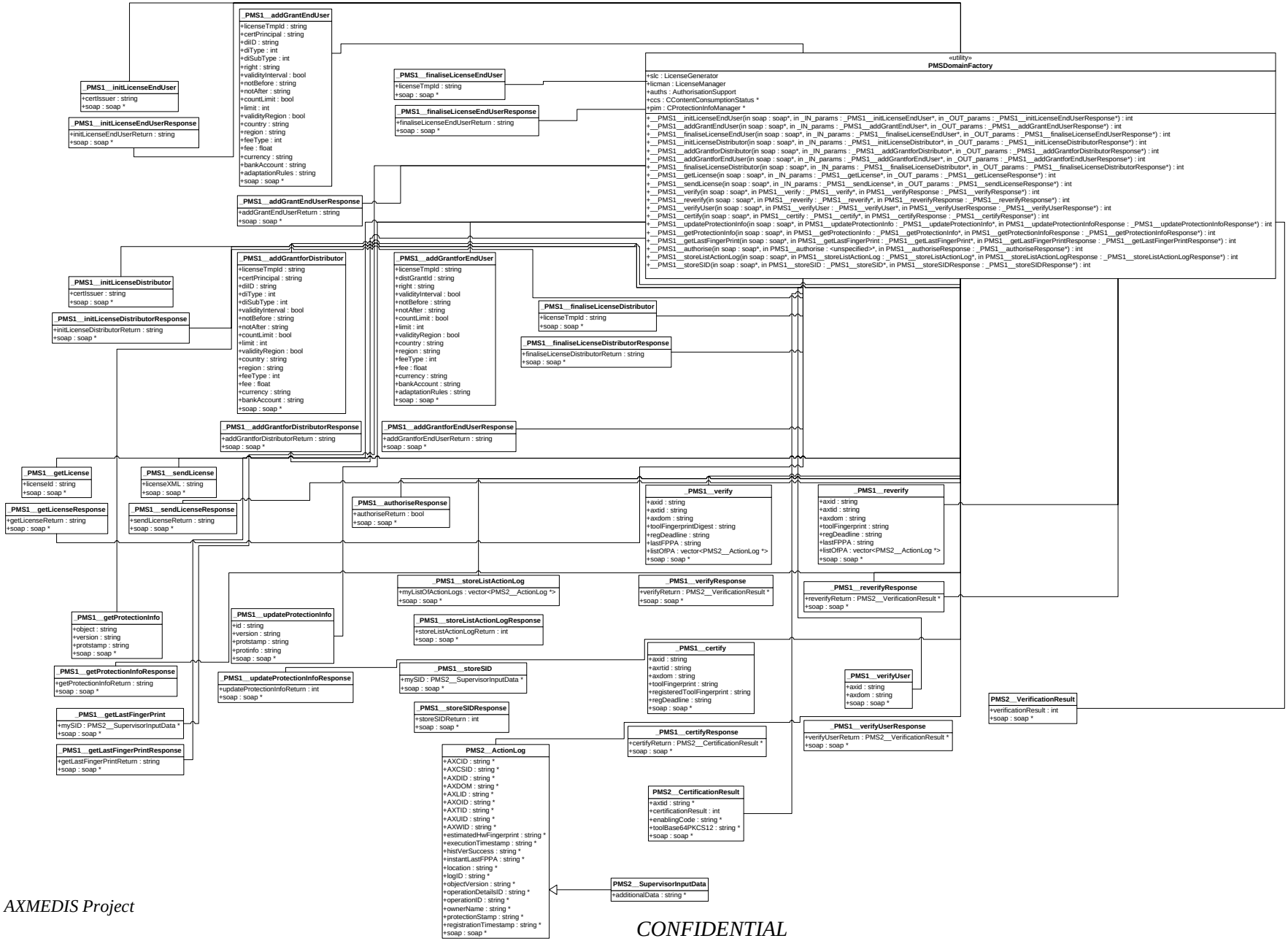
Protection Manager Support Domain Factory provides the protection needed for a set of PMS Clients. It has connection with AXMEDIS Certifier and Supervisor, in order to check that users only perform the actions they are allowed to. In this section, the general functionality of this module is explained. In next sections, the modules forming part of PMS Domain Factory are explained in detail.

PMS Domain Factory	
Methods	***Description***
authorise	This function authorises AXMEDIS users to perform actions over AXMEDIS objects. It proves that a user has the appropriate license that grants him to exercise a right over a resource if the conditions are met based on the execution context of the client.
getLicense	This function retrieves the licenses stored in the license database. It retrieves the license with the licenseID set as a parameter, or the licenses associated to an AXOID.
sendLicense	This function stores a license in the license database.
InitLicenseEndUser	InitLicenseEndUser initialises the creation of a license. This is the first web service to be called in the process of an End User License creation. The service initLicenseEndUser returns the temporal identifier of the license. This identifier is usable while the license is being created.

	When the license is finished and stored this identifier is not used any more and it is deleted from the database.
AddGrantEndUser	AddGrantEndUser is the web service that adds (one each time) the rights granted in a license. This service has to be called as many times as rights granted by the license. The different parameters allow introducing: the right, the resource over which the right will be exercised, the user who will obtain the right, and finally, the different conditions to be accomplished.
finaliseLicenseEndUser	finaliseLicenseEndUser finalises the license. This is the last service to be invoked in a license reation process. The service builds the license and, if it is correct, then stores it in the database.
InitLicenseDistributor	InitLicenseDistributor initialises the creation of a license. This is the first web service to be called in the process of a Distributor License creation. This service receives information about the creator of the license. The service initLicenseEndUser returns the temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored this identifier is not used any more and it is deleted from the database.
addGrantforDistributor	addGrantforDistributor is the service that adds (one each time) the different rights for distributors and the distribution conditions for each one. The parameters established in this service affect only to the issue right (the one defining distribution). This service has to be called as many times as distributors the license has. The different parameters allow introducing: the distribution conditions, the content that will be distributed and the identification of the distributor.
addGrantforEndUser	addGrantforEndUser is the service that adds (one each time) the rights that a distributor can distribute. In other words, this function adds the rights that can be included in an EndUser License created by a specific distributor. This service has to be called as many times as different rights will be available in the future EndUser licenses. The different parameters allow introducing: right and the different conditions to be accomplished. The resource is established before in the addGrantforDistributor service.
finaliseLicenseDistributor	finaliseLicenseDistributor finalises the license. This is the last service to be invoked in a license creation process. The service builds the licenses and, if it is correct, then stores it in the database.
storeListActionLog	This method is used by PMS Client to store through Supervisor a list of Action Logs. When a user has performed some off-line actions, if PMS Client gets connection to the system, it calls verify method, which reaches AXCV through PMS Server in order to resynchronize the actions that are stored in the local cache.
getLastFingerprint	This method is used by PMS Client to request Supervisor the Last Fingerprint of a user or an object or a tool in order to certify or verify any user.
verifyUser	This method is called by PMS Client and reaches AXCV through PMS Server. It can be used to verify the status of a user, optionally inside a domain. It verifies if the user is registered in the specified domain (if

	present) and checks that the user status and registration deadline are valid, so that the user can still use the AXMEDIS tools and the AXMEDIS framework.
certify	This method is called by PMS Client and reaches AXCV through PMS Server. It is used to certify that the original tool has not been modified and to activate it. It creates a new entry in the CerTools table of the AXCS database which associates the user, tool and device and returns to the Protection Processor an activation code, a tool identifier and a PKCS12 structure with the tool certificate and private key issued by AXMEDIS CA (EJBCA).
verify	This method is called by PMS Client and reaches AXCV through PMS Server. It is used to verify that the tool installed on a device has neither been modified nor blocked, that the user is not blocked and that the registered tool is not blocked. It is also responsible for resynchronizing the offline tool operation through AXMEDIS Supervisor (AXS).
reverify	This method is similar to verify method (see previous). It must be called when the verify method fails because of the tool fingerprint hash doesn't match (error code: –9) to perform a new verification with the full fingerprint. Thus, the reverify method has the same input parameters as the verify method except the full tool fingerprint, which has to be sent instead of the hash.
getProtectionInfo	This method is called by PMS Client and is used to retrieve the protection information related to an object from the Objects Table of the AXCS Objects ID Database.
updateProtectionInfo	This method is called by PMS Client and is used to insert or update the protection information related to an AXMEDIS object in the Objects Table of the AXCS Objects ID Database.

5.2 Module Design in terms of Classes


«utility»
PMSDomainFactory
+slc : LicenseGenerator
+licman : LicenseManager
+auths : AuthorisationSupport
+ccs : CContentConsumptionStatus *
+pim : CProtectionInfoManager *
+__PMS1__initLicenseEndUser(in soap : soap*, in _IN_params : _PMS1__initLicenseEndUser*, in _OUT_params : _PMS1__initLicenseEndUserResponse*) : int
+__PMS1__addGrantEndUser(in soap : soap*, in _IN_params : _PMS1__addGrantEndUser*, in _OUT_params : _PMS1__addGrantEndUserResponse*) : int
+__PMS1__finaliseLicenseEndUser(in soap : soap*, in _IN_params : _PMS1__finaliseLicenseEndUser*, in _OUT_params : _PMS1__finaliseLicenseEndUserResponse*) : int
+__PMS1__initLicenseDistributor(in soap : soap*, in _IN_params : _PMS1__initLicenseDistributor*, in _OUT_params : _PMS1__initLicenseDistributorResponse*) : int
+__PMS1__addGrantforDistributor(in soap : soap*, in _IN_params : _PMS1__addGrantforDistributor*, in _OUT_params : _PMS1__addGrantforDistributorResponse*) : int
+__PMS1__addGrantforEndUser(in soap : soap*, in _IN_params : _PMS1__addGrantforEndUser*, in _OUT_params : _PMS1__addGrantforEndUserResponse*) : int
+__PMS1__finaliseLicenseDistributor(in soap : soap*, in _IN_params : _PMS1__finaliseLicenseDistributor*, in _OUT_params : _PMS1__finaliseLicenseDistributorResponse*) : int
+__PMS1__getLicense(in soap : soap*, in _IN_params : _PMS1__getLicense*, in _OUT_params : _PMS1__getLicenseResponse*) : int
+__PMS1__sendLicense(in soap : soap*, in _IN_params : _PMS1__sendLicense*, in _OUT_params : _PMS1__sendLicenseResponse*) : int
+__PMS1__verify(in soap : soap*, in PMS1__verify : _PMS1__verify*, in PMS1__verifyResponse : _PMS1__verifyResponse*) : int
+__PMS1__reverify(in soap : soap*, in PMS1__reverify : _PMS1__reverify*, in PMS1__reverifyResponse : _PMS1__reverifyResponse*) : int
+__PMS1__verifyUser(in soap : soap*, in PMS1__verifyUser : _PMS1__verifyUser*, in PMS1__verifyUserResponse : _PMS1__verifyUserResponse*) : int
+__PMS1__certify(in soap : soap*, in PMS1__certify : _PMS1__certify*, in PMS1__certifyResponse : _PMS1__certifyResponse*) : int
+__PMS1__updateProtectionInfo(in soap : soap*, in PMS1__updateProtectionInfo : _PMS1__updateProtectionInfo*, in PMS1__updateProtectionInfoResponse : _PMS1__updateProtectionInfoResponse*) : int
+__PMS1__getProtectionInfo(in soap : soap*, in PMS1__getProtectionInfo : _PMS1__getProtectionInfo*, in PMS1__getProtectionInfoResponse : _PMS1__getProtectionInfoResponse*) : int
+__PMS1__getLastFingerPrint(in soap : soap*, in PMS1__getLastFingerPrint : _PMS1__getLastFingerPrint*, in PMS1__getLastFingerPrintResponse : _PMS1__getLastFingerPrintResponse*) : int
+__PMS1__authorise(in soap : soap*, in PMS1__authorise : <unspecified>*, in PMS1__authoriseResponse : _PMS1__authoriseResponse*) : int
+__PMS1__storeListActionLog(in soap : soap*, in PMS1__storeListActionLog : _PMS1__storeListActionLog*, in PMS1__storeListActionLogResponse : _PMS1__storeListActionLogResponse*) : int
+__PMS1__storeSID(in soap : soap*, in PMS1__storeSID : _PMS1__storeSID*, in PMS1__storeSIDResponse : _PMS1__storeSIDResponse*) : int
_PMS1__addGrantEndUser
+licenseTmpId : string
+certPrincipal : string
+diID : string
+diType : int
+diSubType : int
+right : string
+validityInterval : bool
+notBefore : string
+notAfter : string
+countLimit : bool
+limit : int
+validityRegion : bool
+country : string
+region : string
+feeType : int
+fee : float
+currency : string
+bankAccount : string
+adaptationRules : string
+soap : soap *
_PMS1__initLicenseEndUser
+certIssuer : string
+soap : soap *
_PMS1__initLicenseEndUserResponse
+initLicenseEndUserReturn : string
+soap : soap *
_PMS1__finaliseLicenseEndUser
+licenseTmpId : string
+soap : soap *
_PMS1__finaliseLicenseEndUserResponse
+finaliseLicenseEndUserReturn : string
+soap : soap *
_PMS1__addGrantEndUserResponse
+addGrantEndUserReturn : string
+soap : soap *
_PMS1__addGrantforDistributor
+licenseTmpId : string
+certPrincipal : string
+diID : string
+diType : int
+diSubType : int
+validityInterval : bool
+notBefore : string
+notAfter : string
+countLimit : bool
+limit : int
+validityRegion : bool
+country : string
+region : string
+feeType : int
+fee : float
+currency : string
+bankAccount : string
+soap : soap *
_PMS1__addGrantforEndUser
+licenseTmpId : string
+distGrantId : string
+right : string
+validityInterval : bool
+notBefore : string
+notAfter : string
+countLimit : bool
+limit : int
+validityRegion : bool
+country : string
+region : string
+feeType : int
+fee : float
+currency : string
+bankAccount : string
+adaptationRules : string
+soap : soap *
_PMS1__initLicenseDistributor
+certIssuer : string
+soap : soap *
_PMS1__initLicenseDistributorResponse
+initLicenseDistributorReturn : string
+soap : soap *
_PMS1__finaliseLicenseDistributor
+licenseTmpId : string
+soap : soap *
_PMS1__finaliseLicenseDistributorResponse
+finaliseLicenseDistributorReturn : string
+soap : soap *
_PMS1__addGrantforDistributorResponse
+addGrantforDistributorReturn : string
+soap : soap *
_PMS1__addGrantforEndUserResponse
+addGrantforEndUserReturn : string
+soap : soap *
_PMS1__getLicense
+licenseId : string
+soap : soap *
_PMS1__sendLicense
+licenseXML : string
+soap : soap *
_PMS1__getLicenseResponse
+getLicenseReturn : string
+soap : soap *
_PMS1__sendLicenseResponse
+sendLicenseReturn : string
+soap : soap *
_PMS1__authoriseResponse
+authoriseReturn : bool
+soap : soap *
_PMS1__verify
+axid : string
+axtid : string
+axdom : string
+toolFingerprintDigest : string
+regDeadline : string
+lastFPPA : string
+listOfPA : vector<PMS2__ActionLog *>
+soap : soap *
_PMS1__reverify
+axid : string
+axtid : string
+axdom : string
+toolFingerprint : string
+regDeadline : string
+lastFPPA : string
+listOfPA : vector<PMS2__ActionLog *>
+soap : soap *
_PMS1__storeListActionLog
+myListOfActionLogs : vector<PMS2__ActionLog *>
+soap : soap *
_PMS1__verifyResponse
+verifyReturn : PMS2__VerificationResult *
+soap : soap *
_PMS1__reverifyResponse
+reverifyReturn : PMS2__VerificationResult *
+soap : soap *
_PMS1__getProtectionInfo
+object : string
+version : string
+protstamp : string
+soap : soap *
_PMS1__updateProtectionInfo
+id : string
+version : string
+protstamp : string
+protInfo : string
+soap : soap *
_PMS1__storeListActionLogResponse
+storeListActionLogReturn : int
+soap : soap *
_PMS1__certify
+axid : string
+axrtid : string
+axdom : string
+toolFingerprint : string
+registeredToolFingerprint : string
+regDeadline : string
+soap : soap *
_PMS1__getProtectionInfoResponse
+getProtectionInfoReturn : string
+soap : soap *
_PMS1__updateProtectionInfoResponse
+updateProtectionInfoReturn : int
+soap : soap *
_PMS1__storeSID
+mySID : PMS2__SupervisorInputData *
+soap : soap *
_PMS1__verifyUser
+axid : string
+axdom : string
+soap : soap *
PMS2__VerificationResult
+verificationResult : int
+soap : soap *
_PMS1__getLastFingerPrint
+mySID : PMS2__SupervisorInputData *
+soap : soap *
_PMS1__getLastFingerPrintResponse
+getLastFingerPrintReturn : string
+soap : soap *
_PMS1__storeSIDResponse
+storeSIDReturn : int
+soap : soap *
_PMS1__certifyResponse
+certifyReturn : PMS2__CertificationResult *
+soap : soap *
_PMS1__verifyUserResponse
+verifyUserReturn : PMS2__VerificationResult *
+soap : soap *
PMS2__ActionLog
+AXCID : string *
+AXCSID : string *
+AXDID : string *
+AXDOM : string *
+AXLID : string *
+AXOID : string *
+AXTID : string *
+AXUID : string *
+AXWID : string *
+estimatedHwFingerprint : string *
+executionTimestamp : string *
+histVerSuccess : string *
+instantLastFPPA : string *
+location : string *
+logID : string *
+objectVersion : string *
+operationDetailsID : string *
+operationID : string *
+ownerName : string *
+protectionStamp : string *
+registrationTimestamp : string *
+soap : soap *
PMS2__CertificationResult
+axtid : string *
+certificationResult : int
+enablingCode : string *
+toolBase64PKCS12 : string *
+soap : soap *
PMS2__SupervisorInputData
+additionalData : string *

5.3 Formal description of PMS Domain Factory

Authorise	
Method	Authorise
Description	This function authorises AXMEDIS users to perform actions over AXMEDIS objects. It proves that a user has the appropriate license that grants him to exercise a right over a resource if the conditions are met based on the execution context of the client.
Input parameters	String userID: User id to be authorised String action: action to be authorised String resource: resource to be authorised contextData context: context of the client to be authorised ActionLog constructingAL: Actionlog of the authorisation with the “client side” parameters fulfilled
Output parameters	Integer:

getLicense	
Method	getLicense
Description	This function retrieves the licenses stored in the license database. It retrieves the license with the licenseID set as a parameter, or the licenses associated to an AXOID.
Input parameters	String licenseId: License Id
Output parameters	String, the license in XML

sendLicense	
Method	sendLicense
Description	This function stores a license in the license database.
Input parameters	String licenseXML: the license in XML format
Output parameters	String: result of the operation

InitLicenseEndUser	
Method	InitLicenseEndUser
Description	InitLicenseEndUser initialises the creation of a license. This is the first web service to be called in the process of an End User License creation. The service initLicenseEndUser returns the temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	IssuerAXUID String with the Issuer AXUID (creator of the license).
Output parameters	

AddGrantEndUser	
Method	AddGrantEndUser
Description	AddGrantEndUser is the web service that adds (one each time) the rights granted in a license. This service has to be called as many times as rights granted by the license. The different

	parameters allow introducing: the right, the resource over which the right will be exercised, the user who will obtain the right, and finally, the different conditions to be accomplished.
Input parameters	**licenseTmpId** Temporal license identifier, returned by initLicenseEndUser. **AXUIDPrincipal** This is the AXUID of the user (user of the license). **diResource** Establishes that the resource will be referenced by an URI. f.e. http://www.musicserver.org/track1.mp3 If this parameter is TRUE, diReference has to be FALSE diType Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **right** The right that will be granted in the license. Can take the following values: adapt, delete, diminish, embed, enhance, enlarge, execute, install, modify, move, play, print, reduce, uninstall that correspond to rights described in "MPEG-21 multimedia extension". **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. fee If feeType is not 0, this parameter corresponds to the fee. currency If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different condition adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created and added to the license. If the right has not been created, the returned value is 4XX:Error causes. If the right has been correctly created, it returns 200:OK

finaliseLicenseEndUser	
Method	finaliseLicenseEndUser
Description	finaliseLicenseEndUser finalises the license. This is the last service to be invoked in a license reation process.

	The service builds the license and, if it is correct, then stores it in the database.
Input parameters	licenseTmpId String with the Temporal license ID returned by initLicenseEndUser.
Output parameters	A String with the license identifier. This is unique identifier of the license and can be used to retrieve a copy of the license

InitLicenseDistributor	
Method	InitLicenseDistributor
Description	InitLicenseDistributor initialises the creation of a license. This is the first web service to be called in the process of a Distributor License creation. This service receives information about the creator of the license. The service initLicenseEndUser returns the temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	IssuerAXUID String with the Issuer AXUID (normally creator of the content or rights owner).
Output parameters	The temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored, this identifier is not used any more

addGrantforDistributor	
Method	addGrantforDistributor
Description	addGrantforDistributor is the service that adds (one each time) the different rights for distributors and the distribution conditions for each one. The parameters established in this service affect only to the issue right (the one defining distribution). This service has to be called as many times as distributors the license has. The different parameters allow introducing: the distribution conditions, the content that will be distributed and the identification of the distributor.
Input parameters	**licenseTmpId** Temporal license identifier, returned by initLicenseDistributor. **AXUIDPrincipal** This is the AXUID of the principal (the distributor user). **diType** Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised.

	region If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done.
Output parameters	a String with the temporal distributor grant ID. This identifier is usable while the license is being created, and it will be used to assign the different distributable rights to the distributor with AddGrantforEndUser.

addGrantforEndUser	
Method	addGrantforEndUser
Description	addGrantforEndUser is the service that adds (one each time) the rights that a distributor can distribute. In other words, this function adds the rights that can be included in an EndUser License created by a specific distributor. This service has to be called as many times as different rights will be available in the future EndUser licenses. The different parameters allow introducing: right and the different conditions to be accomplished. The resource is established before in the addGrantforDistributor service.
Input parameters	**licenseTmpId** Temporal license identifier, returned by initLicenseDistributor. **distGrantId** Temporal grant identifier, returned by AddGrantforDistributor. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBeforeIf** validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. country If validityRegion is TRUE, this parameter corresponds to the **country** where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter

	corresponds to the different adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created into the license. If the right has not been created, the returned value is 4XX:Error causes. If the right has been created normally, it returns 200:OK.

finaliseLicenseDistributor	
Method	finaliseLicenseDistributor
Description	finaliseLicenseDistributor finalises the license. This is the last service to be invoked in a license creation process. The service builds the licenses and, if it is correct, then stores it in the database.
Input parameters	licenseTmpId Temporal license identifier, returned by initLicenseDistributor.
Output parameters	String with the license identifier. This is a unique identifier of the license and can be used to retrieve a copy of the license

verifyUser	
Method	verifyUser
Description	This method is called by PMS Client and reaches AXCV through PMS Server. It can be used to verify the status of a user, optionally inside a domain. It verifies if the user is registered in the specified domain (if present) and checks that the user status and registration deadline are valid, so that the user can still use the AXMEDIS tools and the AXMEDIS framework.
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axdom**: AXMEDIS domain of certified user (if any)
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: Verification OK -1: invalid AXID -2: user is not registered -3: user is blocked -4: user domain mismatch -5: user registration deadline expired When an error code *x* is returned, it means that all the possible errors *y*, $x<y<0$ did not occur, but all possible errors $y<x$ have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the deadline has expired or not).

certify	
Method	certify
Description	This method is called by PMS Client and reaches AXCV through PMS Server. It is used to certify that the original tool has not been modified and to activate it. It creates a new entry in the CerTools table of the AXCS database, which associates the user, tool and device and returns to the Protection Processor an activation code, a tool identifier and a PKCS12 structure with the tool certificate and private key issued by AXMEDIS CA (EJBCA).
Input parameters	xsd:string axid: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axrtid**: identifier of the registered AXMEDIS tool xsd:string **axdom**: domain where the user is registered. xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool xsd:string **regDeadline**: registration deadline of the installed tool.
Output	CertificationResult complex type formed by sequence of:

parameters	xsd:string **axtid**, the identifier of the installed tool associated to a user and device. xsd:int **certificationResult**, which indicates the result of the certification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: tool not registered (RegTools table) -7: registered tool is blocked -8: received tool deadline exceeds registered tool deadline (user and tool have been blocked) -9: received tool deadline has expired -10: registered tool fingerprint mismatch. Tool has been manipulated (user and tool have been blocked) -11: user-tool-device had already been certified. New tool certificate should be created -20: error updating user status in database -21: error inserting new entry in CerTools table -22: error in AXSupervisor when communicating with database -30: internal AXCV error xsd:string **enablingCode**, the tool activation code sent to the Protection Processor. byte[] **toolBase64PKCS12**, PKCS12 structure bytes encoded in Base 64. It includes the tool certificate signed by the AXCS CA Root Certificate and tool private key together and protected with a password. If the unrestricted policy files for Sun JCE were available at the server (default configuration), the password will be the full AXMEDIS AXID. Otherwise, the password will be the first 8 characters of the AXMEDIS AXID. It proves that an AXMEDIS tool has been certified and can be used in the AXMEDIS framework When an error code *x* is returned, it means that all the possible errors *y*, $x<y<0$ did not occur, but all possible errors $y<x$ have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the registered tool is blocked or not, or if the tool fingerprint did match or not).

verify	
Method	verify
Description	This method is called by PMS Client and reaches AXCV through PMS Server. It is used to verify that the tool installed on a device has neither been modified nor blocked, that the user is not blocked and that the registered tool is not blocked. It is also responsible for resynchronizing the offline tool operation through AXMEDIS Supervisor (AXS).
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axdom**: domain where the user is registered. byte[] **toolFingerprintDigest**: SHA1 hash of the relevant data of hash of the full fingerprint (software and hardware parts) of the installed tool. byte[] **LastFPPA**: fingerprint of the history of the operations performed during the offline operation. tns2:ActionLog **listOfPA**: Array of ActionLogs, which is a complex type defined in AXMEDIS Supervisor, including the actions performed during the offline operation.
Output	VerificationResult complex type formed by sequence of:

parameters	xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: AXTID does not exist -7: installed (and certified) tool is blocked -8: tool deadline has expired -9: toolFingerprintDigest (toolFingerprint hash) mismatch -10: toolFingerprint mismatch (user and tool have been blocked) -11: registered tool is blocked -12: user has been blocked and installed tool has been blocked again -13: tool has been blocked -20: error updating user status in database -21: error updating tool status in database -22: error updating LastFPPA in database -23: error retrieving regtool data from database -24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog -25: error in AXSupervisor when communicating with AXCS accounting database in storeSID -30: internal AXCV error xsd:int **storeListActionLogResult**, which indicates the result of the storage of the action logs, according to the following numeration: 0: ActionLog(s) has been stored: it includes the case of empty list -1: ActionLog(s) has been stored: tool should have been already blocked -2: ActionLog(s) has been stored: tool operation history hash (LastFPPA) is not consistent -3: ActionLog(s) has not been stored: error in AXSupervisor when communicating with AXCS database -4: ActionLog(s) has not been stored: input actionLog(s) do not refer to the same AXTID -5: ActionLog(s) has not been stored: input actionLog(s) have some non-nillable null fields -6: ActionLog(s) has not been stored: user or tool data unsuccessfully verified by AXCV When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur, but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the received tool deadline has expired or not).

reverify	
Method	reverify
Description	This method is similar to verify method (see previous). It must be called when the verify method fails because of the tool fingerprint hash doesn't match (error code: –9) to perform a new verification with the full fingerprint. Thus, the reverify method has the same input parameters as the verify method except the full tool fingerprint, which has to be sent instead of the hash.
Input	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID,

parameters	AXDID, AXCSID or AXTPID) xsd:string axtid: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axdom**: domain where the user is registered. xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool. byte[] **LastFPPA**: fingerprint of the history of the operations performed during the offline operation. tns2:ActionLog **listOfPA**: Array of ActionLogs, which is a complex type defined in AXMEDIS Supervisor, including the actions performed during the offline operation.
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: AXTID does not exist -7: installed (and certified) tool is blocked -8: tool deadline has expired -9: toolFingerprintDigest (toolFingerprint hash) mismatch -10: toolFingerprint mismatch (user and tool have been blocked) -11: registered tool is blocked -12: user has been blocked and installed tool has been blocked again -13: tool has been blocked -20: error updating user status in database -21: error updating tool status in database -22: error updating LastFPPA in database -23: error retrieving regtool data from database -24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog -25: error in AXSupervisor when communicating with AXCS accounting database in storeSID -30: internal AXCV error xsd:int **storeListActionLogResult**, which indicates the result of the storage of the action logs, according to the following numeration: 0: ActionLog(s) has been stored: it includes the case of empty list -1: ActionLog(s) has been stored: tool should have been already blocked -2: ActionLog(s) has been stored: tool operation history hash (LastFPPA) is not consistent -3: ActionLog(s) has not been stored: error in AXSupervisor when communicating with AXCS database -4: ActionLog(s) has not been stored: input actionLog(s) do not refer to the same AXTID -5: ActionLog(s) has not been stored: input actionLog(s) have some non-nillable null fields -6: ActionLog(s) has not been stored: user or tool data unsuccessfully verified by AXCV When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur,

	but all possible errors *y<x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the received tool deadline has expired or not).

getProtectionInfo	
Method	getProtectionInfo
Description	This method is called by PMS Client and is used to retrieve the protection information related to an object from the Objects Table of the AXCS Objects ID Database.
Input parameters	The following fields of the Objects table in the AXCS Objects ID database: type="xsd:string" **AXOID**, AXMEDIS object identifier type="xsd:string" **ObjectVersion**, object version type="xsd:string" **ProtectionStamp**, protection stamp
Output parameters	type="xsd:string" **ProtectionInfo**, protection information associated to the object or a "wrong_object" result if there is no ProtectionInfo for the requested object

UpdateProtectionInfo	
Method	UpdateProtectionInfo
Description	This method is called by PMS Client and is used to insert or update the protection information related to an AXMEDIS object in the Objects Table of the AXCS Objects ID Database.
Input parameters	The following fields of the Objects table in the AXCS Objects ID database: type="xsd:string" **AXOID**, AXMEDIS object identifier type="xsd:string" **ObjectVersion**, object version type="xsd:string" **ProtectionStamp**, protection stamp type="xsd:string" **ProtectionInfo**, protection information to be updated type="xsd:int" **Update**, denotes if the protection info must be inserted (0) or updated (1)
Output parameters	type="xsd:int" **updateProtectionInfoReturn**, which indicates the result of this request, according to the following numeration: 0: OK -1: there is not any entry in AXCS Objects database that matches the input information -2: error in AXSupervisor when updating ProtectionInfo in AXCS Objects database

6 Protection Manager Support Domain Home (UPC)

Module/Tool Profile		
Protection Manager Support Domain Home (PMS Domain Home)		
Responsible Name	Rubén Barrio	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Not Implemented	
Status of the implementation		
Executable or Library/module (Support)	Executable, Web service	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	N/A	
Reference to the AXFW location of the demonstrator executable tool for internal download	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	N/A	
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

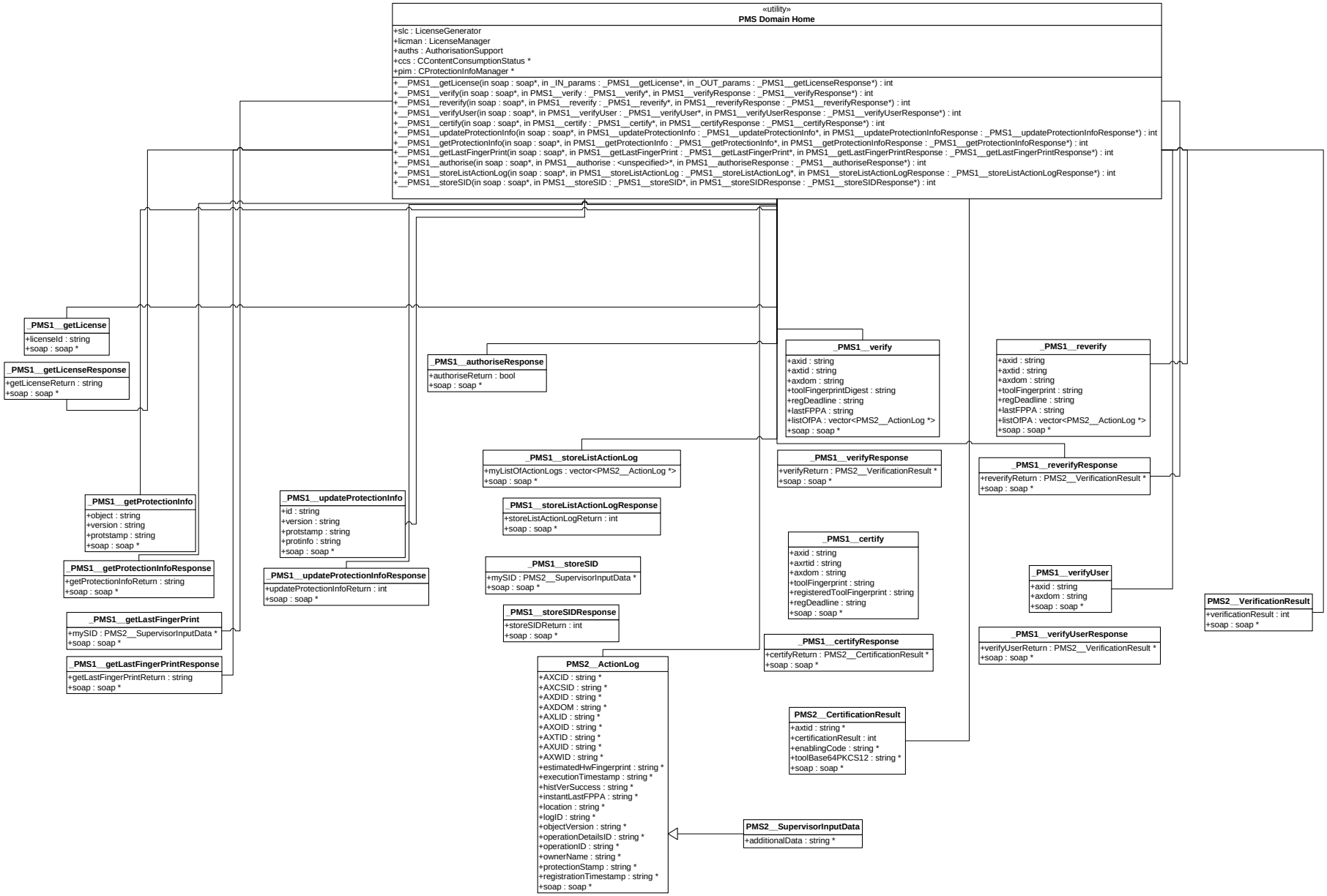
6.1 General Description of the Module

Protection Manager Support Domain Home provides the protection needed for a set of PMS Clients in a home environment. It has connection with AXMEDIS Certifier and Supervisor, in order to check that users only perform the actions they are allowed to. In this section, the general functionality of this module is explained. In next sections, the modules forming part of PMS Domain Home are explained in detail.

PMS Domain Home	
Methods	***Description***
authorise	This function authorises AXMEDIS users to perform actions over AXMEDIS objects. It proves that a user has the appropriate license that grants him to exercise a right over a resource if the conditions are met based on the execution context of the client.
getLicense	This function retrieves the licenses stored in the license database. It retrieves the license with the licenseID set as a parameter, or the licenses associated to an AXOID.
storeListActionLog	This method is used by PMS Client to store through Supervisor a list of Action Logs. When a user has performed some off-line actions, if PMS Client gets connection to the system, it calls verify method, which reaches AXCV through PMS Server in order to resynchronize the actions that are stored in the local cache.
getLastFingerprint	This method is used by PMS Client to request Supervisor the Last Fingerprint of a user or an object or a tool in order to certify or verify any

	user.
verifyUser	This method is called by PMS Client and reaches AXCV through PMS Server. It can be used to verify the status of a user, optionally inside a domain. It verifies if the user is registered in the specified domain (if present) and checks that the user status and registration deadline are valid, so that the user can still use the AXMEDIS tools and the AXMEDIS framework.
certify	This method is called by PMS Client and reaches AXCV through PMS Server. It is used to certify that the original tool has not been modified and to activate it. It creates a new entry in the CerTools table of the AXCS database which associates the user, tool and device and returns to the Protection Processor an activation code, a tool identifier and a PKCS12 structure with the tool certificate and private key issued by AXMEDIS CA (EJBCA).
verify	This method is called by PMS Client and reaches AXCV through PMS Server. It is used to verify that the tool installed on a device has neither been modified nor blocked, that the user is not blocked and that the registered tool is not blocked. It is also responsible for resynchronizing the offline tool operation through AXMEDIS Supervisor (AXS).
reverify	This method is similar to verify method (see previous). It must be called when the verify method fails because of the tool fingerprint hash doesn't match (error code: –9) to perform a new verification with the full fingerprint. Thus, the reverify method has the same input parameters as the verify method except the full tool fingerprint, which has to be sent instead of the hash.
GetProtectionInfo	This method is called by PMS Client and is used to retrieve the protection information related to an object from the Objects Table of the AXCS Objects ID Database.
UpdateProtectionInfo	This method is called by PMS Client and is used to insert or update the protection information related to an AXMEDIS object in the Objects Table of the AXCS Objects ID Database.

6.2 Module Design in terms of Classes


«utility»
PMS Domain Home
+slc : LicenseGenerator
+licman : LicenseManager
+auths : AuthorisationSupport
+ccs : CContentConsumptionStatus *
+pim : CProtectionInfoManager *
+__PMS1__getLicense(in soap : soap*, in _IN_params : _PMS1__getLicense*, in _OUT_params : _PMS1__getLicenseResponse*) : int
+__PMS1__verify(in soap : soap*, in PMS1__verify : _PMS1__verify*, in PMS1__verifyResponse : _PMS1__verifyResponse*) : int
+__PMS1__reverify(in soap : soap*, in PMS1__reverify : _PMS1__reverify*, in PMS1__reverifyResponse : _PMS1__reverifyResponse*) : int
+__PMS1__verifyUser(in soap : soap*, in PMS1__verifyUser : _PMS1__verifyUser*, in PMS1__verifyUserResponse : _PMS1__verifyUserResponse*) : int
+__PMS1__certify(in soap : soap*, in PMS1__certify : _PMS1__certify*, in PMS1__certifyResponse : _PMS1__certifyResponse*) : int
+__PMS1__updateProtectionInfo(in soap : soap*, in PMS1__updateProtectionInfo : _PMS1__updateProtectionInfo*, in PMS1__updateProtectionInfoResponse : _PMS1__updateProtectionInfoResponse*) : int
+__PMS1__getProtectionInfo(in soap : soap*, in PMS1__getProtectionInfo : _PMS1__getProtectionInfo*, in PMS1__getProtectionInfoResponse : _PMS1__getProtectionInfoResponse*) : int
+__PMS1__getLastFingerPrint(in soap : soap*, in PMS1__getLastFingerPrint : _PMS1__getLastFingerPrint*, in PMS1__getLastFingerPrintResponse : _PMS1__getLastFingerPrintResponse*) : int
+__PMS1__authorise(in soap : soap*, in PMS1__authorise : <unspecified>*, in PMS1__authoriseResponse : _PMS1__authoriseResponse*) : int
+__PMS1__storeListActionLog(in soap : soap*, in PMS1__storeListActionLog : _PMS1__storeListActionLog*, in PMS1__storeListActionLogResponse : _PMS1__storeListActionLogResponse*) : int
+__PMS1__storeSID(in soap : soap*, in PMS1__storeSID : _PMS1__storeSID*, in PMS1__storeSIDResponse : _PMS1__storeSIDResponse*) : int
_PMS1__getLicense
+licenseId : string
+soap : soap *
_PMS1__getLicenseResponse
+getLicenseReturn : string
+soap : soap *
_PMS1__authoriseResponse
+authoriseReturn : bool
+soap : soap *
_PMS1__verify
+axid : string
+axtid : string
+axdom : string
+toolFingerprintDigest : string
+regDeadline : string
+lastFPPA : string
+listOfPA : vector<PMS2__ActionLog *>
+soap : soap *
_PMS1__reverify
+axid : string
+axtid : string
+axdom : string
+toolFingerprint : string
+regDeadline : string
+lastFPPA : string
+listOfPA : vector<PMS2__ActionLog *>
+soap : soap *
_PMS1__storeListActionLog
+myListOfActionLogs : vector<PMS2__ActionLog *>
+soap : soap *
_PMS1__verifyResponse
+verifyReturn : PMS2__VerificationResult *
+soap : soap *
_PMS1__reverifyResponse
+reverifyReturn : PMS2__VerificationResult *
+soap : soap *
_PMS1__getProtectionInfo
+object : string
+version : string
+protstamp : string
+soap : soap *
_PMS1__updateProtectionInfo
+id : string
+version : string
+protstamp : string
+protinfo : string
+soap : soap *
_PMS1__storeListActionLogResponse
+storeListActionLogReturn : int
+soap : soap *
_PMS1__certify
+axid : string
+axrtid : string
+axdom : string
+toolFingerprint : string
+registeredToolFingerprint : string
+regDeadline : string
+soap : soap *
_PMS1__verifyUser
+axid : string
+axdom : string
+soap : soap *
PMS2__VerificationResult
+verificationResult : int
+soap : soap *
_PMS1__getProtectionInfoResponse
+getProtectionInfoReturn : string
+soap : soap *
_PMS1__updateProtectionInfoResponse
+updateProtectionInfoReturn : int
+soap : soap *
_PMS1__storeSID
+mySID : PMS2__SupervisorInputData *
+soap : soap *
_PMS1__storeSIDResponse
+storeSIDReturn : int
+soap : soap *
_PMS1__certifyResponse
+certifyReturn : PMS2__CertificationResult *
+soap : soap *
_PMS1__verifyUserResponse
+verifyUserReturn : PMS2__VerificationResult *
+soap : soap *
_PMS1__getLastFingerPrint
+mySID : PMS2__SupervisorInputData *
+soap : soap *
_PMS1__getLastFingerPrintResponse
+getLastFingerPrintReturn : string
+soap : soap *
PMS2__ActionLog
+AXCID : string *
+AXCSID : string *
+AXDID : string *
+AXDOM : string *
+AXLID : string *
+AXOID : string *
+AXTID : string *
+AXUID : string *
+AXWID : string *
+estimatedHwFingerprint : string *
+executionTimestamp : string *
+histVerSuccess : string *
+instantLastFPPA : string *
+location : string *
+logID : string *
+objectVersion : string *
+operationDetailsID : string *
+operationID : string *
+ownerName : string *
+protectionStamp : string *
+registrationTimestamp : string *
+soap : soap *
PMS2__CertificationResult
+axtid : string *
+certificationResult : int
+enablingCode : string *
+toolBase64PKCS12 : string *
+soap : soap *
PMS2__SupervisorInputData
+additionalData : string *

6.3 Formal description of PMS Domain Home

authorise	
Method	authorise
Description	This function authorises AXMEDIS users to perform actions over AXMEDIS objects. It proves that a user has the appropriate license that grants him to exercise a right over a resource if the conditions are met based on the execution context of the client.
Input parameters	String userID: User id to be authorised String action: action to be authorised String resource: resource to be authorised contextData context: context of the client to be authorised ActionLog constructingAL: Actionlog of the authorisation with the "client side" parameters fulfilled
Output parameters	Integer:

getLicense	
Method	getLicense
Description	This function retrieves the licenses stored in the license database. It retrieves the license with the licenseID set as a parameter, or the licenses associated to an AXOID.
Input parameters	String licenseId: License Id
Output parameters	String, the license in XML

verifyUser	
Method	verifyUser
Description	This method is called by PMS Client and reaches AXCV through PMS Server. It can be used to verify the status of a user, optionally inside a domain. It verifies if the user is registered in the specified domain (if present) and checks that the user status and registration deadline are valid, so that the user can still use the AXMEDIS tools and the AXMEDIS framework.
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axdom**: AXMEDIS domain of certified user (if any)
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: Verification OK -1: invalid AXID -2: user is not registered -3: user is blocked -4: user domain mismatch -5: user registration deadline expired When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur, but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the deadline has expired or not).

certify	
Method	certify
Description	This method is called by PMS Client and reaches AXCV through PMS Server. It is used to certify that the original tool has not been modified and to activate it. It creates a new entry in

	the CerTools table of the AXCS database which associates the user, tool and device and returns to the Protection Processor an activation code, a tool identifier and a PKCS12 structure with the tool certificate and private key issued by AXMEDIS CA (EJBCA).
Input parameters	xsd:string axid: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axrtid**: identifier of the registered AXMEDIS tool xsd:string **axdom**: domain where the user is registered. xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool xsd:string **regDeadline**: registration deadline of the installed tool.
Output parameters	CertificationResult complex type formed by sequence of: xsd:string **axtid**, the identifier of the installed tool associated to a user and device. xsd:int **certificationResult**, which indicates the result of the certification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: tool not registered (RegTools table) -7: registered tool is blocked -8: received tool deadline exceeds registered tool deadline (user and tool have been blocked) -9: received tool deadline has expired -10: registered tool fingerprint mismatch. Tool has been manipulated (user and tool have been blocked) -11: user-tool-device had already been certified. New tool certificate should be created -20: error updating user status in database -21: error inserting new entry in CerTools table -22: error in AXSupervisor when communicating with database -30: internal AXCV error xsd:string **enablingCode**, the tool activation code sent to the Protection Processor. byte[] **toolBase64PKCS12**, PKCS12 structure bytes encoded in Base 64. It includes the tool certificate signed by the AXCS CA Root Certificate and tool private key together and protected with a password. If the unrestricted policy files for Sun JCE were available at the server (default configuration), the password will be the full AXMEDIS AXID. Otherwise, the password will be the first 8 characters of the AXMEDIS AXID. It proves that an AXMEDIS tool has been certified and can be used in the AXMEDIS framework When an error code *x* is returned, it means that all the possible errors *y*, $x<y<0$ did not occur, but all possible errors $y<x$ have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the registered tool is blocked or not, or if the tool fingerprint did match or not).

verify	
Method	verify
Description	This method is called by PMS Client and reaches AXCV through PMS Server. It is used to verify that the tool installed on a device has neither been modified nor blocked, that the user is not blocked and that the registered tool is not blocked. It is also responsible for resynchronizing the offline tool operation through AXMEDIS Supervisor (AXS).
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a

	device). xsd:string **axdom**: domain where the user is registered. byte[] **toolFingerprintDigest**: SHA1 hash of the relevant data of hash of the full fingerprint (software and hardware parts) of the installed tool. byte[] **LastFPPA**: fingerprint of the history of the operations performed during the offline operation. tns2:ActionLog **listOfPA**: Array of ActionLogs, which is a complex type defined in AXMEDIS Supervisor, including the actions performed during the offline operation.
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: AXTID does not exist -7: installed (and certified) tool is blocked -8: tool deadline has expired -9: toolFingerprintDigest (toolFingerprint hash) mismatch -10: toolFingerprint mismatch (user and tool have been blocked) -11: registered tool is blocked -12: user has been blocked and installed tool has been blocked again -13: tool has been blocked -20: error updating user status in database -21: error updating tool status in database -22: error updating LastFPPA in database -23: error retrieving regtool data from database -24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog -25: error in AXSupervisor when communicating with AXCS accounting database in storeSID -30: internal AXCV error xsd:int **storeListActionLogResult**, which indicates the result of the storage of the action logs, according to the following numeration: 0: ActionLog(s) has been stored: it includes the case of empty list -1: ActionLog(s) has been stored: tool should have been already blocked -2: ActionLog(s) has been stored: tool operation history hash (LastFPPA) is not consistent -3: ActionLog(s) has not been stored: error in AXSupervisor when communicating with AXCS database -4: ActionLog(s) has not been stored: input actionLog(s) do not refer to the same AXTID -5: ActionLog(s) has not been stored: input actionLog(s) have some non-nillable null fields -6: ActionLog(s) has not been stored: user or tool data unsuccessfully verified by AXCV When an error code *x* is returned, it means that all the possible errors *y*, $x<y<0$ did not occur, but all possible errors $y<x$ have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the received tool deadline has expired or not).

reverify	
Method	reverify
Description	This method is similar to verify method (see previous). It must be called when the verify method fails because of the tool fingerprint hash doesn't match (error code: –9) to perform a new verification with the full fingerprint. Thus, the reverify method has the same input parameters as the verify method except the full tool fingerprint, which has to be sent instead of the hash.
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string axtid: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axdom**: domain where the user is registered. xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool. byte[] **LastFPPA**: fingerprint of the history of the operations performed during the offline operation. tns2:ActionLog **listOfPA**: Array of ActionLogs, which is a complex type defined in AXMEDIS Supervisor, including the actions performed during the offline operation.
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: AXTID does not exist -7: installed (and certified) tool is blocked -8: tool deadline has expired -9: toolFingerprintDigest (toolFingerprint hash) mismatch -10: toolFingerprint mismatch (user and tool have been blocked) -11: registered tool is blocked -12: user has been blocked and installed tool has been blocked again -13: tool has been blocked -20: error updating user status in database -21: error updating tool status in database -22: error updating LastFPPA in database -23: error retrieving regtool data from database -24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog -25: error in AXSupervisor when communicating with AXCS accounting database in storeSID -30: internal AXCV error xsd:int **storeListActionLogResult**, which indicates the result of the storage of the action logs, according to the following numeration: 0: ActionLog(s) has been stored: it includes the case of empty list -1: ActionLog(s) has been stored: tool should have been already blocked -2: ActionLog(s) has been stored: tool operation history hash (LastFPPA) is not consistent -3: ActionLog(s) has not been stored: error in AXSupervisor when communicating with AXCS database -4: ActionLog(s) has not been stored: input actionLog(s) do not refer to the same

	AXTID -5: ActionLog(s) has not been stored: input actionLog(s) have some non-nillable null fields -6: ActionLog(s) has not been stored: user or tool data unsuccessfully verified by AXCV When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur, but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the received tool deadline has expired or not).

getProtectionInfo	
Method	getProtectionInfo
Description	This method is called by PMS Client and is used to retrieve the protection information related to an object from the Objects Table of the AXCS Objects ID Database.
Input parameters	The following fields of the Objects table in the AXCS Objects ID database: type="xsd:string" **AXOID**, AXMEDIS object identifier type="xsd:string" **ObjectVersion**, object version type="xsd:string" **ProtectionStamp**, protection stamp
Output parameters	type="xsd:string" **ProtectionInfo**, protection information associated to the object or a "wrong_object" result if there is no ProtectionInfo for the requested object

UpdateProtectionInfo	
Method	UpdateProtectionInfo
Description	This method is called by PMS Client and is used to insert or update the protection information related to an AXMEDIS object in the Objects Table of the AXCS Objects ID Database.
Input parameters	The following fields of the Objects table in the AXCS Objects ID database: type="xsd:string" **AXOID**, AXMEDIS object identifier type="xsd:string" **ObjectVersion**, object version type="xsd:string" **ProtectionStamp**, protection stamp type="xsd:string" **ProtectionInfo**, protection information to be updated type="xsd:int" **Update**, denotes if the protection info must be inserted (0) or updated (1)
Output parameters	type="xsd:int" **updateProtectionInfoReturn**, which indicates the result of this request, according to the following numeration: 0: OK -1: there is not any entry in AXCS Objects database that matches the input information -2: error in AXSupervisor when updating ProtectionInfo in AXCS Objects database

7 License Manager

Module/Tool Profile		
License Manager		
Responsible Name	Rubén Barrio	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First version available	
Executable or Library/module (Support)	Library	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/licensemanager	
Reference to the AXFW location of the demonstrator executable tool for internal download	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

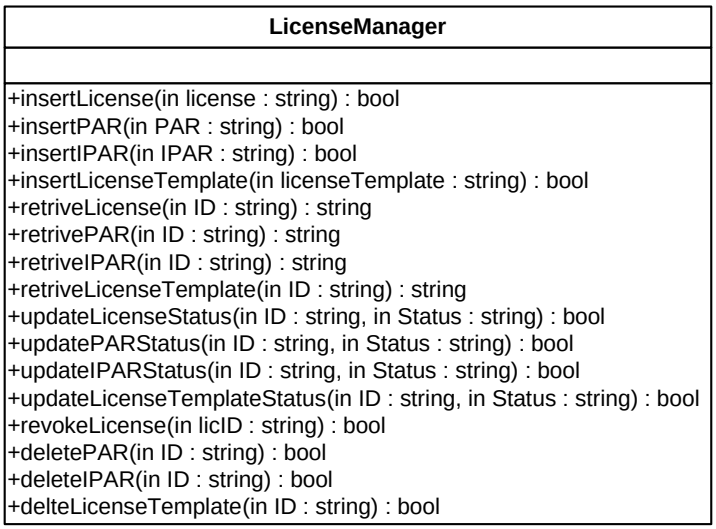
7.1 General Description of the Module

This module performs storage and retrieval of licenses. It is the responsible of storing the licenses into the database.

The functions it performs are:

- Storage and retrieval of licenses.
- Retrieval of conditions for authorisation support.
- Calculating the hash of the data stored in the database for a license.

7.2 Module Design in terms of Classes



7.3 Formal description of license manager algorithm

insertLicense	
Method	insertLicense
Description	This function stores a license in the license database.
Input parameters	type="xsd:string" **license**, This is the XML MPEG21 REL License to be stored
Output parameters	type="xsd:bool" True if the license can be stored correctly

InsertPAR	
Method	InsertPAR
Description	This function stores a PAR in the PAR database.
Input parameters	type="xsd:string" **PAR**, This is the PAR to be stored
Output parameters	type="xsd:bool" True if the PAR can be stored correctly

insertIPAR	
Method	insertIPAR
Description	This function stores an internal PAR in the PAR database.
Input parameters	type="xsd:string" **IPAR**, This is the IPAR to be stored
Output parameters	type="xsd:bool" True if the internal PAR can be stored correctly

insertLicenseTemplate	
Method	insertLicenseTemplate
Description	This function stores a license template in the license template database.
Input parameters	type="xsd:string" **licenseTemplate**, This is the XML MPEG21 REL License template to be stored
Output parameters	type="xsd:bool" True if the license template can be stored correctly

retrieveLicense	
Method	retrieveLicense
Description	This function retrives a license from the license database.
Input parameters	type="xsd:string" **ID**, This is the ID of the XML MPEG21 REL License.
Output parameters	type="xsd:string" The license.

retrievePAR	
Method	retrievePAR
Description	This function retrives a PAR from the PAR database.
Input parameters	type="xsd:string" **ID**, This is the ID of the PAR
Output parameters	type="xsd:string" The PAR

retrieveIPAR	
Method	retrieveIPAR
Description	This function retrives an internal PAR from the PAR database.
Input parameters	type="xsd:string" **ID,** This is the ID of the internal PAR
Output parameters	type="xsd:string" The internal PAR

retrieveLicenseTemplate	
Method	retrieveLicenseTemplate
Description	This function retrives a LicenseTemplate from the LicenseTemplate database.
Input parameters	type="xsd:string" **ID,** This is the ID of the LicenseTemplate
Output parameters	type="xsd:string" The LicenseTemplate

LicenseManager	
Method	updateLicenseStatus
Description	This function changes the status of a License
Input parameters	type="xsd:string" **ID**, This is the ID of the XML MPEG21 REL License. type="xsd:string" **Status**, This is the new status..
Output parameters	type="xsd:bool" True if the status has changed correctly

updatePARStatus	
Method	updatePARStatus
Description	This function changes the status of a PAR
Input parameters	type="xsd:string" **ID**, This is the ID of the PAR. type="xsd:string" **Status**, This is the new status..
Output parameters	type="xsd:bool" True if the status has changed correctly

updateIPARStatus	
Method	updateIPARStatus
Description	This function changes the status of an internal PAR
Input parameters	type="xsd:string" **ID**, This is the ID of the internal PAR. type="xsd:string" **Status**, This is the new status.
Output parameters	type="xsd:bool" True if the status has changed correctly

updateLicenseTemplateStatus	
Method	updateLicenseTemplateStatus
Description	This function changes the status of a LicenseTemplate
Input parameters	type="xsd:string" **ID**, This is the ID of License Template. type="xsd:string" **Status**, This is the new status..
Output parameters	type="xsd:bool" True if the status has changed correctly

RevokeLicense	
Method	RevokeLicense
Description	This function revokes a License
Input parameters	type="xsd:string" **ID**, This is the ID of the XML MPEG21 REL License.
Output parameters	type="xsd:bool" True if the license has been revoked

deletePAR	
Method	deletePAR
Description	This function deletes a PAR from the database .
Input parameters	type="xsd:string" **ID**, This is the ID of the PAR to be deleted.
Output parameters	type="xsd:bool" True if the PAR has been deleted

deleteIPAR	
Method	deleteIPAR
Description	This function deletes an internal PAR from the database .
Input parameters	type="xsd:string" **ID**, This is the ID of the internal PAR to be deleted.
Output parameters	type="xsd:bool" True if the internal PAR has been deleted

deleteLicenseTemplate	
Method	deleteLicenseTemplate
Description	This function deletes a License Template from the database .
Input parameters	type="xsd:string" **ID**, This is the ID of the license template to be deleted.
Output parameters	type="xsd:bool" True if the license template has been deleted

8 License Verificator

Module/Tool Profile	
License Verificator	
Responsible Name	Rubén Barrio
Responsible Partner	UPC
Status (proposed/approved)	Approved
Implemented/not implemented	Implemented
Status of the implementation	First version available
Executable or Library/module (Support)	Library
Single Thread or Multithread	Multithread
Language of Development	C++
Platforms supported	Windows
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/licenseverificator
Reference to the AXFW	N/A

location of the demonstrator executable tool for internal download		
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	N/A	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	None	
Major pending requirements	- Verify against PAR	
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.)
verifyLicense		
verifyCreatedLicense		
verifyTemporalLicense		
verifyPAR		
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		
RDDServer		
License Database		
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
licenseModelD.lib	LicenseModel	
licenseManagerD.lib	LicenseManager	

8.1 General Description of the Module

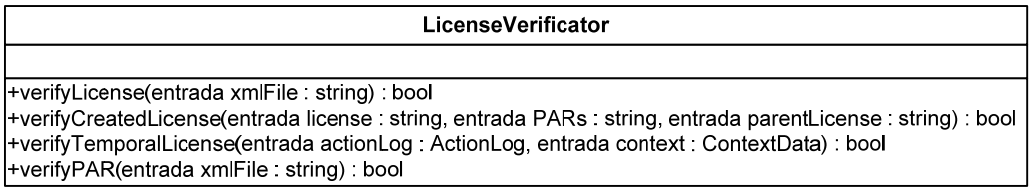
This module performs validation operations against licenses and PARs.

The functions it performs are:

- It determines if an MPEG-21 REL license is valid syntactically or against the schemas used by the license.
- Verifies if the license can be generated according to the PARs and the parent licenses
- Verifies that the license generated by the user fulfils the initial desirables requirements of the user. For example, the user can verify that with this license he could exercise the desired action over the AXObject.
- Verifies a PAR syntactically against the schemas defined within the PAR.

8.2 Module Design in terms of Classes

This module is inside PMS.



8.3 User interface description

This module does not have user interface.

8.4 Technical and Installation information

To use this library, it is only needed to link the corresponding library and the XERCES lib.

References to other major components needed	RDD Server
Problems not solved	
Configuration and execution context	The configuration is stablished in parameters in licman.ini.

8.5 Draft User Manual

It is needed just to call the public methods of this library.

8.6 Examples of usage

Example 1: Syntactic verification of a license
In order to verify if a license is valid against the schemas defined within the license the method verifyLicense should be called and as parameter the path where the license is located. This method returns true if the license is valid and false otherwise.

Example 2: License verification according to the PARs and the parent licenses
In order to verify if a license has been correctly generated according to the PARs and to the parent licenses, the verifyCreated method of this class should be invoked. The parameters of this method are the license created, the PARs and the parent license. This method returns true if the license has been appropriately generated and false otherwise.

Example 3: License generation verification
In order to verify if a license fulfil the requirements desired by the user, the verifyTemporalMethod of this class should be invoked. This method has as inputs the license generated, the context, the AXUID, the right and the AXOID. This method returns true if the license generated accomplishes the requirements of the user and false otherwise.

Example 4: Syntactic verification of a PAR
In order to verify if a PAR is valid against the schemas defined within the PAR the method verifyPAR should be called and as parameter the path where the license is located. This method returns true if the PAR is valid and false otherwise.

8.7 Integration and compilation issues

As this module does not use any system dependent library, it should be compatible with the different operating systems where it is compiled.

8.8 Configuration Parameters

These values are defined in file licman.ini.

Config parameter	Possible values
user	axmedis
password	axmedis
database	axmedis
RDDDsn	AXRDDServer

8.9 Formal description of License Verificator

verifyCreatedLicense	
Method	verifyCreatedLicense
Description	Verifies if the license can be generated according to the PARs and the parent licenses
Input parameters	- license: the generated license - PARs: possible available rights associated to the object of the generated license - parentLicense: the license of the previous actor in the value chain that governs the object of the generated license
Output parameters	A Boolean value that indicates if the license has been generated according to the PARs and parent licenses or not.

verifyTemporalLicense	
Method	verifyTemporalLicense
Description	Verifies that the license generated by the user fulfils the initial desirables requirements of the user
Input	• actionLog: this structure is used only for getting filled fields, like AXOID, AXUID,

parameters	operationID, … that we help us to check the license. • Context: the context data for the generated license.
Output parameters	A Boolean value that indicates if the license fulfil the requirements of the user or not.

verifyLicense	
Method	verifyLicense
Description	Validates an XML license against the schemas specified in it
Input parameters	• xmlFile: the generated license in xml.
Output parameters	A Boolean value that indicates if the license is ok or not.

verifyPAR	
Method	verifyPAR
Description	Verifies that the license generated by the license creator and checks against PARs.
Input parameters	• xmlFile: the generated license in xml.
Output parameters	A Boolean value that indicates if the license is ok or not.

9 License Generator

Module/Tool Profile		
License Generator		
Responsible Name	Rubén Barrio	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First version available	
Executable or Library/module (Support)	Library	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/licensegenerator	
Reference to the AXFW location of the demonstrator executable tool for internal download	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

9.1 General Description of the Module

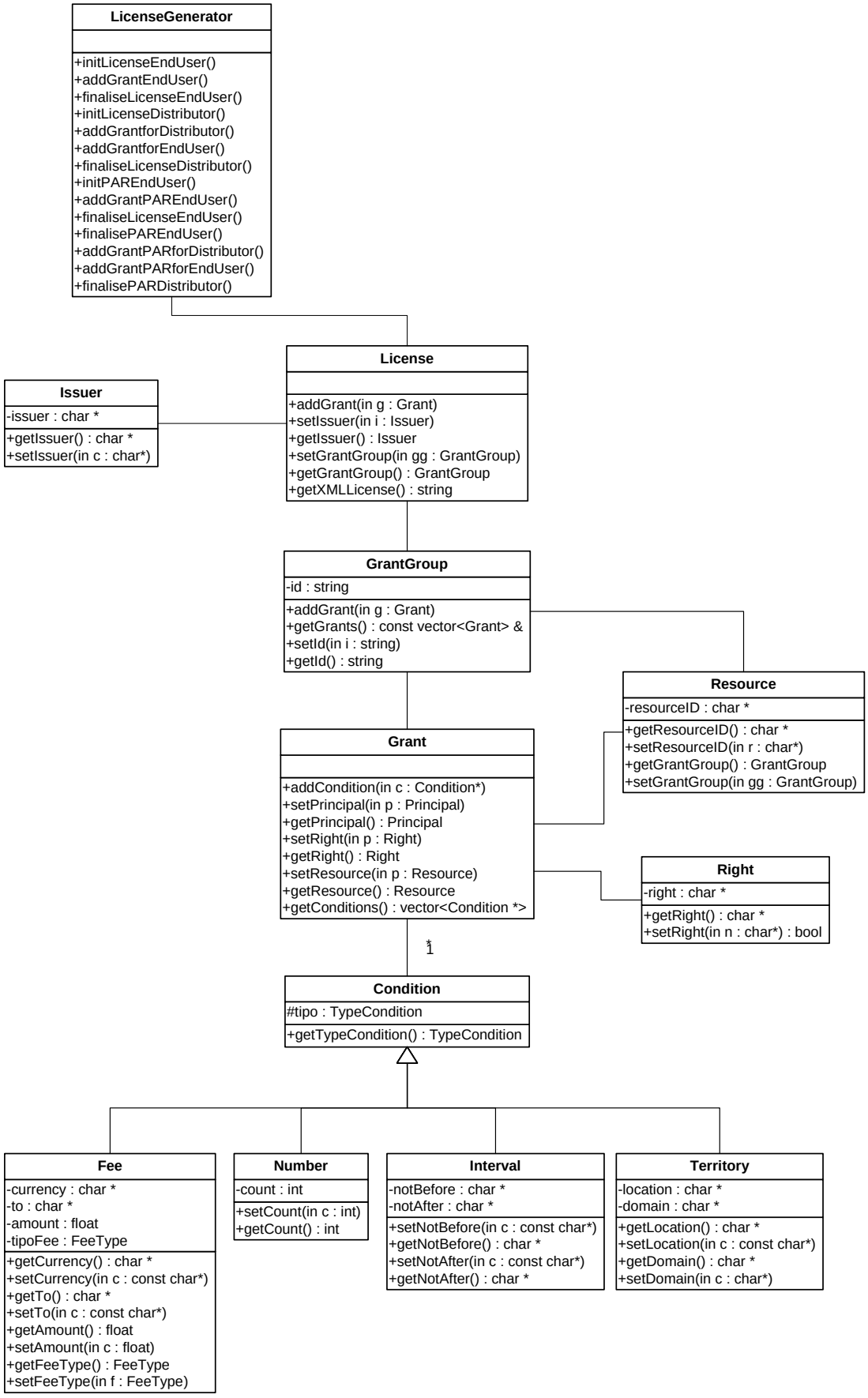
This module is the responsible of the creation of licenses. In this module is also described the license object model.

The license object model follows the MPEG-21 REL format to store licensing information. It can store rights, users and conditions related to content.

License Generator module, also offers functions that allow, in a simple way, the creation of complex licenses (as an object model). These functions are described below, and all works in the same way:

- Initialization of a license.
- Add rights
- Finalization of a license.

9.2 Module design in terms of Classes



9.3 Formal description of License Generator algorithms

InitLicenseEndUser	
Method	InitLicenseEndUser
Description	InitLicenseEndUser initialises the creation of a license. This is the first function to be called in the process of an End User License creation. The service initLicenseEndUser returns the temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	IssuerAXUID String with the Issuer AXUID (creator of the license).
Output parameters	

AddGrantEndUser	
Method	AddGrantEndUser
Description	AddGrantEndUser is the function that adds (one each time) the rights granted in a license. This service has to be called as many times as rights granted by the license. The different parameters allow introducing: the right, the resource over which the right will be exercised, the user who will obtain the right, and finally, the different conditions to be accomplished.
Input parameters	**licenseTmpId** Temporal license identifier, returned by initLicenseEndUser. **AXUIDPrincipal** This is the AXUID of the user (user of the license). **diResource** Establishes that the resource will be referenced by an URI. f.e. http://www.musicserver.org/track1.mp3 If this parameter is TRUE, diReference has to be FALSE diType Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **right** The right that will be granted in the license. Can take the following values: adapt, delete, diminish, embed, enhance, enlarge, execute, install, modify, move, play, print, reduce, uninstall that correspond to rights described in “MPEG-21 multimedia extension”. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised.

	feeType This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. fee If feeType is not 0, this parameter corresponds to the fee. currency If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different condition adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created and added to the license. If the right has not been created, the returned value is 4XX:Error causes. If the right has been correctly created, it returns 200:OK

finaliseLicenseEndUser	
Method	finaliseLicenseEndUser
Description	finaliseLicenseEndUser finalises the license. This is the function to be invoked in a license reation process. The function builds the license and, if it is correct, then stores it in the database.
Input parameters	licenseTmpId String with the Temporal license ID returned by initLicenseEndUser.
Output parameters	A String with the license identifier. This is unique identifier of the license and can be used to retrieve a copy of the license

InitLicenseDistributor	
Method	InitLicenseDistributor
Description	InitLicenseDistributor initialises the creation of a license. This is the first function to be called in the process of a Distributor License creation. This service receives information about the creator of the license. The function initLicenseEndUser returns the temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	IssuerAXUID String with the Issuer AXUID (normally creator of the content or rights owner).
Output parameters	The temporal identifier of the license. This identifier is usable while the license is being created. When the license is finished and stored, this identifier is not used any more

addGrantforDistributor	
Method	addGrantforDistributor
Description	addGrantforDistributor is the function that adds (one each time) the different rights for distributors and the distribution conditions for each one. The parameters established in this function affect only to the issue right (the one defining distribution). This function has to be called as many times as distributors the license has. The different parameters allow introducing: the distribution conditions, the content that will be distributed and the identification of the distributor.
Input parameters	**licenseTmpId** Temporal license identifier, returned by initLicenseDistributor. **AXUIDPrincipal** This is the AXUID of the principal (the distributor user). **diType** Establishes the type of the resource. It can be:

	If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised. region If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done.
Output parameters	a String with the temporal distributor grant ID. This identifier is usable while the license is being created, and it will be used to assign the different distributable rights to the distributor with AddGrantforEndUser.

addGrantforEndUser	
Method	addGrantforEndUser
Description	addGrantforEndUser is the function that adds (one each time) the rights that a distributor can distribute. In other words, this function adds the rights that can be included in an EndUser License created by a specific distributor. This function has to be called as many times as different rights will be available in the future EndUser licenses. The different parameters allow introducing: right and the different conditions to be accomplished. The resource is established before in the addGrantforDistributor service.
Input parameters	**licenseTmpId** Temporal license identifier, returned by initLicenseDistributor. **distGrantId** Temporal grant identifier, returned by AddGrantforDistributor. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBeforeIf** validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times.

	limit If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. country If validityRegion is TRUE, this parameter corresponds to the **country** where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created into the license. If the right has not been created, the returned value is 4XX:Error causes. If the right has been created normally, it returns 200:OK.

finaliseLicenseDistributor	
Method	finaliseLicenseDistributor
Description	finaliseLicenseDistributor finalises the license. This is the last function to be invoked in a license creation process. The service builds the licenses and, if it is correct, then stores it in the database.
Input parameters	licenseTmpId Temporal license identifier, returned by initLicenseDistributor.
Output parameters	String with the license identifier. This is a unique identifier of the license and can be used to retrieve a copy of the license

InitPAREndUser	
Method	InitPAREndUser
Description	InitPAREndUser initialises the creation of a PAR. This is the first function to be called in the process of an End User PAR creation. The service initPAREndUser returns the temporal identifier of the PAR. This identifier is usable while the PAR is being created. When the PAR is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	
Output parameters	The temporal identifier of the PAR. This identifier is usable while the PAR is being created. When the PAR is finished and stored, this identifier is not used any more

AddGrantPAREndUser	
Method	AddGrantPAREndUser
Description	AddGrantPAREndUser is the function that adds (one each time) the rights granted in a PAR. This service has to be called as many times as rights granted by the PAR. The different parameters allow introducing: the right, the resource over which the right will be exercised, the user who will obtain the right, and finally, the different conditions to be accomplished.
Input	**PARTmpId** Temporal PAR identifier, returned by initPAREndUser.

parameters	**diResource** Establishes that the resource will be referenced by an URI. f.e. http://www.musicserver.org/track1.mp3 If this parameter is TRUE, diReference has to be FALSE diType Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **right** The right that will be granted in the PAR. Can take the following values: adapt, delete, diminish, embed, enhance, enlarge, execute, install, modify, move, play, print, reduce, uninstall that correspond to rights described in "MPEG-21 multimedia extension". **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. fee If feeType is not 0, this parameter corresponds to the fee. currency If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different condition adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created and added to the PAR. If the right has not been created, the returned value is 4XX:Error causes. If the right has been correctly created, it returns 200:OK

finalisePAREndUser	
Method	finalisePAREndUser
Description	finalisePAREndUser finalises the PAR. This is the function to be invoked in a PAR reation process. The function builds the PAR and, if it is correct, then stores it in the database.
Input parameters	PARTmpId String with the Temporal PAR ID returned by initPAREndUser.
Output parameters	A String with the PAR identifier. This is unique identifier of the PAR and can be used to retrieve a copy of the PAR

InitPARDistributor	
Method	InitPARDistributor
Description	InitPARDistributor initialises the creation of a PAR. This is the first function to be called in the process of a Distributor PAR creation. This service receives information about the creator of the PAR. The function initPAREndUser returns the temporal identifier of the PAR. This identifier is usable while the PAR is being created. When the PAR is finished and stored this identifier is not used any more and it is deleted from the database.
Input parameters	
Output parameters	The temporal identifier of the PAR. This identifier is usable while the PAR is being created. When the PAR is finished and stored, this identifier is not used any more

addGrantPARforDistributor	
Method	addGrantPARforDistributor
Description	addGrantPARforDistributor is the function that adds (one each time) the different rights for distributors and the distribution conditions for each one. The parameters established in this function affect only to the issue right (the one defining distribution). This function has to be called as many times as distributors the PAR has. The different parameters allow introducing: the distribution conditions, the content that will be distributed and the identification of the distributor.
Input parameters	**PARTmpId** Temporal PAR identifier, returned by initPARDistributor. **diType** Establishes the type of the resource. It can be: If diType is 0 means that AXOID is an AXOID (digitalResource). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476) If diType is 1 means that AXOID is an AXOID reference (diReference). (urn:mpegRA:mpeg21:dii:isrc:US-ZO3-99-32476#CollineAzzurre) **AXOID** The resource identifier. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBefore** If validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. **country** If validityRegion is TRUE, this parameter corresponds to the country where the right can be exercised. region If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is

	exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done.
Output parameters	a String with the temporal distributor grant ID. This identifier is usable while the PAR is being created, and it will be used to assign the different distributable rights to the distributor with AddGrantforEndUser.

addGrantPARforEndUser	
Method	addGrantPARforEndUser
Description	addGrantPARforEndUser is the function that adds (one each time) the rights that a distributor can distribute. In other words, this function adds the rights that can be included in an EndUser PAR created by a specific distributor. This function has to be called as many times as different rights will be available in the future EndUser PARs. The different parameters allow introducing: right and the different conditions to be accomplished. The resource is established before in the addGrantforDistributor service.
Input parameters	**PARTmpId** Temporal PAR identifier, returned by initPARDistributor. **distGrantId** Temporal grant identifier, returned by AddGrantforDistributor. **validityInterval** If this parameter is TRUE the right can be exercised within a time period. If it is FALSE it could be exercised always. **notBeforeIf** validityInterval is TRUE, this parameter corresponds to the date from which the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **notAfter** If validityInterval is TRUE, this parameter corresponds to the date until the right will be effective. It has to have the next format: YYYY-MM-DDTHH:MM:SS **countLimit** This parameters shows if the right will be effective for a specific number of uses (TRUE) or could be exercised any number of times. **limit** If countLimit is TRUE, this parameter corresponds to the number of times that the right can be exercised. **validityRegion** It shows if the right can be exercised only in a specific region or everywhere. country If validityRegion is TRUE, this parameter corresponds to the **country** where the right can be exercised. **region** If validityRegion is TRUE, this parameter corresponds to the region where the right can be exercised. **feeType** This parameter shows if a fee has to be paid to exercise the right If feeType is 0 means that no payment is needed. If feeType is 1 (FeeFlat) means that is needed a unique payment to exercise the right as many times as the user wants. If feeType is 2 (FeePerUse) means that is needed a payment each time that the right is exercised. **fee** If feeType is not 0, this parameter corresponds to the fee. **currency** If feeType is not 0, this parameter corresponds to the currency of the fee. **bankAccount** If feeType is not 0, this parameter corresponds to the bank account where the payment will be done. **adaptationRules** If right is adapt, enhance, enlarge, modify o reduce, this parameter corresponds to the different adaptation rules of the content.
Output parameters	String that shows if the right and its parameters have been created into the PAR. If the right has not been created, the returned value is 4XX:Error causes. If the right has been created normally, it returns 200:OK.

finalisePARDistributor	
Method	finalisePARDistributor

Description	finalisePARDistributor finalises the PAR. This is the last function to be invoked in a PAR creation process. The service builds the PARs and, if it is correct, then stores it in the database.
Input parameters	PARTmpId Temporal PAR identifier, returned by initPARDistributor.
Output parameters	String with the PAR identifier. This is a unique identifier of the PAR and can be used to retrieve a copy of the PAR

10 Authorisation support

Module/Tool Profile		
Authorisation Support		
Responsible Name	Rubén Barrio	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First version available	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/authorisationsupport	
Reference to the AXFW location of the demonstrator executable tool for internal download	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	N/A	
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
MPEG-21 REL license		
Protocol Used	Shared with	Protocol name or reference to a

		section
Used Database name		
License Database		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
N/A		
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
wxmsw24d.lib	Wx Windows for Windows 2.4.2.0	GPL

10.1 General Description of the Module

Authorisation support module is implemented as a C++ class, which checks if the user can perform the action taking into account the licenses he owns.

There is one overloaded method called authorise to do so. There is a first authorisation for local client, another one in server side.

Authorise looks at the current user context (retrieved from local database – securecache - or received as a parameter on server) and compares if the data is correct, that means:

- License should be conceded before than the current date.
- Territory is more restricted in license than in local context. If user has ES-CT in license, but in context information we only get ES, the license is rejected.
- The number expressed in exercise limit license condition for an action should be less than the value stored in the user context.
- If license is derived from a trusted Parent License, then the Parent License context data is equal to the child.

This module needs the configuration file licman.ini, containing the following fields:

- host=193.145.44.41
- user=axmedis
- password=axmedis
- RDDDsn=AXRDDServer
- database=axmedis

The implemented module is supported on different platforms, as Windows OS specific libraries are not used (we use wxWindows instead), so it is only needed to recompile the source code. There is no support for Multilanguage, as this module does not have GUI.

10.2 Module Design in terms of Classes

The figure shows the definition of the AuthorisationSupport Class. This class is located inside PMS.

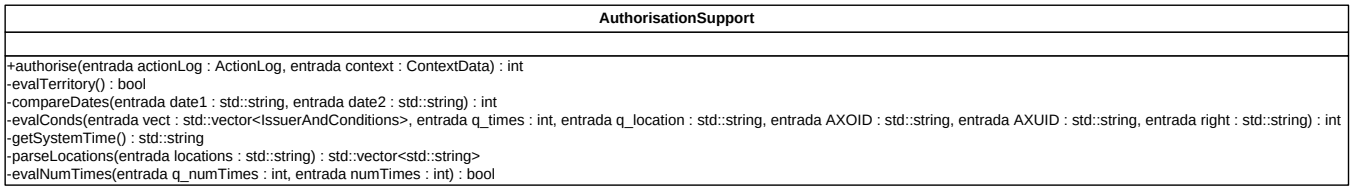


Figure. Authorisation Support Class

10.3 Technical and Installation information

To use this library, it is only needed to link the authorisationSupport.lib and the wxWindows required library with the corresponding module.

References to other major components needed	Secure Cache
Problems not solved	• SecureCache context table has to be revised.
Configuration and execution context	Needs licman.ini file described database parameters

10.4 Draft User Manual

In order to use this library, it is needed to look for the correct information and then call authorise. If authorisation is local, the user should have a local ODBC link referencing secure cache (its name should be securecache), but now the parameter is got from the pmsclient. The configuration can be changed in the licman.ini file.

10.5 Examples of usage

In Client side:

```
alog.AXUID  = "AXUID:129292-228974ddd";
alog.AXOID  = "AXOID:283519878-373949";
alog.operationID = "mx:play";
// Context Filling
cdata.territoryOfEmission = "country{iso:ES}region{iso:ES-CT}";
cdata.timesUsed = 5;
bitwise=aus.authorise(alog, cdata);
```

10.6 Integration and compilation issues

As this module does not use any system dependent library, it should be compatible with the different operating systems supported by wxWidgets.

10.7 Configuration Parameters

These values are stored in the file licman.ini.

Config parameter	Possible values
user	axmedis
password	axmedis
database	axmedis
RDDDsn	AXRDDServer

10.8 Errors reported and that may occur

The error reporting is bitwise (in a integer) and also a descriptive string is returned. Some of the error codes reported are warnings as they give advice of problems arose during license validation.

Error code	Description and rationales
128d = 10000000b	`Territory not satisfied`
64d = 01000000b	`The resource was so many times played`
32d = 00100000b	`License is out of date`
16d = 00010000b	`Cannot Connect Database`
8d = 00001000b	`Current Date is before than emission date`
4d = 00000100b	`Error opening file licman.ini`
2d = 00000010b	`Conditions Rejected. Invalid License.`
1d = 00000001b	`Unknown error (probably NULL pointer).`

10.9 Formal description of authorisation algorithm

License Verification algorithm

Right is in license, or is on rddServer as a child of the right specified in the license
Grant
Resource is allowed in any license for current user (AXUID)
Conditions (Parent License) are Satisfied

timeOfIssue is within the interval of the verification process and not larger than current date.

authorise	
Method	authorise
Description	Check if license is OK and right can be done. Returns zero if all is ok, if doesn't returns bitwise specified in error codes. The context is obtained from pmsclient, and passed as a parameter now.
Input parameters	ActionLog actLog, ContextData context
Output parameters	integer, zero if license accepted, a bitwise containing errors if rejected. The bitwise will have packed all errors to check clearly why the license was rejected.

11 RDD Server

Module/Tool Profile		
RDD Server		
Responsible Name	Rubén Barrio	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First version available	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithreaded	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/rddserver	
Reference to the AXFW location of the demonstrator executable tool for internal download	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	N/A	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	None	
Major pending requirements	None	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a

		section
Used Database name		
RDD Database		
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
wxmsw24d.lib		

11.1 General Description of the Module

RDDServer is a class for obtaining information about rights hierarchy. It is used when checking licenses' rights like when we have a license that allows adapt, but user wants to perform a play.

As adapt is a parent right of play, we will allow the user to perform the action play when authorising with AuthorisationSupport module, as the hierarchy of the play right is checked using rddserver.

This module needs the configuration file licman.ini, containing the following fields:

- user=axmedis
- password=axmedis
- RDDDsn=AXRDDServer

11.2 Module Design in terms of Classes

The figure shows the definition of the Rdd Server Class. This class is located inside PMS.

rddServer
-DBODBC : std::string -DBuser : std::string -DBpassword : std::string -db : wxDb * -connected : bool
+RDDServer(in DBODBC : std::string, in DBuser : std::string, in DBpassword : std::string) +retrieveRightsGenealogy(in right : std::string) : std::vector<std::string> +getPARGenealogy(in right : std::string) : std::vector<std::string> +ConnectDB() : bool -GetParents(in right : std::string, in rights : std::vector<std::string>) : void -GetChildren(in right : std::string, in rights : std::vector<std::string>) : void -setLastError(in lasterrorString : std::string, in code_error : unsigned int) : void +getErrors(in outputString & : std::string) : unsigned int

Figure. Rdd Server Class

11.3 User interface description

This module does not have user interface.

11.4 Technical and Installation information

In linker parameters, the header file (rddserver.h) has to be included where it is used and selected as input for the linker.

References to other major components needed	ODBC
Problems not solved	None
Configuration and execution context	User from this library should link wxWidgets library.

11.5 Draft User Manual

To get all parents for a right, call retreiveRightsGenealogy. To get children rights, call getPARGenealogy.

11.6 Examples of usage

To get parent rights:

```
std::vector<std::string> parents = RetreiveRightsGenealogy(“play”);
```

To get child rights:

```
std::vector<std::string> children = getPARGenealogy(“adapt”);
```

11.7 Integration and compilation issues

As this module does not use any system dependent library, it should be compatible with the different operating systems supported by wxWidgets. The only requirement is that an ODBC data source for rddserver has to be configured. The password for it is supplied in the licman.ini file.

11.8 Configuration Parameters

These values are stored in the file licman.ini.

Config parameter	Possible values
user	axmedis

password	axmedis
odbcrdds	rddserver

11.9 Errors reported and that may occur

The error reporting is bitwise (in a integer) and also a descriptive string is returned. Some of the error codes reported are warnings as they give advice of problems arose during license validation.

Error code	Description and rationales
8d = 00001000	Invalid Licman.ini
2d = 00000010	Cannot connect database
1d = 00000001	Unkown error

11.10 Formal description of algorithm

retrieveRightsGenealogy	
Method	retrieveRightsGenealogy
Description	The algorithm is created for checking rights hierarchy searching if right name X authorises the user to perform right name Y. From this purpose, we start calling GetChildren, getting all the children of a specific right, push them on a vector (rights) and do a recursive call within rights vector (passed as reference) and current sons as parameters.
Input parameters	std::string right
Output parameters	std::vector<std::string> & rights

getPARGenealogy	
Method	getPARGenealogy
Description	The algorithm is created for checking rights hierarchy searching if right name X authorises the user to perform right name Y. From this purpose, we start calling GetParents, getting all the parents of a specific right, push them on a vector (rights) and do a recursive call within rights vector (passed as reference) and current parents as parameters.
Input parameters	std::string right
Output parameters	std::vector<std::string> & rights

GetChildren	
Method	GetChildren
Description	Returns the sons for the current leaf, recursively auto-called with the obtained child.
Input parameters	std::string right
Output parameters	std::vector<std::string> & rights

GetParents	
Method	GetParents
Description	Returns the parents for the current leaf, recursively auto-called with the obtained parent.
Input parameters	std::string right
Output parameters	std::vector<std::string> & rights

12 Protection Info Manager

Module/Tool Profile		
Protection Info Manager		
Responsible Name	Víctor Rodríguez	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First version available	
Executable or Library/module (Support)	Library	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/protectioninfomanager	
Reference to the AXFW location of the demonstrator executable tool for internal download	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.)
Encryption Decryption Support		
Secure Cache Manager		
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
Openssl.0.9.7g		
Xercesc 2.6.0		

12.1 General Description of the Module

The Protection Info Manager offers different functionalities:

- Generation of keys, either for symmetric ciphering or asymmetric ciphering. Keys are represented with a pair of classes: KeyAX and RSAKeyAX.
- Storage and retrieval of Protection Info in the SecureCache. Protection information is represented in a class (CProtectionInfo).
- Storage and retrival of context information:
    - Context value itself
    - History hash
    - Number of executions of the resource

Information stored in the SecureCache is protected.

The real functionality does not lie in this module by itself but in the EncryptionDecryptionSupport (EncDecSup) and the SecureCache. The relationship of dependece can be seen in the following diagram:

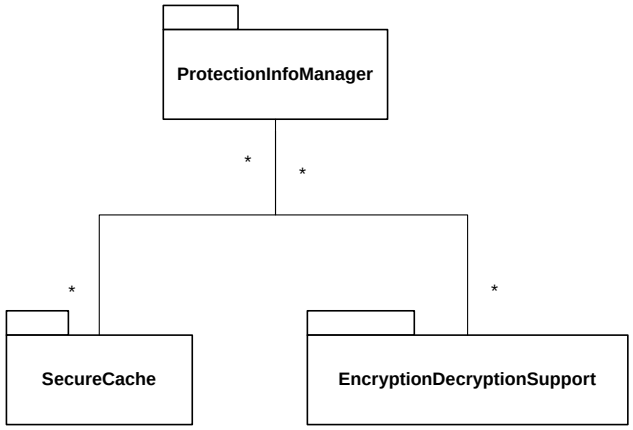


Figure. Protection Info Manager general class diagram

12.2 Module Design in terms of Classes

ProtectionInfoManager::**CProtectionInfoManager**
+CProtectionInfoManager() +CProtectionInfoManager(entrada dsn : string, entrada user : string, entrada passwd : string) +initializeSecureCache(entrada pathSC : const char*, entrada countryregion : const char*) : bool +insertProtectionInfo(entrada axoid : string, entrada objectversion : string, entrada protectionstamp : string, entrada pi : CProtectionInfo*) : bool +retrieveProtectionInfo(entrada axoid : string, entrada objectversion : string, entrada protectionstamp : string) : CProtectionInfo * +generateSymmetricKey(entrada lenghtKey : int) : KeyAX & +generateRSAKey(entrada lengthKey : unsigned int) : RSAKeyAX +deleteProtectionInfo(entrada axoid : string, entrada objectversion : string, entrada protectionstamp : string) : bool +getHistoryHash() : unsigned char * +storeHistoryHash(entrada hash[] : const unsigned char) : bool +getContextValue() : string +setContextValue(entrada contexto : string) : bool +getNumberOfExecutions(entrada userid : string, entrada objectid : string, entrada protectionstamp : string, entrada version : string, entrada derecho : string) : int +setNumberOfExecutions(entrada userid : string, entrada objectid : string, entrada protectionstamp : string, entrada version : string, entrada derecho : string, entrada i : int) : bool

12.3 Examples of usage

This sample code introduces a simple protection information.

```
CProtectionInfoManager pim;
CProtectionInfo *pri=new CProtectionInfo;
pri->setProtectionInfo("FA8963A3");
bool b=pim.insertProtectionInfo("axoid0","version0","protst",pri);
```

And in order to produce a new key, it can be considered

```
KeyAX clave=pim.generateSymmetricKey(1024);
unsigned char *c=new unsigned char[1024];
c=(unsigned char *)clave.getKey();
```

12.4 Integration and compilation issues

How to compile

In order to compile, the following environment variables must point to the path of the packages
OPENSSL -> Path to OpenSSL library
XERCESROOT -> Path to Xerces Library

12.5 Errors reported and that may occur

Error code	Description and rationales

N/a	This module relies on secure cache manager module to store information. Any failure in SecureCache Manager, will throw the same error.
N/a	This module relies on key generator module to generate keys. Any failure in that module will revert here also.

12.6 Formal description of Protection Info Manager operations

initializeSecureCache	
Method	initializeSecureCache(const char* pathSC, const char* countryregion)
Description	Initialises the securecache at the very first time. This function erases all possible data in the previus secure cache (if exists), it also creates a ODBC connector, and links it to the file that is going to be created. This operation also stores some values of the context like the country and region
Input parameters	pathSC Path (name of the file) where the Secure Cache will be created. countryregion The country and region where the pms client is located. It must have the next format: country{iso:ES}region{iso:ES-CT}
Output parameters	It returns true if Secure Cache has been created properly.

generateRSAKey	
Method	RSAKeyAX generateRSAKey(unsigned int lengthKey);
Description	Generate a new RSA pair of keys.
Input parameters	lenghtKey. The length of the keys (Optional, by default is 1024 bits)
Output parameters	KeyAX A RSA AXMEDIS pair of keys

generateSymmetricKey	
Method	KeyAX generateSymmetricKey(int lenghtKey);
Description	This method permits the creation of a key for protecting an AXMEDIS object.
Input parameters	lenghtKey. The length of the key
Output parameters	True on success

insertProtectionInfo	
Method	Bool insertProtectionInfo(string axoid, string objectversion,string protectionstamp, class CProtectionInfo *proti);
Description	This method stores the given protection information associated to an AXMEDIS object identifier, the object version and the protection stamp. Protection Information Manager will not physically store this information, but it will call the Light / Secure Cache Manager module who will in turn store it into the Secure Cache
Input parameters	Axoid, objectversion, protectionstamp. Describe the object proti Protection Information to be stored.
Output parameters	True on success

retrieveProtectionInfo	
Method	CProtectionInfo *retrieveProtectionInfo(string axoid, string objectversion, string protectionstamp);
Description	This method retrieves the requested protection information.

	The information needed to retrieve the protection information is the AXMEDIS object identifier, the object version and the protection stamp. This information will be requested to the Light / Secure Cache Manager module, which is in charge of retrieving it in the Secure Cache.
Input parameters	Axoid, objectversion, protectionstamp. Describe the object protectionstamp protection stamp to be stored
Output parameters	The protection info object.

deleteProtectionInfo	
Method	Bool deleteProtectionInfo(string axoid, string objectversion, string protectionstamp)
Description	Deletes the given protection information.
Input parameters	Axoid, objectversion, protectionstamp. Describe the object protectionstamp protection stamp to be deleted
Output parameters	True on success

getHistoryHash	
Method	unsigned char *getHistoryHash() const
Description	This method retrieves the last fingerprint.
Input parameters	
Output parameters	The history hash(). Memory must be freed with "delete".

storeHistoryHash	
Method	bool storeHistoryHash(const unsigned char hash[]);
Description	This method stores the last fingerprint.
Input parameters	
Output parameters	True on success

getContextValue	
Method	std::string getContextValue() const;
Description	Gets the application context value (territory etc.)
Input parameters	
Output parameters	A string with the context value

setContextValue	
Method	Bool setContextValue(std::string context)
Description	Ses the application context value (territory etc.)
Input parameters	A string with the context value
Output parameters	True on success

getNumberOfExecutions	
Method	int getNumberOfExecutions(string userid, string objectid, string protectionstamp, string version,string derecho);
Description	Gets the number of times that a method has been executed.
Input parameters	User, object description (id, protection stamp, version) and right.
Output parameters	The number of times that a method has been executed.

setNumberOfExecutions	
Method	bool setNumberOfExecutions(std::string userid, string objectid, string protectionstamp, string version, string derecho,int i);
Description	Sets the number of times that a method can been executed.
Input parameters	User, object description (id, protection stamp, version) and right, and number of times that the object can be used.
Output parameters	True on success.

13 Key Generator

Module/Tool Profile		
Key Generator		
Responsible Name	Víctor Rodríguez	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First version available	
Executable or Library/module (Support)	Library	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/keygen	
Reference to the AXFW location of the demonstrator executable tool for internal download	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	N/A	
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
Openssl.0.9.7g		
Xercesc 2.6.0		

13.1 General Description of the Module

Key generator module generates cryptographic keys, for both symmetric and asymmetric ciphering.

It relies on OpenSSL library to implement its functionality. Also takes advantage of EncDecSup module offered functionality.

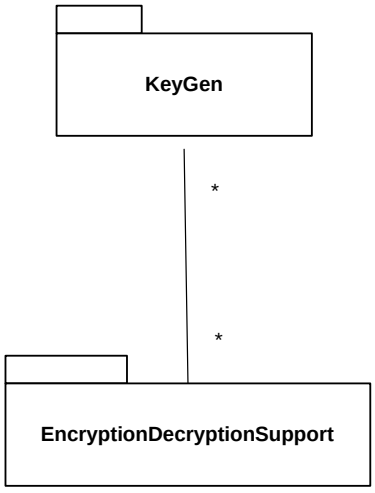


Figure: Key Generator general architecture

13.2 Module Design in terms of Classes

It consists of several classes. However, the access point to the functionality is the class called KeyGenerator.

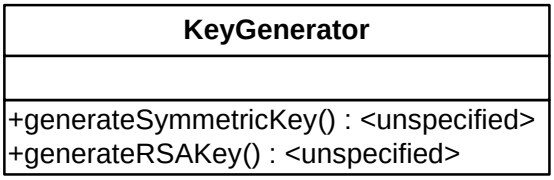


Figure: Key generator public functions

Data is returned as RSAKey or DSAKey objects.

13.3 Examples of usage

The KeyGenerator class methods must be accessed through the Protection Info Manager and therefore no examples are provided.

13.4 Integration and compilation issues

How to compile

In order to compile, the following environment variables must point to the path of the packages
OPENSSL -> Path to OpenSSL library

13.5 Errors reported and that may occur

Error code	Description and rationales
N/a	Very weird conditions would lead to a failiure of methods of this modules (i.e. a sudden O.S. denial of memory allocation etc.)

13.6 Formal description of the Key Generator funcionality

generateRSAKey	
Method	Static RSAKey& generateRSAKey(unsigned int lengthKey);
Description	Generate a new RSA pair of keys, this method is overload, the length of the key can be spcecified or by default will be 1024 bits
Input parameters	lengthKey The length of the keys (Optional) Default 1024
Output parameters	A RSA AXMEDIS pair of keys

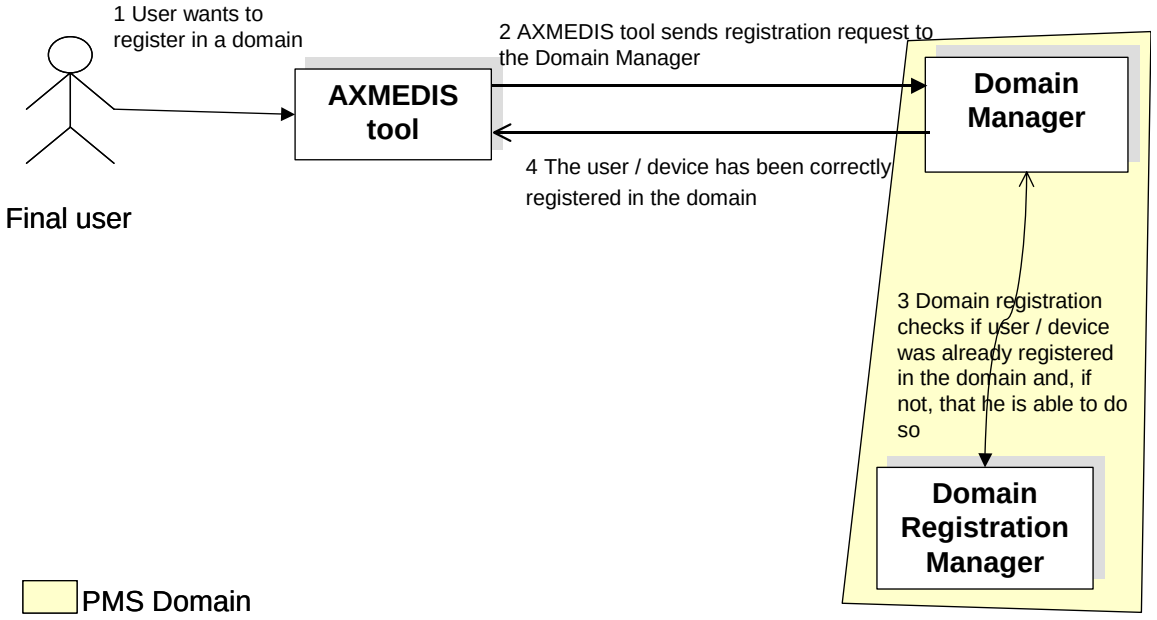
generateSymmetricKey	
Method	static KeyAX &generateSymmetricKey(unsigned int lengthKey);
Description	Generates a new symmetric key
Input parameters	lengthKey The length of the key
Output parameters	The AXMEDIS key

14 Domain Manager

This module, together with the domain registration manager, keeps track of the users that are associated to a domain, giving them the possibility to register, unregister and, in general, to manage the domains available for a user.

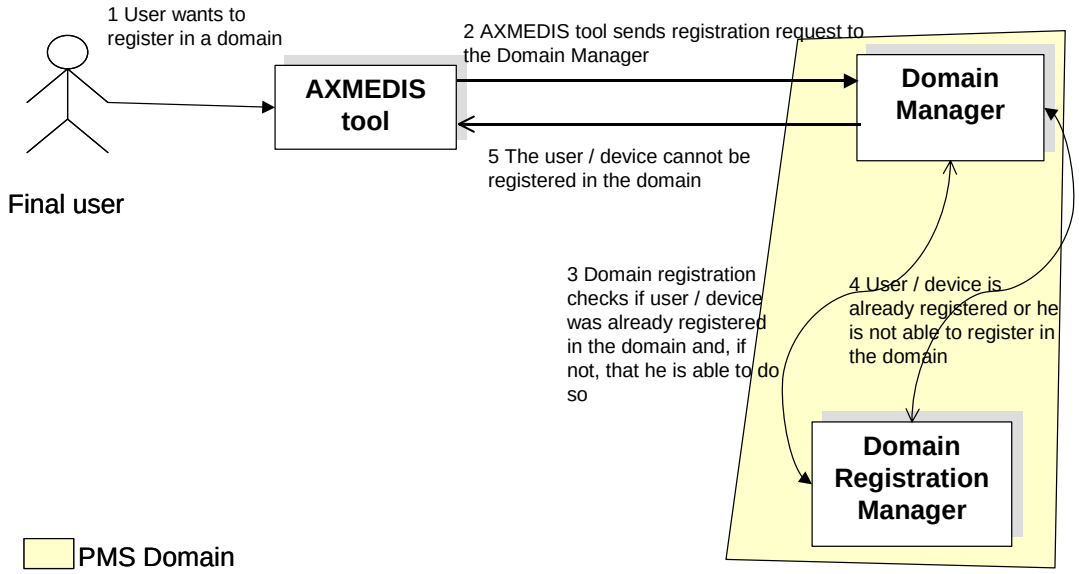
14.1 Domain related scenarios

The scenario described defines how a user can be registered in a domain.



Successful registration in a domain

The scenario described next shows how a user cannot be registered in a domain.



Unsuccessful registration in a domain

Module/Tool Profile		
Domain Manager		
Responsible Name	Rubén Barrio	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	Implemented	
Executable or Library/module (Support)	Application	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Applications/domainmanager/source	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Applications/domainmanager/bin/win32	
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.)
PMS Domain		
Secure Cache Manager		
Secure Cache		
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

14.2 General Description of the Module

The Domain manager module has to keep track of the domain the user is assigned. Together with the Domain registration manager, provides the functionality for allowing the user to register in the domain, unregister and perform actions only available at domain level, based on the licenses at domain level.

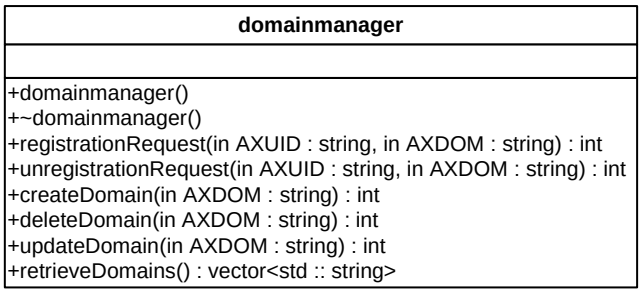
The Domain manager and Domain registration manager are located in the PMS Domain Factory and Home modules.

Domain manager and Domain registration manager will be implemented as a C++ library to facilitate integration with current implemented modules to be used inside PMS Domain (Factory, Home).

The relationship with other modules is shown in the general description section. The functionality for accessing domain facilities will be provided by the PMS Domain WS (which will be very similar to the current PMS Server WS).

The access to a domain should be requested by a final user application in the user side (which integrates Protection Processor and PMS Client). After the needed checks, the domain the user has registered to is stored in the secure cache, as this information is stored in the action logs sent to AXCV when a user action is requested.

14.3 Module Design in terms of Classes



14.4 Formal description of algorithm

registrationRequest	
Method	registrationRequest
Description	A user tries to register in the Domain
Input parameters	std::string AXUID: User identifier std::string AXDOM: Domain identifier
Output parameters	std::int result

unregistrationRequest	
Method	unregistrationRequest
Description	A user tries to unregister from the Domain
Input parameters	std::string AXUID: User identifier std::string AXDOM: Domain identifier
Output parameters	std::int result

createDomain	
Method	createDomain
Description	Creation of a domain given its identifier
Input parameters	std::string AXDOM: Domain identifier
Output parameters	std::int result

deleteDomain	
Method	deleteDomain
Description	Deletion of a domain given its identifier
Input parameters	std::string AXDOM: Domain identifier
Output parameters	std::int result

updateDomain	
Method	updateDomain
Description	Update of a domain given its identifier
Input parameters	None
Output parameters	std::int result

retrieveDomains

Method	retrieveDomains
Description	Retrieves the list of registered domains
Input parameters	std::string AXDOM: Domain identifier
Output parameters	std::vector<std::string> result

15 Domain Registration Manager

Module/Tool Profile		
Domain Registration Manager		
Responsible Name	Rubén Barrio	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	Implemented	
Executable or Library/module (Support)	Library	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/domainregistrationmanager	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Framework/bin/domainregistrationmanager	
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	N/A	
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.)
Domain Manager		

Secure Cache		
PMS Domain		
Secure Cache manager		
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

15.1 General Description of the Module

The Domain registration manager module allows the registration of a user in a domain. It provides the functionality to register users (used by the Domain Manager) and

The Domain manager and Domain registration manager are located in the PMS Domain Factory and Home modules.

Domain manager and Domain registration manager will be implemented as a C++ library to facilitate integration with current implemented modules to be used inside PMS Domain (Factory, Home).

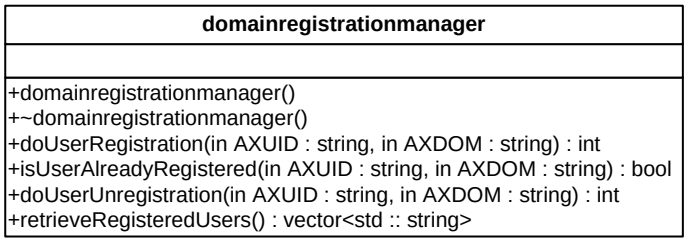
The relationship with other modules is shown in the general description section. The functionality for accessing domain facilities will be provided by the PMS Domain WS (which will be very similar to the current PMS Server WS).

The access to a domain should be requested by a final user application in the user side (which integrates Protection Processor and PMS Client). After the needed checks, the domain the user has registered to is

stored in the secure cache, as this information is stored in the action logs sent to AXCV when a user action is requested.

Specifically, Domain Registration Manager should check that the registration of a user in a domain is feasible, controlling that he is not already registered in that domain. On the user side, it has also to be checked that the user does not belong to another domain, or ask for unregistration before registration, as it is a requirement that a final user can only belong to one domain at a time.

15.2 Module Design in terms of Classes



15.3 Formal description of algorithm

DoUserRegistration	
Method	DoUserRegistration
Description	Domain manager requests user registration
Input parameters	std::string AXUID: User identifier std::string AXDOM: Domain identifier
Output parameters	std::int result

DoUserUnregistration	
Method	DoUserUnregistration
Description	Domain manager requests user unregistration
Input parameters	std::string AXUID: User identifier std::string AXDOM: Domain identifier
Output parameters	std::int result

isUserAlreadyRegistered	
Method	isUserAlreadyRegistered
Description	Check if user is already registered in the domain
Input parameters	std::string AXUID: User identifier std::string AXDOM: Domain identifier
Output parameters	std::int result

RetrieveRegisteredUsers	
Method	retrieveRegisteredUsers
Description	Retrieve the list of registered users in a domain
Input parameters	None
Output parameters	std::vector<std::string> result

RetrieveDomains	
Method	RetrieveDomains
Description	Retrieve the list of all domains
Input parameters	None
Output parameters	std::vector<std::string> result

getDomainOfUser	
Method	getDomainOfUser
Description	Get the domain where a user is registered
Input parameters	std::string AXUID: User identifier
Output parameters	std::string: Domain ID

16 Rights Expression Translator

Module/Tool Profile		
Rights Expression Translator		
Responsible Name	Xavier Maroñas	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	Implemented	
Executable or Library/module (Support)	Library	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Framework/source/rightsexpressiontranslator	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/newrepos/Framework/bin/rightsexpressiontranslator	
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	N/A	
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.)
PMS		
Formats Used	Shared with	format name or reference to a section
MPEG-21 REL		

OMA DRM REL		
MPEG-21 REL Profiles		
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
Xerces		
Licensemodel		
licensemanager		
Omalicensemodel		
omalicensemanager		

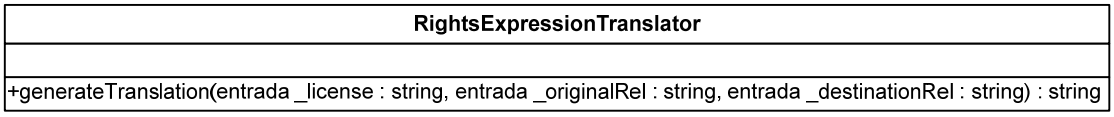
16.1 General Description of the Module

Rights Expression Translator involves the translation of rights expressions from one rights expression language to another. For the moment, translation between MPEG-21 REL and OMA DRM REL (rights expression language based on ODRL). This module will evolve as new rights expression language appear in the state of the art. For the moment, some profiles for MPEG-21 REL are being defined and its translation is foreseen.

Translation between rights expression languages may be done using different techniques, based on XML tools (like Xerces or XSL) or based on operations over relational databases modeling licenses expressed on different rights expression languages.

The translation could be caused by several reasons: the device does not support a specific rights expression language, an authorization can only be done using one rights expression language, the tool does not support the rights expression, etc.

16.2 Module Design in terms of Classes



16.3 Errors reported and that may occur

Error code	Description and rationales
512d = 1000000000	Invalid Licman.ini
256d = 100000000	XSD File Invalid
128d = 10000000	Original License Invalid
64d = 01000000	XSD File not found
32d = 00100000	License Not found
16d = 00010000	Cannot create final license, path invalid
8d = 00001000	Cannot Connect Database
2d = 00000010	Cannot create final license, cannot transform
1d = 00000001	Unknown error

16.4 Formal description of Rights Expression Translator

The Rights Expression Translator is a module to input license in xml and convert to another. The map will be done getting the original XML license in its model format. Depending on the destinationLanguage and origin Language the algorithm will call the appropriate functions to transform each object of the original license model, into the new one. After that, the system calls the last method that generate the new XML license from the one in the license model.

Rights Expression Translator	
Method	GenerateTranslation
Description	Converts from one license type to another getting the license as input and the destinationLanguage as a string.
Input parameters	string originalLicense: the complete XML license. string originalRel: string to identify the original license language string destinationRel: string to identify the REL in wich the license will be translated.
Output parameters	string: Full XML license converted to destinationLanguage

17 Protection Support for Mobiles

Module/Tool Profile		
Protection Support for Mobiles		
Responsible Name	Rubén Barrio	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Not Implemented	
Status of the implementation	Not Implemented	
Executable or Library/module (Support)	Module	
Single Thread or Multithread		
Language of Development	C++/Java (Depending on the device)	
Platforms supported	To be defined, depending on the devices supported	
Reference to the AXFW location of the source code demonstrator	N/A	
Reference to the AXFW location of the demonstrator executable tool for internal download	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	N/A	
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.)
PMS		
Formats Used	Shared with	format name or reference to a section
Protocol Used	Shared with	Protocol name or reference to a

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

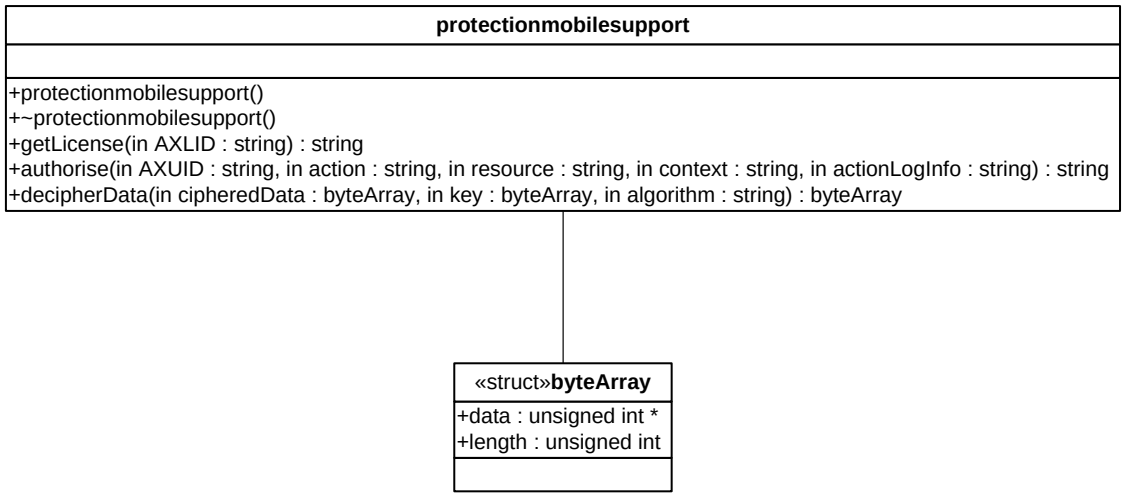
17.1 General Description of the Module

This module will provide basic functionality for providing protection support for Mobiles.

The functions defined are part of the existing Protection Management Support Client. Depending on the mobile equipment used, the implementation language and the method signature may experience some changes.

The language of development will depend on the device supported. PDA's will mainly use C++, but other kind of devices will need J2ME support. In the last case, the functionality provided should be translated from C++ language to Java, as PMS Client is currently implemented in C++.

17.2 Module Design in terms of Classes



17.3 Formal description of Protection Support for Mobiles

getLicense	
Method	getLicense
Description	This method retrieves a license given its identifier
Input parameters	Std::string **AXLID**
Output parameters	Std::string licenseResult, contains the license or a message “wrong license”

authorise	
Method	authorise
Description	This method asks for user authorisation for content consumption
Input parameters	Std::string **AXUID**, User identifier Std::string **action**, Action to be done over the object Std::string **resource**, AXMEDIS object to be consumed Std::string **actionLogInfo**, Action Log information expressed as a string for facilitating use in a mobile equipment
Output parameters	Std::string result, contains information for unprotecting the object to be used, if it is ciphered.

decipherData	
Method	decipherData
Description	This method deciphers a protected object
Input parameters	ByteArray **cipheredData**, Byte representation of the ciphered data ByteArray **key**, Key for unprotecting the object Std::string **algorithm**, Algorithm used for deciphering data
Output parameters	ByteArray **result**, contains the deciphered information

verifyUser	
Method	verifyUser
Description	This method is called by the Protection Processor and reaches AXCV through PMS Server. It can be used to verify the status of a user, optionally inside a domain. It verifies if the user is registered in the specified domain (if present) and checks that the user status and registration deadline are valid, so that the user can still use the AXMEDIS tools and the AXMEDIS

	framework.
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID)
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: Verification OK -1: invalid AXID -2: user is not registered -3: user is blocked -4: user domain mismatch -5: user registration deadline expired When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur, but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the deadline has expired or not).

certifyForMobile	
Method	certifyForMobile
Description	This method is called by the Protection Processor and reaches AXCV through PMS Server. It is used to certify that the original tool has not been modified and to activate it. It creates a new entry in the CerTools table of the AXCS database, which associates the user, tool and device and returns to the Protection Processor an activation code, a tool identifier and a PKCS12 structure **with the tool certificate and private key issued by AXMEDIS CA (EJBCA).**
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axrtid**: identifier of the registered AXMEDIS tool xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool xsd:string **regDeadline**: registration deadline of the installed tool.
Output parameters	CertificationResult complex type formed by sequence of: xsd:string **axtid**, the identifier of the installed tool associated to a user and device. xsd:int **certificationResult**, which indicates the result of the certification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: tool not registered (RegTools table) -7: registered tool is blocked -8: received tool deadline exceeds registered tool deadline (user and tool have been blocked) -9: received tool deadline has expired -10: registered tool fingerprint mismatch. Tool has been manipulated (user and tool have been blocked) -11: user-tool-device had already been certified. New tool certificate should be created -20: error updating user status in database -21: error inserting new entry in CerTools table -22: error in AXSupervisor when communicating with database -30: internal AXCV error When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur,

	but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the registered tool is blocked or not, or if the tool fingerprint did match or not).

verifyForMobile	
Method	verifyForMobile
Description	This method is called by the Protection Processor and reaches AXCV through PMS Server. It is used to verify that the tool installed on a device has neither been modified nor blocked, that the user is not blocked and that the registered tool is not blocked. It is also responsible for resynchronizing the offline tool operation through AXMEDIS Supervisor (AXS).
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a device). String (base64) **toolFingerprintDigest**: md5 hash of the full fingerprint (software and hardware parts) of the installed tool.
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: AXTID does not exist -7: installed (and certified) tool is blocked -8: tool deadline has expired -9: toolFingerprintDigest (toolFingerprint hash) mismatch -10: toolFingerprint mismatch (user and tool have been blocked) -11: registered tool is blocked -12: user has been blocked and installed tool has been blocked again -13: tool has been blocked -20: error updating user status in database -21: error updating tool status in database -22: error updating LastFPPA in database -23: error retrieving regtool data from database -24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog -25: error in AXSupervisor when communicating with AXCS accounting database in storeSID -30: internal AXCV error When an error code *x* is returned, it means that all the possible errors *y*, *x*<*y*<0 did not occur, but all possible errors *y*<*x* have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the received tool deadline has expired or not).

reverifyForMobile	
Method	reverifyForMobile
Description	This method is similar to verify method (see previous). It must be called when the verify method fails because of the tool fingerprint hash doesn't match (error code: –12) to perform a new verification with the full fingerprint. Thus, the reverify method has the same input parameters as the verify method except the full tool fingerprint, which has to be sent instead of

	the hash.
Input parameters	xsd:string **axid**: identifier of the AXMEDIS final user (AXUID) or B2BUser (AXCID, AXDID, AXCSID or AXTPID) xsd:string axtid: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a device). xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool.
Output parameters	VerificationResult complex type formed by sequence of: xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration: 0: OK -1: invalid AXID -2: user not registered -3: user blocked -4: user domain mismatch -5: user registration deadline expired -6: AXTID does not exist -7: installed (and certified) tool is blocked -8: tool deadline has expired -9: toolFingerprintDigest (toolFingerprint hash) mismatch -10: toolFingerprint mismatch (user and tool have been blocked) -11: registered tool is blocked -12: user has been blocked and installed tool has been blocked again -13: tool has been blocked -20: error updating user status in database -21: error updating tool status in database -22: error updating LastFPPA in database -23: error retrieving regtool data from database -24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog -25: error in AXSupervisor when communicating with AXCS accounting database in storeSID -30: internal AXCV error When an error code *x* is returned, it means that all the possible errors *y*, $x<y<0$ did not occur, but all possible errors $y<x$ have not been checked. (E.g error code –2 means that AXID is valid but doesn't inform about if the user is blocked or not, or if the received tool deadline has expired or not).

18 Secure cache manager

Module/Tool Profile		
Secure Cache Manager		
Responsible Name	Víctor Rodríguez	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First version available	
Executable or Library/module (Support)	Library	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/securecache	
Reference to the AXFW location of the demonstrator executable tool for internal download	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	N/A	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	Encrypted information in the client side is ciphered with a static key. The key must be kept in the client side, what constitutes a non-solvable problem. At least, it could be considered changing the key every time that the cache has connection, accepting a key passed from the PMS Server	
Major pending requirements	None	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
License model		
Encryption decryption support		
Formats Used	Shared with	format name or reference to a section

Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
Wxwindows 2.4.2		
Openssl.0.9.7g		
Xercesc 2.6.0		

18.1 General Description of the Module

This module provides the functionality needed to access to information stored in the Local Cache
The dependency graph is shown here:

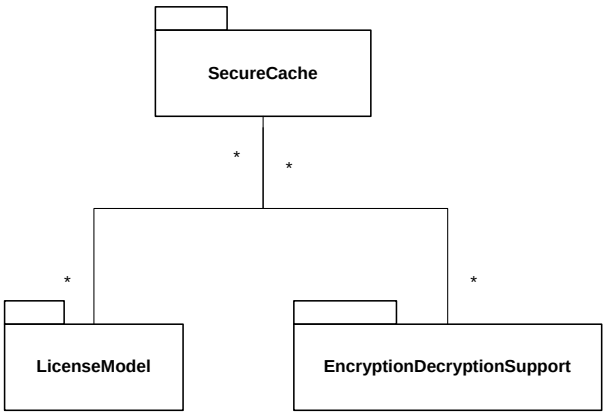
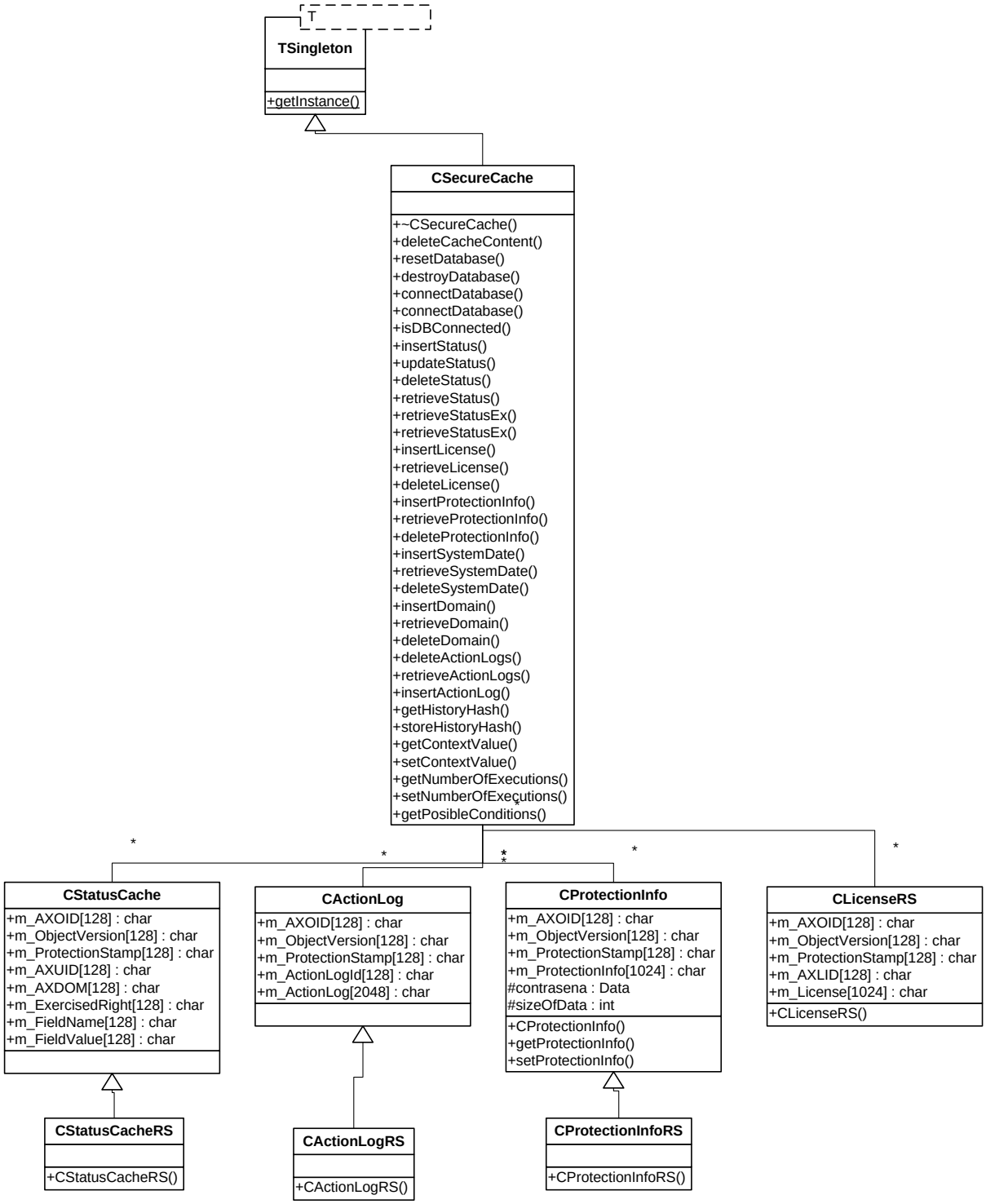


Figure. Secure Cache general architecture

18.2 Module Design in terms of Classes



The main class is CsecureCache. As it works with StatusCahe, ActionLogs, Licenses and ProtectionInfo, there are classes for each of them respectively. Each of these classes have another associated class, the RecordSet class (the same name has been kept with RS as apendix).

RecordSet classes help database operations through simple operations, as Recordset classes derive also from the Recordset class, in turn provided by the wxWindows platform.

18.3 Technical and Installation information

References to other major components needed	SQLLite driver (publicly available and with no license restrictions) must be provided during installation.
Problems not solved	Where to store and how to generate the key to cipher locally stored information?

18.4 Draft User Manual

SecureCache is a singleton class that ensures that one and only one SecureCache is instantiated in the same process. Singleton classes must be instantiated properly, and the way of doing this is by calling the getInstance() method. For example:

```
CsecureCache *cache=CsecureCache::getInstance()
```

And then the pointer can be used normally. Actually, only the first of the calls is the one who calls the contructor. This first invocation is importante, as optionally initialization can be here for (for example initializating the database and thus become a lenghty operation)
Before performing any read/write operation, it should be considered checking that retrieval and storage work properly, by calling the isDBConnected() method.

18.5 Examples of usage

The usage of the methods of secure cache is quite strightforward.

```
CSecureCache *cache=CSecureCache::getInstance();
if (cache->isDBConnected())
{
      cache->deleteSystemDate();
      cache->insertSystemDate();
}
```

18.6 Integration and compilation issues

As seen in another modules, some evironment variables must be set.

OPENSSL -> Path to OpenSSL library
XERCESROOT -> Path to Xerces Library
WXWIN -> Path to wxWidgets library.

The secure cache stores the information information in a very flexible manner. Actually, it writes the data through an ODBC driver. This generic capability allows very different systems of information storage and retrieval.

The current specification states that no database is available in the client side, therefore all the info must remain in files. The adopted solution has been the use of the SQLlite ODBC driver, that allows to store information in database embedded in a single file.

- ODBC access is granted through wxWindows. WxWindows, by default, does not include database capabilities between its functionalities, and this characteristics has to be enabled: In the adequate file `setup.h` provided with wxWindows, the define `#define wxUSE_ODBC` has to be changed 1 (Default was 0). In the same file, `#define wxODBC_FWD_ONLY_CURSORS` has to be set to 0 (default was 1).
- ODBC installation is different from Windows (ultimately, it means changing some entries in the windows registry) to Unix (changing the file `odbc.ini`)

18.7 Formal description of Secure Cache Manager algorithms

deleteCacheContent	
Method	bool deleteCacheContent();
Description	This method deletes the whole content of the Secure Cache. It should also invalidate it, as it is called when a tool has not been verified.
Input parameters	
Output parameters	Returns true on success.

resetDatabase	
Method	bool resetDatabase(string filename="securecache.db");
Description	Prepares a database to be used by SecureCache. This function creates a database, with the valid filename given. A DSN ("securecache") is created for future references. After this function (if successful) connection is stablished also. bugs In case other DSN with the name "securecache" (rather strange case), it would fail.
Input parameters	filename Relative or absolute path to a given file. i.e. "c:\\securecache.db"
Output parameters	Returns true on success.

destroyDatabase	
Method	bool destroyDatabase();
Description	Destroys the database and its associated DSN. This operation is not reversible, and data will be lost forever.
Input parameters	
Output parameters	Returns true on success.

connectDatabase	
Method	bool connectDatabase();
Description	Tries to connect the cache to the default database. Tries to connect to a SQLite database whose DSN is "securecache". If it fails (i.e. DSN does not exist, file does not exist), it returns false.
Input parameters	
Output parameters	Returns true on success.

isDBConnected	
Method	bool isDBConnected();
Description	Returns true if the SecureCache is properly connected to the database.
Input parameters	
Output parameters	True if the SecureCache is properly connected to the database.

insertStatus	
Method	bool insertStatus(string axoid,string objectversion, string protectionstamp,string right,string statusname, string statusvalue);
Description	This method stores some status information associated to AXMEDIS objects usage in order to be able to perform local authorizations. This information has to be stored ciphered. It is used by the Authorization support module.
Input parameters	Object given by axoid, version and protection stamp. Rights and pairs of status name and value.
Output parameters	Return true on success.

updateStatus	
Method	bool updateStatus(string axoid,string objectversion, string protectionstamp,string right,string statusname, string statusvalue);
Description	This method updates some status information associated to AXMEDIS objects usage in order to be able to perform local authorizations. This information has to be stored ciphered. It is used by the Authorization support module.
Input parameters	Object given by axoid, version and protection stamp. Rights and pairs of status name and value.
Output parameters	Return true on success.

deleteStatus	
Method	bool deleteStatus(string axoid, string objectversion, string protectionstamp);
Description	This method deletes status information associated to AXMEDIS objects usage
Input parameters	Object given by axoid, version and protection stamp.
Output parameters	Return true on success.

retrieveStatus	
Method	class CStatusCache *retrieveStatus(string axoid,string objectversion, string protectionstamp);
Description	This method retrieves status information associated to AXMEDIS objects usage in order to be able to perform local authorizations. It is used by the Authorization support module.
Input parameters	Object given by axoid, version and protection stamp.
Output parameters	Returns a cstatuscache object or NULL if error ocurred or no status cache existed.

insertLicense	
Method	bool insertLicense(string axoid, string objectversion, string protectionstamp, class License *license);
Description	This method allows the insertion of a license into the local cache by the LicenseManager module.
Input parameters	Object given by axoid, version and protection stamp. Pointer to licese to be inserted.
Output parameters	Return true on success.

retrieveLicense

Method	class License *retrieveLicense(string axoid,string objectversion, string protectionstamp);
Description	This method allows the retrieval of a license from the local cache by the LicenseManager module.
Input parameters	Object given by axoid, version and protection stamp. Pointer to licese to be inserted.
Output parameters	The license to be retrieved, or NULL.

deleteLicense	
Method	bool deleteLicense(string axlid);
Description	This method allows the deletion of a license from the local cache by the LicenseManager module.
Input parameters	License ID to be deleted.
Output parameters	True on success.

insertProtectionInfo	
Method	bool insertProtectionInfo(string axoid, string objectversion, string protectionstamp, CProtectionInfo *protectioninfo);
Description	Stores the protection information.
Input parameters	Object given by axoid, version and protection stamp. Pointer to licese to be inserted.
Output parameters	True on success.

retrieveProtectionInfo	
Method	CProtectionInfo *retrieveProtectionInfo(string axoid, string objectversion, string protectionstamp);
Description	This method retrieves the protection information
Input parameters	Object given by axoid, version and protection stamp.
Output parameters	The protection info to be retrieved, or NULL.

insertSystemDate	
Method	bool insertSystemDate();
Description	Stores the current system date in order to perform local checks over the operations done over the Secure Cache. It erases any other previously introduced date
Input parameters	None
Output parameters	True on success.

retrieveSystemDate	
Method	string retrieveSystemDate();
Description	Returns the last system date stored in o the Secure Cache.
Input parameters	None
Output parameters	A string with the system date. The format is given by the Operative System. The string is empty in case an error ocurred.

deleteSystemDate	
Method	bool deleteSystemDate();
Description	Deletes the system date from the Secure Cache.
Input parameters	None
Output parameters	True on success.

insertDomain	
Method	bool insertDomain(string axdom);
Description	Stores the domain a user is registered to. This information has to be stored ciphered.
Input parameters	Domain to be inserted.
Output parameters	True on success.

retrieveDomain	
Method	string retrieveDomain();
Description	Returns the domain a user is registered to.
Input parameters	None
Output parameters	Retrieves the domain of the user.

deleteDomain	
Method	bool deleteDomain(string axdom);
Description	Deletes the domain.
Input parameters	Domain to be eliminated.
Output parameters	True on success.

retrieveActionLogs	
Method	vector<CActionLog> retrieveActionLogs();
Description	This method retrieves the action logs stored into the Secure Cache. It is called by the Content Consumption status module.
Input parameters	None
Output parameters	A vector with the action logs.

insertActionLog	
Method	bool insertActionLog(string axoid, string objectversion, string protectionstamp, CActionLog *actlog);
Description	This method inserts an action log into the Secure Cache. It is called by the Content Consumption status module.
Input parameters	Object given by axoid, version and protection stamp. Pointer to the action log to be inserted.
Output parameters	True on success.

retrieveStatusEx

Method	vector<CStatusCache> retrieveStatusEx(string axoid,string objectversion, string protectionstamp, string axuid);
Description	A version of retrieveStatus with different parameters.
Input parameters	Object given by axoid, version and protection stamp, user given by axuid.
Output parameters	Vector with the cache status.

retrieveStatusEx	
Method	vector<CStatusCache> retrieveStatusEx(string axoid,string objectversion, string protectionstamp, string axuid, string axlid,string right);
Description	A version of retrieveStatus with different parameters.
Input parameters	Object given by axoid, version and protection stamp, user given by axuid.
Output parameters	Vector with the cache status.

getHistoryHash	
Method	byte *getHistoryHash() const;
Description	This method retrieves the last fingerprint.
Input parameters	None
Output parameters	The history hash(). Memory must be freed with "delete"

setHistoryHash	
Method	Bool setHistoryHash(const byte hash[])
Description	This method stores the last fingerprint.
Input parameters	The history hash().
Output parameters	True on success

getContextValue	
Method	string getContextValue() const;
Description	Gets the application context value (territory etc.)
Input parameters	None
Output parameters	A string with the context value.

setContextValue	
Method	bool setContextValue(string contexto);
Description	Sets the application context value (territory etc.)
Input parameters	The context value
Output parameters	True on success.

getNumberOfExecutions	
Method	int getNumberOfExecutions(string userid, string objectid, string protectionstamp, string version, string derecho);
Description	Get the number of times that an object has been executed for a given user, object and right
Input parameters	User, object and right.
Output parameters	The number of times that the object has been executed or -1 if error.

setNumberOfExecutions	
Method	int setNumberOfExecutions(string userid, string objectid, string protectionstamp, string version, string derecho,int i);
Description	Set the number of times that an object has been executed for a given user, object and right
Input parameters	User, object, right and the number of times i.
Output parameters	True on success.

getPosibleConditions	
Method	vector <ConditionsAndMore> getPosibleConditions(string AXUID,string right,string AXOID);
Description	Gets all the possible conditions for a user, a right and an object
Input parameters	User, object, right.
Output parameters	Set of conditions for the given user right and object

19 Secure Cache

Module/Tool Profile		
Secure Cache		
Responsible Name	Víctor Rodríguez	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First version available	
Executable or Library/module (Support)	Database	
Single Thread or Multithread		
Language of Development	N/A	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/securecache	
Reference to the AXFW location of the demonstrator executable tool for internal download	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	N/A	
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
SQL		
Protocol Used	Shared with	Protocol name or reference to a

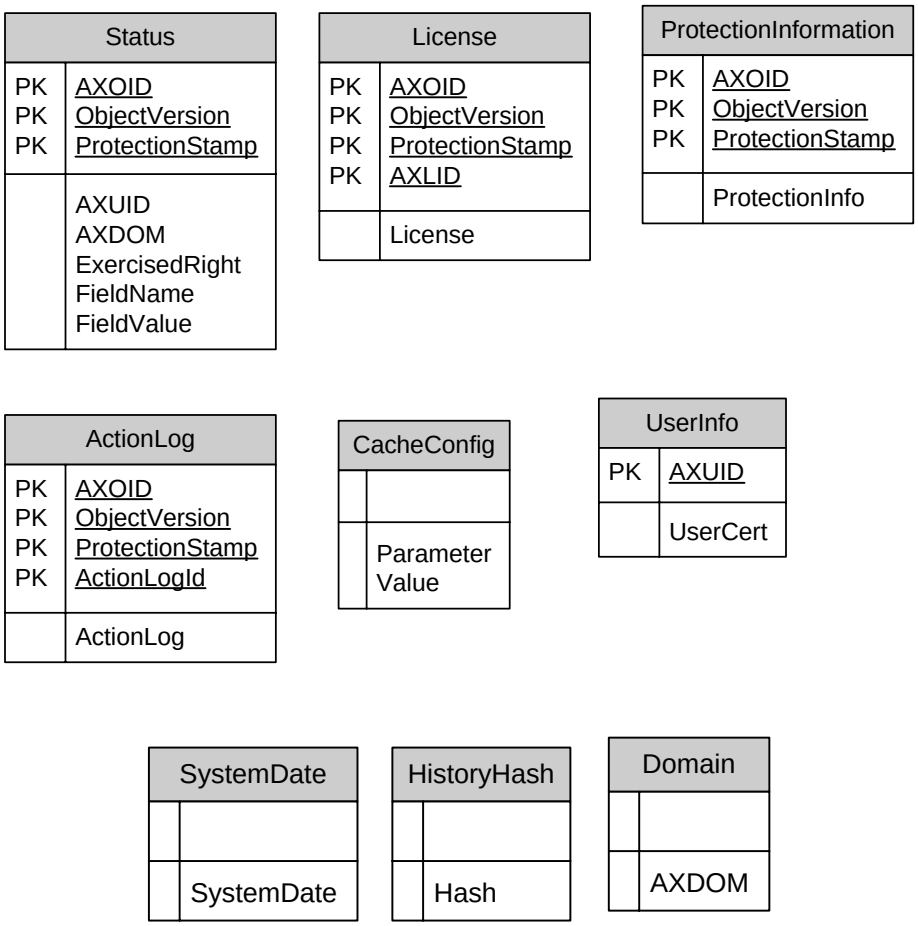
		section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

19.1 General Description of the Module

This section describes how information is structured and stored in the Secure Cache.

It is a constraint that no database can be installed in the client side. This is desired in order to avoid administration tasks in the client, and to avoid the installation of complex database managers, that might be accessed by the users. The information of the Secure Cache is stored in an embedded SQLite database, under the form of a single binary file. The only needed thing is a dynamic library (that could be also statically linked within the executable), and the rest of the code is embedded within the application.

The data structure is as follows in the next diagram:



The folowing list describes the columns in the tables:

- AXOID: AXmedis Object Identification
- ObjectVersion: Version of the Axmedis object.
- ProtectionStamp: Protection stamp of the object.
- AXUID: Axmedis UserID.
- AXLID: Axmedis License ID.
- License: The license as a XML file.
- ActionLog: String defining an action log as it has been described in the Axmedis documentation.
- UserCert: Certificate of user. Format pending to be determined (either PKCS or PEM).
- ActionLogID: ID of the action log.
- Exercised Right: MPEG21 REL Right
- FieldName/FieldValue to express properties/values couples.
- Parameter/Value in the table CacheConfig, to store unique variables with a general purpose, such as the system date, the history hash etc.

20 Content consumption status

Module/Tool Profile		
Content Consumption Status		
Responsible Name	Víctor Rodríguez	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	First version available	
Executable or Library/module (Support)	Library	
Single Thread or Multithread		
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/Framework/source/contentconsumptionstatus	
Reference to the AXFW location of the demonstrator executable tool for internal download	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)	N/a	
Test cases location	N/a	
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

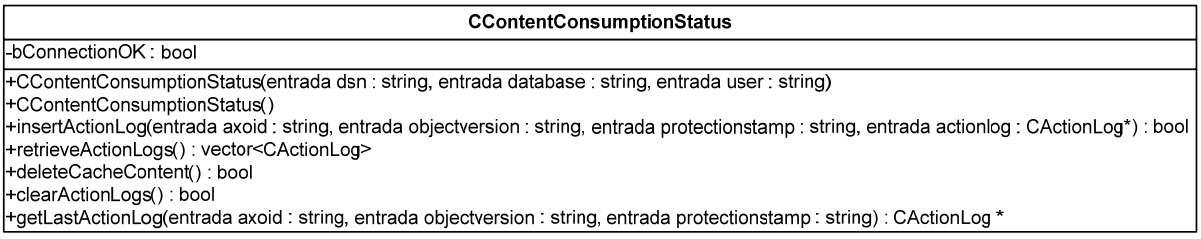
Protocol Used	Shared with	Protocol name or reference to a section
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
Wxwindows 2.4.2		
Openssl.0.9.7g		
Xercesc 2.6.0		

20.1 General Description of the Module

Content Consumption status module keeps track of the actions performed by the user when he is working in an unconnected environment

20.2 Module Design in terms of Classes

This module is entirely defined and implemented by a single class called CcontemtionConsumptionStatus. It also makes use of another class, CActionLog, defined in the SecureCache module (see above).



20.3 Examples of usage

The next example shows how to store a single log information piece in the secure cache.

```
CActionLog actionlog;
CcontentConsumptionStatus ccs("securecache","C:\\tmp\\cache","");
sprintf(actionlog.m_ActionLog,"10/02/2006 > Hello world");
ccs.insertActionLog(getNewUUID(),"-","-",&actionlog);
```

20.4 Errors reported and that may occur

Error code	Description and rationales
N/a	This module depends on the SecureCache to store and retrieve information. In case secure cache fails, error will be thrown also here.

20.5 Formal description of Content Consumption Status methods

insertActionLog	
Method	bool insertActionLog(string axoid, string objectversion, string protectionstamp, CActionLog *actionlog);
Description	This method inserts an action log inside the Secure Cache through the secure cache manager. The action log is identified by the AXMEDIS Object, Version and protection stamp.
Input parameters	Object described in terms of axoid, version and protection info. Action log to be inserted.
Output parameters	True on success.

retrieveActionLogs	
Method	vector<CActionLog> retrieveActionLogs();
Description	This method retrieves all the action logs inside the Secure Cache when the user connects to the PMS server in order to verify and synchronize the actions performed off-line with the previously performed actions
Input parameters	none
Output parameters	Vector with the Action Logs.

deleteCacheContent	
Method	bool deleteCacheContent();
Description	This method is for deleting the contents of the cache. It can be used when the tool cannot be verified because of illegal manipulation.
Input parameters	none
Output parameters	True on success.

clearActionLogs	
Method	bool clearActionLogs();
Description	Deletes action logs from the cache, after positive authorisation of the user in the connected environment
Input parameters	None
Output parameters	True on success.
getLastActionLog	
Method	CActionLog *getLastActionLog(string axoid, string objectversion, string protectionstamp);

Description	Returns the last action log.
Input parameters	axoid AXMEDIS identification of the object objectversion Version of the object protectionstamp Protection of the object
Output parameters	The last ActionLog

21 AXCS Proxy

The functionality of this module has been integrated inside PMS client. See PMS client specification section for details.

22 Automatic Generation of Contracts and Licenses (UPC)

Module/Tool Profile		
Automatic Generation of Contracts and Licenses		
Responsible Name	Victor Rodríguez	
Responsible Partner	UPC	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented but not complete	
Status of the implementation	Started	
Executable or Library/module (Support)	Executable	
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/contractgen	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/repos/contractgen/bin/win32	
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	Generation of complete license from contracts	
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
Perl	Perl 5.6	GNU
PDFLib (Lite)	PDFLibLite 6.0	Free (if using only the Lite package). Superior capabilities with the commercial version.
Xerces	Xerces 2.6.0	Apache Software License 2.0
wxWindows	wxWindows 2.4.2	

22.1 General Description of the Module

There is an evident relationship between traditional contracts and digital licenses. By **contract** we understand a binding agreement between two or more parties that is enforceable by law. A particular kind of contract is a **license**, where one of the parties gives the other the authorization to do something.
When these licenses give permission to perform operations over digital items, it seems reasonable that the license is digital itself, and if it is expressed in terms of computer understandable language (ODRL, MPEG21 REL) then we speak about **digital license**.

While contracts and digital licenses satisfy different demands, and therefore is accepted that both will survive (digital licenses will not replace contracts), this module aims at making easier the task of their conversion. This specification states that digital license format must be MPEG 21 REL.

The relationship between PARs (possible available rights) and licenses templates is also considered in this module. A set of different (and frequent) PARs will be kept in the tool, so that it will make easier the transformations

The functionalities to be satisfied in this module are described in the next subsections.

22.1.1 Digital license generation from contracts

This functionality can be useful in a case where a contract already exists, and it is requested to be expressed as a digital license. It is assumed the following:

- **There is a version of the contract as a digital text** (i.e. old paper contracts should previously be scanned, and be subject to an OCR process). In its first version, this module shall accept TXT formats.
- **It is a human assisted process.** The state of the art in natural language processing does not grant a full success in the operations by which a computer extracts information from a human language. However, a contract, with legal consequences, is quite a sensitive document; where a slight variation in the text can imply very different liabilities for the parts. The user will create the licence, and the computer will only bring suggestions and help based on the intelligent text processing the computer makes.

22.1.2 Contract generation from a digital license

In this case, it is an existing digital license that is wanted to be represented in a human understandable way. Such a text may not be legally valid, and yet, may be useful for checking that a digital license expresses some contract clauses.

- The operation is done automatically. In this case, and no human supervising is expected.
- Output of the contract will be written in text format and a basic PDF format.
- Structure of the text will be one among the several contract templates already existing (and to be more precisely defined in a further document).

22.1.3 Process of license generation

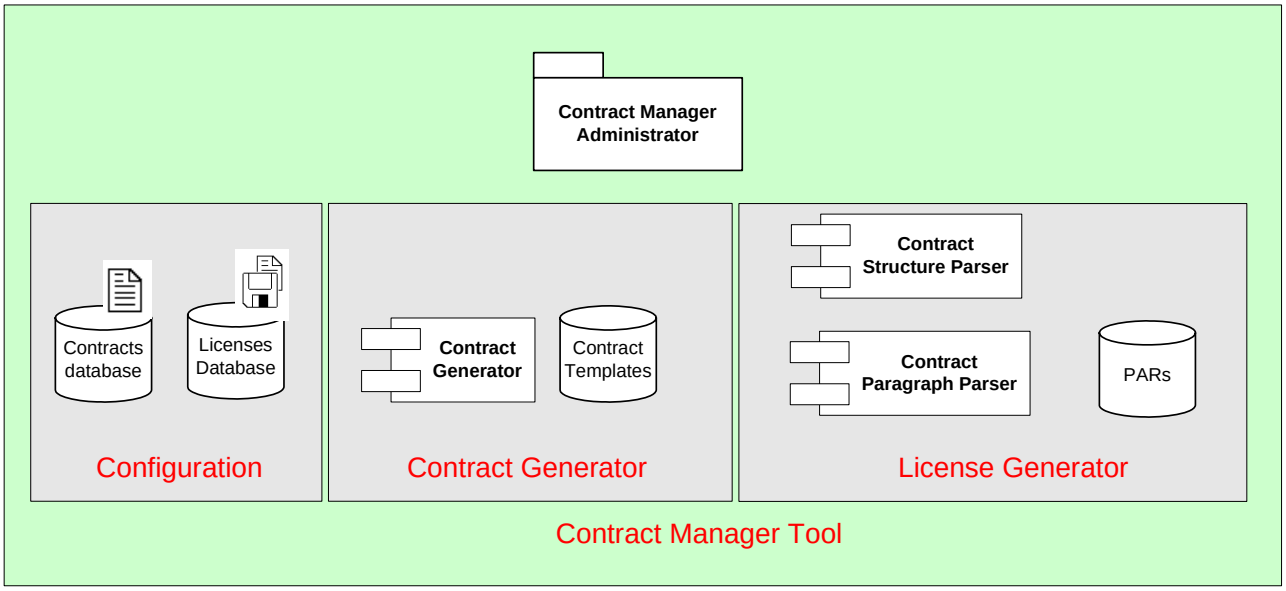
The process of extracting a license from a contract is as follows:

A contract is considered to be the input. From it, is extracted a certain structure, based on the contract clauses identified. Paragraphs are classified if possible, and are assigned a kind of known clause. (i.e. “territory clauses”, “jurisdiction clauses etc.”. Internally, a XML file with this information will be stored. This XML file will not be a REL license, but will be a pre-parsed contract.

Then the user will be asked to introduce the most common license terms. At the time of introducing the data, the user will be helped by the computer: the text of the identified clause in the contract will be shown. If it is possible, the computer will even suggest the full REL element to be inserted, although state of the art in natural language processing only grants a very low success rate.

22.2 Module Design in terms of Classes

Given that the architecture of the application is not trivial, the following diagram shows the compontent structure of the application, instead of the class diagram.



A single executable tool coordinates all the operations related to contracts.
The two main functionalities (contract generation and license generation) represent the two big constituent blocks. An additional block, *configuration*, permits configuration of the tool, (directories for the contracts, and licenses, and models etc.). The blocks called "contracts database" or licenses data

- **Contract Generator.** This is the submodule that generates a contract from a given license.
- **License Generator.** With the assistance of the user, it gives a tentative license from a contract. It is structured in two submodules (structure parser and paragraph parser). It additionally considers the set of PARs that were previously defined.
    - The structure parser determines (with the supervision of the user) the structure of the contract, and chooses one of the license models. Initially the number and form of the license models is predefined, and cannot be extended. Further versions of the tool, could consider the possibility of allowing user to change the models.
    - The contract paragraph parser, extracts from paragraph those elements that can be recognised. Those element s that cannot be recognised, will be able to be added manually by the user.
- **Configuration.** Should carry out some configuration tasks (definition of directories, format of the input / output) etc.

22.3 Integration and Compilation Issues

Local Environment variables to be defined

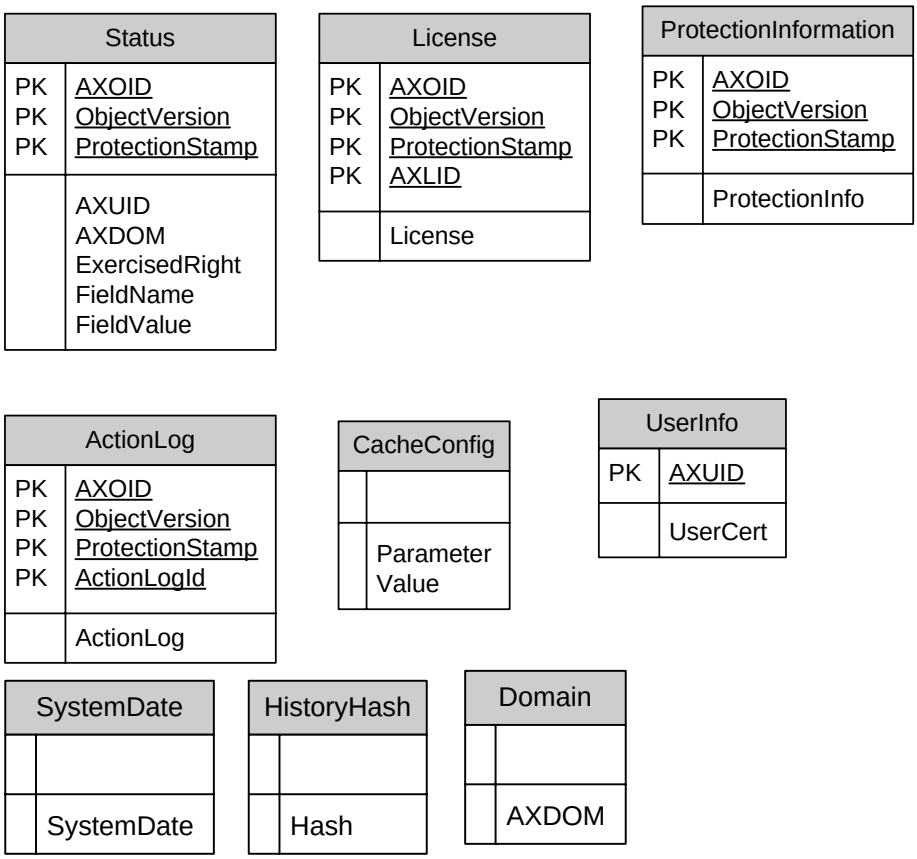
OPENSSL -> Path to OpenSSL library
WXWIN -> Path to WxWidgets
XERCESROOT -> Path to Xerces Library

22.4 User interface description

Still pending to be refined. It might adopt the structure of a wizard application. Ease of use will be considered, attending to the fact that the user is non-expert.

23 Table description for Secure Cache

It should be stressed once more, that although a traditional relational database structure is shown, in practice information is stored in a single file. Anyway, it is a database. No foreign keys have been included to bind the tables, according to the general policy of other Axmedis databases.



License

Columns	Data type	Allow NULLs	Value/Range
AXOID	C-Large Length	Not allowed	
ObjectVersion	C-Large Length	Not allowed	
ProtectionStamp	C-Large Length	Not allowed	
AXLID	C-Large Length	Not allowed	
License	C-Large Length	Not allowed	

Column details

1. AXOID
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Pertinent Object ID.

2. ObjectVersion
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Pertinent Object version.

3. ProtectionStamp
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Indicates the way to protect the related object.

4. AXLID
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Identifier of the stored license.

5. License
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: The whole license.

Protection Information

Columns		idx	Data type	Allow NULLs	Value/Range
AXOID	**PK**		C-Large Length	Not allowed	
ObjectVersion	**PK**		C-Large Length	Not allowed	
ProtectionStamp	**PK**		C-Large Length	Not allowed	
ProtectionInfo			C-Large Length	Not allowed	

1. AXOID
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Pertinent Object ID.

2. ObjectVersion
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Pertinent Object version.

3. ProtectionStamp
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Indicates the way to protect the related object.

4. ProtectionInformation
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Protection Information associated to the object.

ActionLog

Columns	Data type	Allow NULLs	Value/Range
AXOID	C-Large Length	Not allowed	
ObjectVersion	C-Large Length	Not allowed	
ProtectionStamp	C-Large Length	Not allowed	
ActionLog	C-Large Length	Not allowed	

Column details

1. AXOID
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Pertinent Object ID.

2. ObjectVersion
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Pertinent Object version.

3. ProtectionStamp
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Indicates the way to protect the related object.

4. ActionLog

Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Information of an Action log, ciphered in a unique field.

Status

Columns	Data type	Allow NULLs	Value/Range
AXOID	C-Large Length	Not allowed	
ObjectVersion	C-Large Length	Not allowed	
ProtectionStamp	C-Large Length	Not allowed	
Right	C-Large Length	Not allowed	
FieldName	C-Large Length	Not allowed	
FieldValue	C-Large Length	Not allowed	

Column details

1. AXOID
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Pertinent Object ID.

2. ObjectVersion
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Pertinent Object version.

3. ProtectionStamp
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Indicates the way to protect the related object.

4. Right
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Right exercised over the object.

5. FieldName
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Status information name.

6. FieldValue
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Status information name.

CacheConfig

Columns	Data type	Allow NULLs	Value/Range
Parameter	C-Large Length	Not allowed	N/a
Value	C-Large Length	Not allowed	N/a

Column details

1. Parameter
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Stores the name of a parameter value is contained in the same row. For example: “SystemDate”,”Domain” or “Historyhash”

2. Value
Physical data type: LONGTEXT
Allow NULLs: Not allowed

Notes: Undefined meaning, it holds the value for the property given in the “parameter” column of the same row.

SystemDate

Columns	Data type	Allow NULLs	Value/Range
SystemDate	C-Large Length	Not allowed	

Column details

1. SystemDate
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Pertinent Object ID.

HistoryHash

Columns	Data type	Allow NULLs	Value/Range
Hash	C-Large Length	Not allowed	

Column details

1. Hash
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Pertinent Object ID.

Domain

Columns	Data type	Allow NULLs	Value/Range
AXDOM	C-Large Length	Not allowed	

Column details

1. AXDOM
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: Pertinent Object ID.

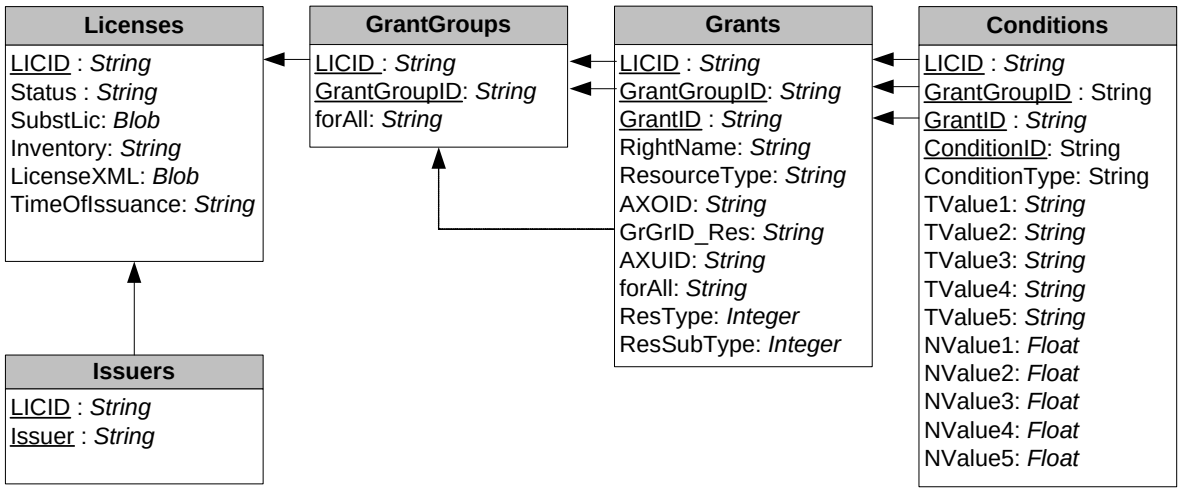
24 Table description for License Database

Complete specification on PAR and license database on DE3-1-2-2-9, Database and Gathering.

24.1 ER diagram for Licenses

To represent the content of a license in an Entity-Relationship diagram, we have to focus on the relations with a multiplicity 0..n. These relations show us the number of different tables that we need to store the represented information. The relations with a multiplicity of 1 – 1 can be stored always in the same table.

The next diagram shows how to create the different tables to store the license information. This solution provides the model for storing End-user Licenses, and also for storing Distributor Licenses.



ER diagram for licenses

Licenses

Columns		idx	Data type	Allow NULLs	Value/Range
AXLID	**PK**	**I**	C-Large Length	Not allowed	
Status			C-Large Length	Not allowed	
SubsLic			C-Large Length	Allowed	
Inventory			C-Large Length	Allowed	
TimeofIssuance			C-Large Length	Not allowed	
LicenseXML			C-Blob	Not allowed	

Column details

1. AXLID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String representing the unique identifier for the license

2. Status
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String that contains the status of the license, possible values are valid or revoked

3. SubsLic
Physical data type: LONGTEXT
Allow NULLs: Allowed
Notes: String that contains the MPEG-21 REL license that replaces the revoked one, if any

4. Inventory
Physical data type: LONGTEXT

Allow NULLs: Allowed
Notes: String that contains the variables defined in the license, that can be referenced throught this license

5. TimeofIssuance
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String that represents the specific date and time at which the license has been issued

6. LicenseXML
Physical data type: BLOB
Allow NULLs: Not allowed
Notes: contains the XML MPEG-21 REL license

Issuers

Number of indexes: ?
Number of foreign keys: ?

Columns		idx	Data type	Allow NULLs	Value/Range
AXLID	**PK**	**I**	C-Large Length	Not allowed	
AXUID	**PK**	**I**	C-Large Length	Not allowed	

Column details

1. AXLID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String representing the unique identifier for the license

2. AXUID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String thar represents the unique identifier of the AXMEDIS user that has issued the license

GrantGroups

Number of indexes: ?
Number of foreign keys: ?

Columns		idx	Data type	Allow NULLs	Value/Range
AXLID	**PK**	**I**	C-Large Length	Not allowed	
GrantGroupID	**PK**	**I**	C-Large Length	Not allowed	
forAll			C-Large Length	Allowed	

Column details

1. AXLID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String representing the unique identifier for the license

2. GrantGroupID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String containing the unique identifier of the grantGroup

3. forAll
Physical data type: LONGTEXT
Allow NULLs: Allowed
Notes: String that contains variables whose scope is this entire grantGroup uniquely identified by the GrantGroupID

Grants

Number of indexes: ?
Number of foreign keys: ?

Columns		idx	Data type	Allow NULLs	Value/Range
AXLID	**PK**	**I**	C-Large Length	Not allowed	
GrantGroupID	**PK**	**I**	C-Large Length	Not allowed	
GrantID	**PK**	**I**	C-Large Length	Not allowed	
Right			C-Large Length	Not allowed	
ResourceType			C-Large Length	Not allowed	
AXOID		**I**	C-Large Length	Allowed	
GrGrID_Res	**FK**	**I**	C-Large Length	Allowed	
AXUID			C-Large Length	Not allowed	
forAll			C-Large Length	Allowed	
ResType			Integer	Not allowed	
ResSubType			Integer	Not allowed	

Column details

1. AXLID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String representing the unique identifier for the license

2. GrantGroupID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String containing the unique identifier of the grantGroup

3. GrantID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String containing the unique identifier of the grant

4. Right
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String that specifies the right granted

5. ResourceType
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String that specifies the type of object against which the principal of this grant has the right to perform an action. If the resouceType is Resource, then the object against which the AXMEDIS user can exercise the right is an AXMEDIS object, and if the resourceType is GrantGroup then the object is a grant or granGroup, typically for distribution licenses

6. AXOID
Physical data type: LONGTEXT
Allow NULLs: Allowed
Notes: String containing ths unique identifier of the AXMEDIS object

7. GrGrID_Res (FK)
Physical data type: LONGTEXT
Allow NULLs: Allowed
Notes: String containing ths unique identifier of the grantGroup that can be issued

8. AXUID
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String identifying the AXMEDIS user to whom this grant conveys rights

9. forAll

Physical data type: LONGTEXT
Allow NULLs: Allowed
Notes: String that contains variables whose scope is the entire grant uniquely identified by the GrantID

10. ResType
Physical data type: INTEGER
Allow NULLs: Not Allowed
Notes: If ResourceType is "Resource" this field sets the type of the "reference" to the resource found in AXOID. 0 → Digital Item Item, 1→ Digital Item Reference

11. ResSubType
Physical data type: INTEGER
Allow NULLs: Not Allowed
Notes: If ResType is 0 (Digital Item Item) this field sets the type of the reference. 0(id), 1(uri), 2(type)

Conditions

Number of indexes: ?
Number of foreign keys: ?

Columns		idx	Data type	Allow NULLs	Value/Range
AXLID	**PK**	**I**	C-Large Length	Not allowed	
GrantGroupID	**PK**	**I**	C-Large Length	Not allowed	
GrantID	**PK**	**I**	C-Large Length	Not allowed	
ConditionID	**PK**	**I**	C-Large Length	Not allowed	
ConditionType		**I**	C-Large Length	Not allowed	
TValue1			C-Large Length	Allowed	
TValue2			C-Large Length	Allowed	
TValue3			C-Large Length	Allowed	
TValue4			C-Large Length	Allowed	
TValue5			C-Large Length	Allowed	
NValue1			C-Float	Allowed	
NValue2			C-Float	Allowed	
NValue3			C-Float	Allowed	
NValue4			C-Float	Allowed	
NValue5			C-Float	Allowed	

Column details

1. AXLID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String representing the unique identifier for the license

2. GrantGroupID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String containing the unique identifier of the grantGroup

3. GrantID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String containing the unique identifier of the grant

4. ConditionID
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String containing the unique identifier of the condition

5. ConditionType
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String representing the type of the condition. This field can have the values specificied in the ConditionType column of the Condition Table. For example, this field can take the values territory or validityInterval

6. TValue(1-5)
Physical data type: LONGTEXT
Allow NULLs: Allowed
Notes: String that contains information related to the condition according to the ConditionType as defined in the Condition Table. For example, if the ConditionType is validityInterval, the TValue1 contains a String that represents the date at which the interval of time defined by this condition begins and the Tvalue2 contains a String that represents the date at which the interval of time defined by this condition ends

7. Nvalue(1-5)
Physical data type: FLOAT
Allow NULLs: Allowed
Notes: Numeric value that contains information related to the condition according to the ConditionType as defined in the Condition Table. For example, if the ConditionType is exerciseLimit, the NValue1 represents the limit on the number of times that certain exercises may occur

The relation between Tables and Classes is:

Table (ER)	Classes (UML) stored in the table
Licenses	License
Issuers	Issuer
GrantGroups	GrantGroup
Grants	Grant, Right, Resource, Principal
Conditions	Condition (all types)

To represent all type of conditions, we have decided to store the data in one unique table with a set of "standard" fields. Each field of this table corresponds to an attribute of the condition depending on the condition type.

We provide a table where we describe the mapping between the standard fields of the table (ER) and the condition attributes (UML).

In this model is very easy to add new types of conditions to the system without causing the reimplementation of a lot of modules. And, moreover, it makes easier and much more efficient the search of the information needed in the authorisation model.

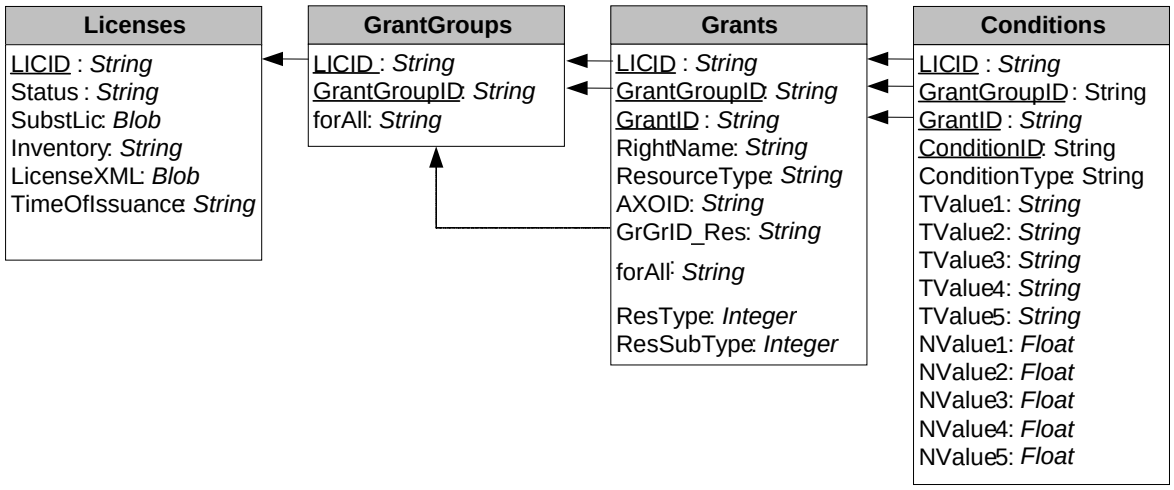
25 Table description for PAR Database

Complete specification on PAR and license database on DE3-1-2-2-9, Database and Gathering.

25.1 ER diagram for PARs

To represent the content of a license in an Entity-Relationship diagram, we have to focus on the relations with a multiplicity 0..n. These relations show us the number of different tables that we need to store the represented information. The relations with a multiplicity of 1 – 1 can be stored always in the same table.

The next diagram shows how to create the different tables to store the license information. This solution provides the model for storing End-user Licenses, and also for storing Distributor Licenses.



ER diagram for PARs

PARs

Columns		idx	Data type	Allow NULLs	Value/Range
AXLID	**PK**	**I**	C-Large Length	Not allowed	
Status			C-Large Length	Not allowed	
SubsLic			C-Large Length	Allowed	
Inventory			C-Large Length	Allowed	
TimeofIssuance			C-Large Length	Not allowed	
LicenseXML			C-Blob	Not allowed	

Column details

1. AXLID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String representing the unique identifier for the license

2. Status
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String that contains the status of the license, possible values are valid or revoked

3. SubsLic
Physical data type: LONGTEXT
Allow NULLs: Allowed
Notes: String that contains the MPEG-21 REL license that replaces the revoked one, if any

4. Inventory
Physical data type: LONGTEXT
Allow NULLs: Allowed

Notes: String that contains the variables defined in the license, that can be referenced throught this license

5. TimeofIssuance
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String that represents the specific date and time at which the license has been issued

6. LicenseXML
Physical data type: BLOB
Allow NULLs: Not allowed
Notes: contains the XML MPEG-21 REL license

GrantGroups

Number of indexes: ?
Number of foreign keys: ?

Columns		idx	Data type	Allow NULLs	Value/Range
AXLID	**PK**	**I**	C-Large Length	Not allowed	
GrantGroupID	**PK**	**I**	C-Large Length	Not allowed	
forAll			C-Large Length	Allowed	

Column details

1. AXLID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String representing the unique identifier for the license

2. GrantGroupID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String containing the unique identifier of the grantGroup

3. forAll
Physical data type: LONGTEXT
Allow NULLs: Allowed
Notes: String that contains variables whose scope is this entire grantGroup uniquely identified by the GrantGroupID

Grants

Number of indexes: ?
Number of foreign keys: ?

Columns		idx	Data type	Allow NULLs	Value/Range
AXLID	**PK**	**I**	C-Large Length	Not allowed	
GrantGroupID	**PK**	**I**	C-Large Length	Not allowed	
GrantID	**PK**	**I**	C-Large Length	Not allowed	
Right			C-Large Length	Not allowed	
ResourceType			C-Large Length	Not allowed	
AXOID		**I**	C-Large Length	Allowed	
GrGrID_Res	**FK**	**I**	C-Large Length	Allowed	
AXUID			C-Large Length	Not allowed	
forAll			C-Large Length	Allowed	
ResType			Integer	Not allowed	
ResSubType			Integer	Not allowed	

Column details

1. AXLID (PK)
Physical data type: LONGTEXT

Allow NULLs: Not allowed
Notes: String representing the unique identifier for the license

2. GrantGroupID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String containing the unique identifier of the grantGroup

3. GrantID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String containing the unique identifier of the grant

4. Right
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String that specifies the right granted

5. ResourceType
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String that specifies the type of object against which the principal of this grant has the right to perform an action. If the resouceType is Resource, then the object against which the AXMEDIS user can exercise the right is an AXMEDIS object, and if the resourceType is GrantGroup then the object is a grant or granGroup, typically for distribution licenses

6. AXOID
Physical data type: LONGTEXT
Allow NULLs: Allowed
Notes: String containing ths unique identifier of the AXMEDIS object

7. GrGrID_Res (FK)
Physical data type: LONGTEXT
Allow NULLs: Allowed
Notes: String containing ths unique identifier of the grantGroup that can be issued

8. AXUID
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String identifying the AXMEDIS user to whom this grant conveys rights

9. forAll
Physical data type: LONGTEXT
Allow NULLs: Allowed
Notes: String that contains variables whose scope is the entire grant uniquely identified by the GrantID

10. ResType
Physical data type: INTEGER
Allow NULLs: Not Allowed
Notes: If ResourceType is “Resource” this field sets the type of the “reference” to the resource found in AXOID. 0 → Digital Item Item, 1→ Digital Item Reference

11. ResSubType
Physical data type: INTEGER
Allow NULLs: Not Allowed
Notes: If ResType is 0 (Digital Item Item) this field sets the type of the reference. 0(id), 1(uri), 2(type)

Conditions

Number of indexes: ?
Number of foreign keys: ?

Columns	idx	Data type	Allow NULLs	Value/Range

AXLID	**PK**	**I**	C-Large Length	Not allowed
GrantGroupID	**PK**	**I**	C-Large Length	Not allowed
GrantID	**PK**	**I**	C-Large Length	Not allowed
ConditionID	**PK**	**I**	C-Large Length	Not allowed
ConditionType		**I**	C-Large Length	Not allowed
TValue1			C-Large Length	Allowed
TValue2			C-Large Length	Allowed
TValue3			C-Large Length	Allowed
TValue4			C-Large Length	Allowed
TValue5			C-Large Length	Allowed
NValue1			C-Float	Allowed
NValue2			C-Float	Allowed
NValue3			C-Float	Allowed
NValue4			C-Float	Allowed
NValue5			C-Float	Allowed

Column details

1. AXLID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String representing the unique identifier for the license

2. GrantGroupID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String containing the unique identifier of the grantGroup

3. GrantID (PK)
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String containing the unique identifier of the grant

4. ConditionID
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String containing the unique identifier of the condition

5. ConditionType
Physical data type: LONGTEXT
Allow NULLs: Not allowed
Notes: String representing the type of the condition. This field can have the values specificied in the ConditionType column of the Condition Table. For example, this field can take the values territory or validityInterval

6. TValue(1-5)
Physical data type: LONGTEXT
Allow NULLs: Allowed
Notes: String that contains information related to the condition according to the ConditionType as defined in the Condition Table. For example, if the ConditionType is validityInterval, the TValue1 contains a String that represents the date at which the interval of time defined by this condition begins and the Tvalue2 contains a String that represents the date at which the interval of time defined by this condition ends

7. Nvalue(1-5)
Physical data type: FLOAT
Allow NULLs: Allowed
Notes: Numeric value that contains information related to the condition according to the ConditionType as defined in the Condition Table. For example, if the ConditionType is exerciseLimit, the NValue1 represents the limit on the number of times that certain exercises may occur

The relation between Tables and Classes is:

Table (ER)	Classes (UML) stored in the table

Licenses	License
Issuers	Issuer
GrantGroups	GrantGroup
Grants	Grant, Right, Resource, Principal
Conditions	Condition (all types)

To represent all type of conditions, we have decided to store the data in one unique table with a set of "standard" fields. Each field of this table corresponds to an attribute of the condition depending on the condition type.

We provide a table where we describe the mapping between the standard fields of the table (ER) and the condition attributes (UML).

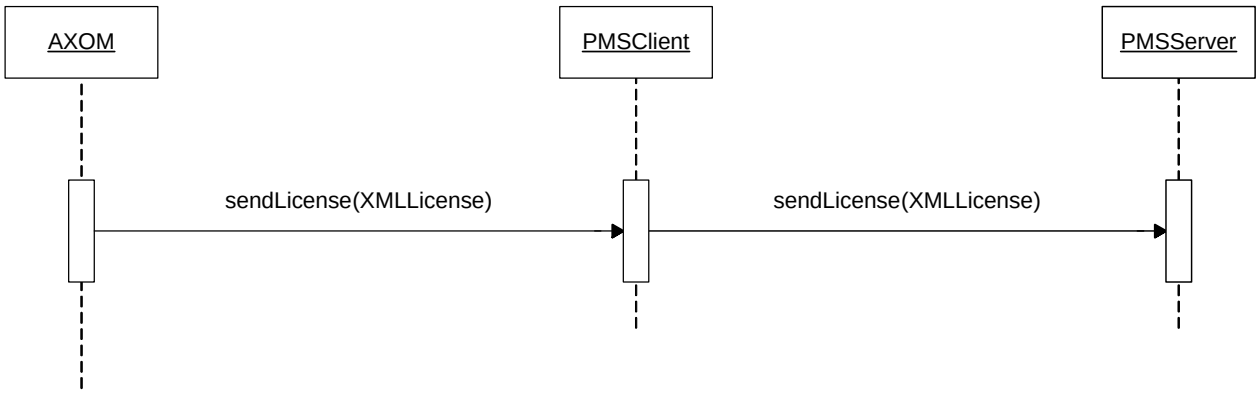
In this model is very easy to add new types of conditions to the system without causing the reimplementation of a lot of modules. And, moreover, it makes easier and much more efficient the search of the information needed in the authorisation model.

26 Formal description of License Format (MPEG-21 REL)

Current license format is based on Part 5 of MPEG-21 standard, MPEG-21 Rights Expression Language [1]. The serialisation of MPEG-21 REL licenses is done using XML language, but we have defined a relational structure for licenses in order to speed up operations done using licenses (authorisation of actions, search of distribution licenses and PAR, etc.).

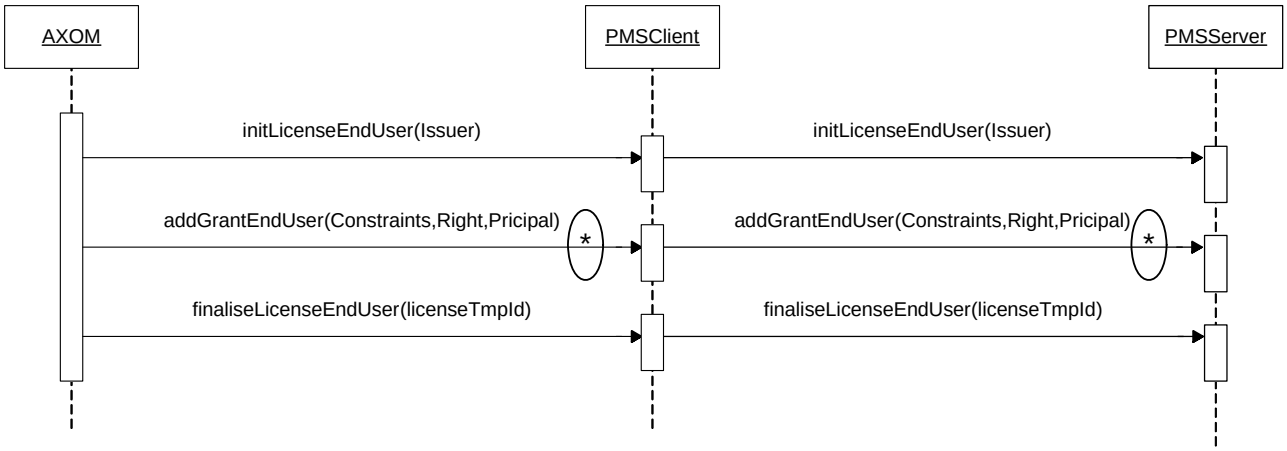
Nevertheless, MPEG-21 REL will not be only language supported. OMA DRM REL [2] and MPEG-21 REL Base profiles [3] are also being considered. These languages can be serialised using XML and a relational model will be defined for them (in the case of the MPEG-21 REL base profiles, the format will be common to the MPEG-21 REL, only new conditions are added). The definition of these modules will facilitate translation and adaptation of licenses to accomplish the requirements of the different business models and scenarios.

27 Formal description of Posting License on PMS

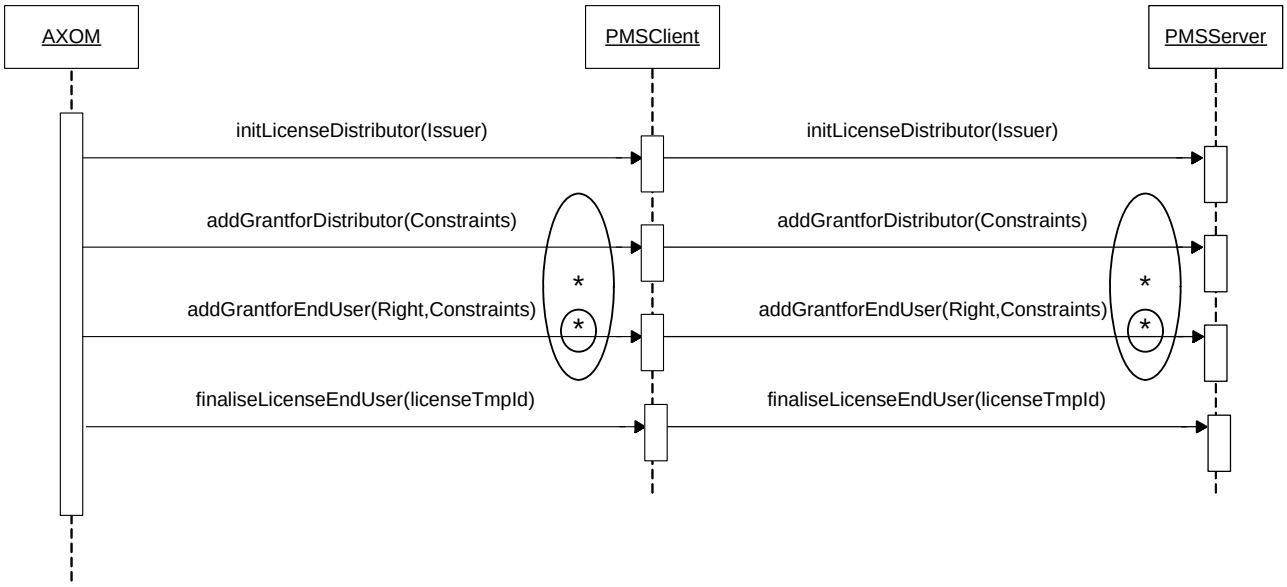


28 Formal description of License Creation

License Creation por End User Licenses.

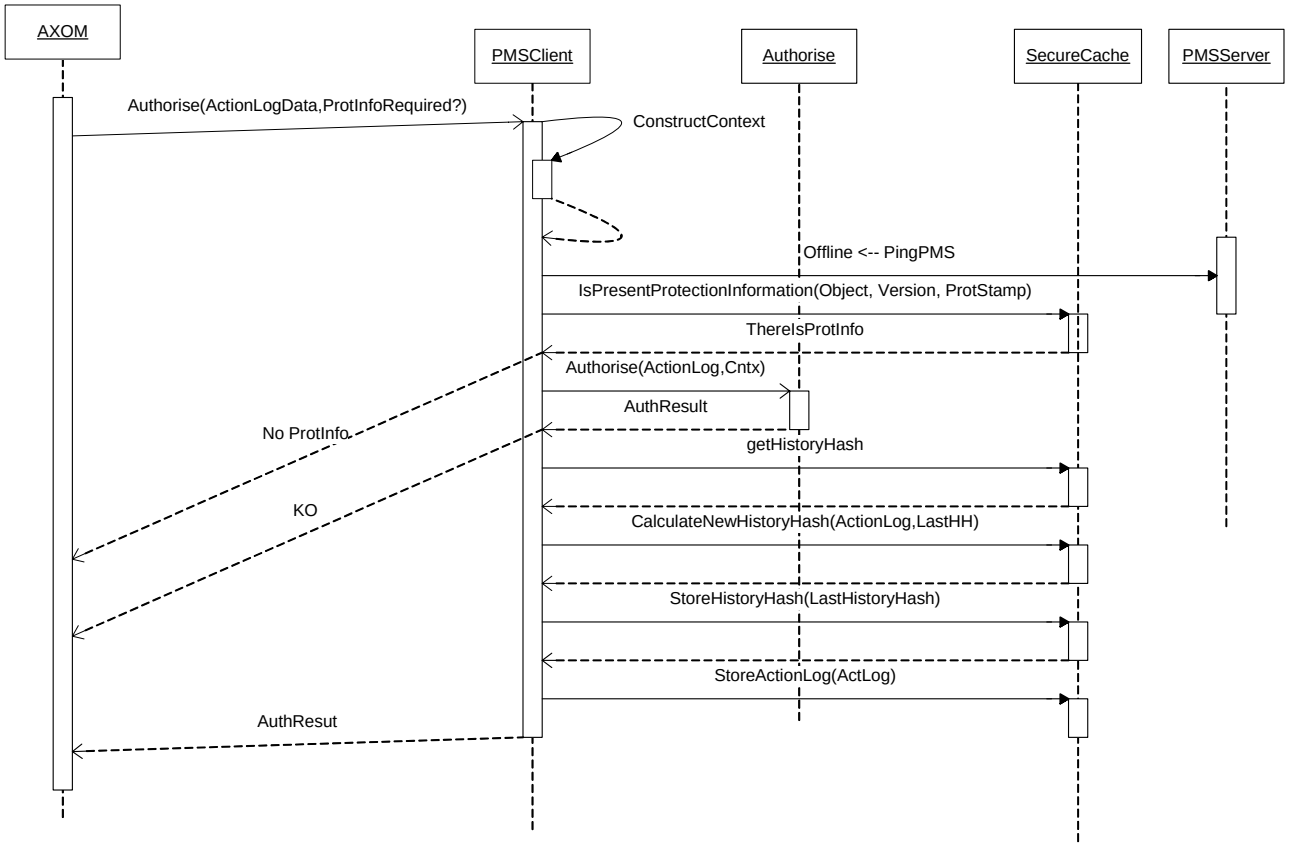


Licenses Creation for Distributor Licenses

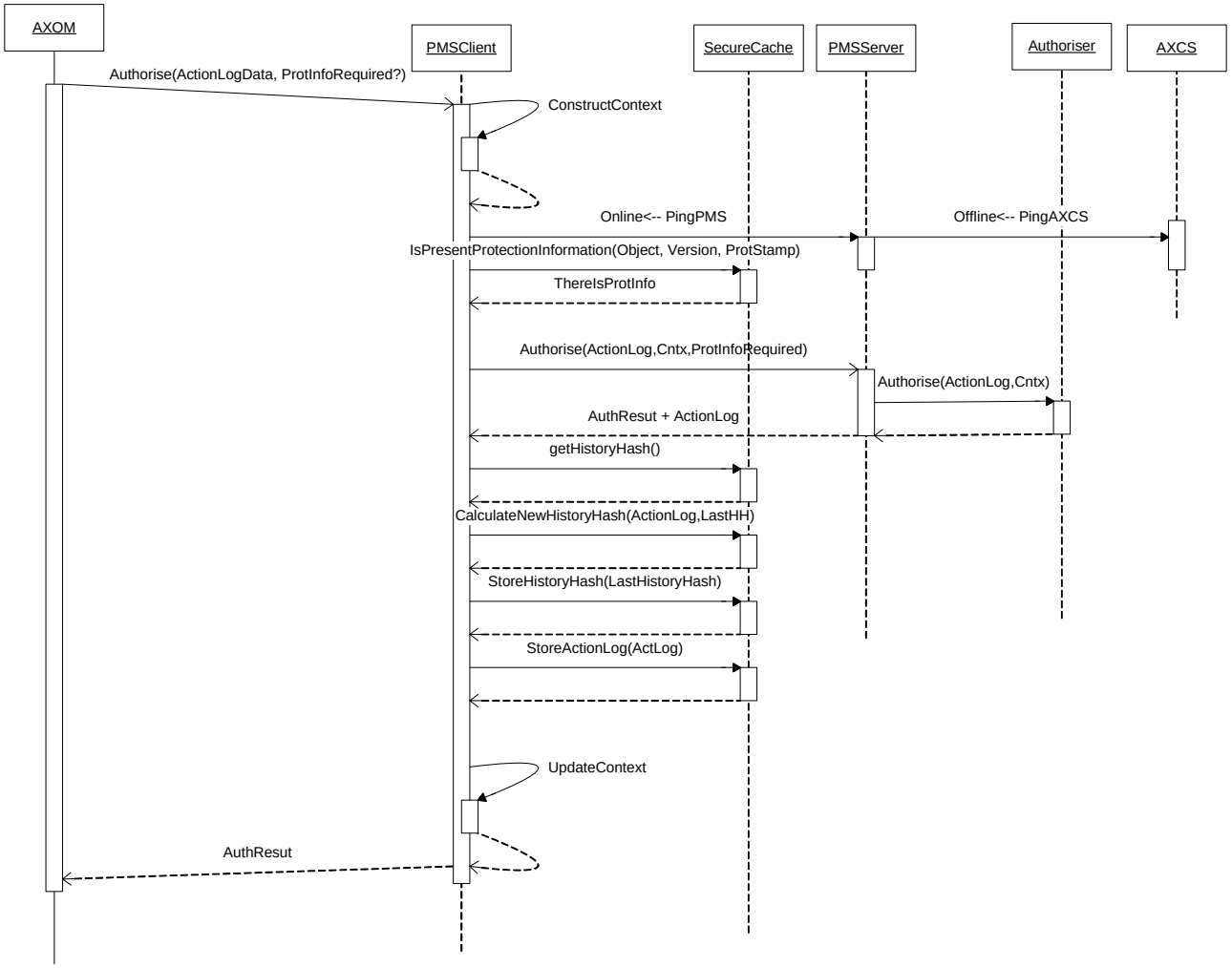


29 Formal description of Authorisation

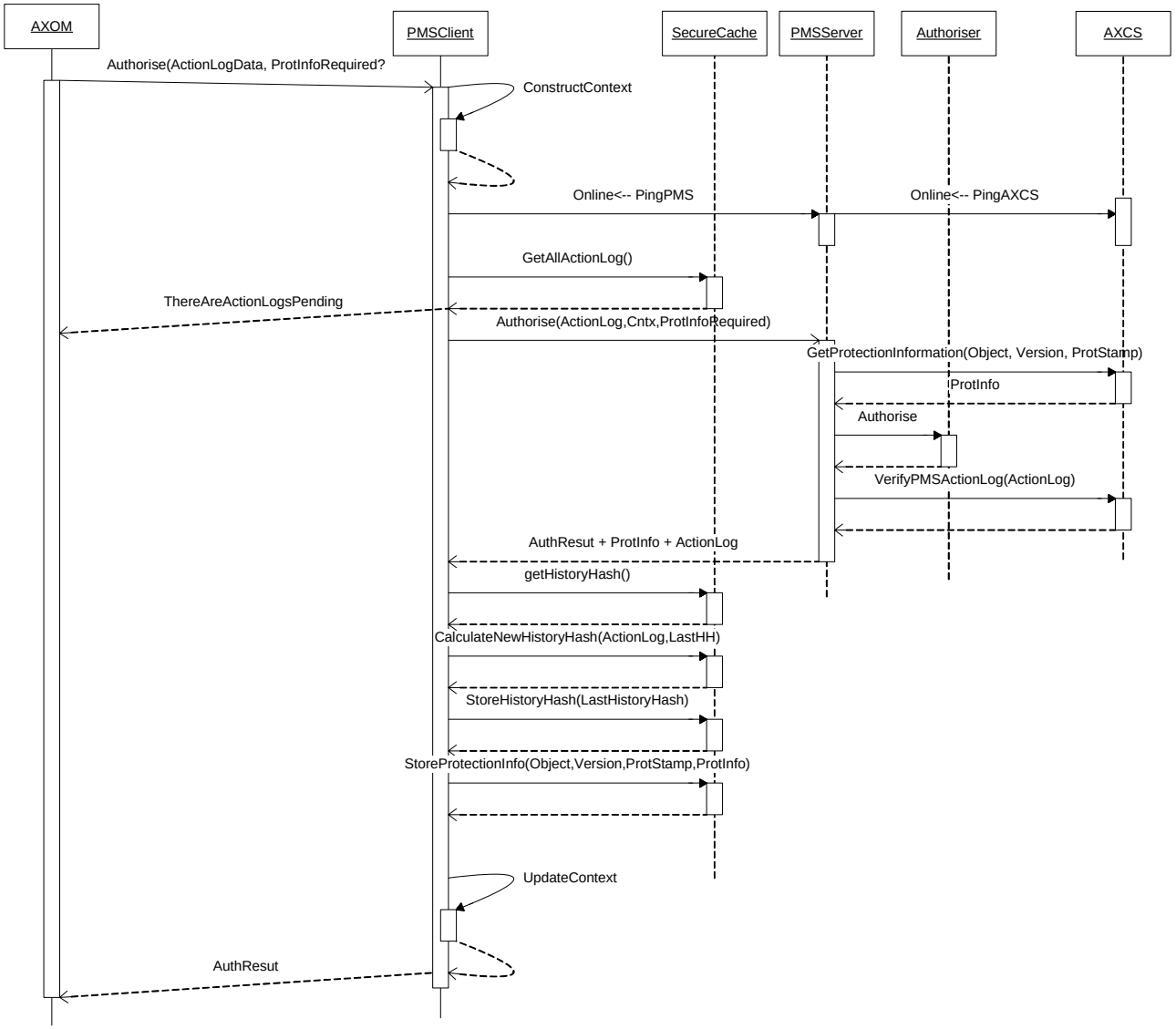
Authorisation diagram when PMS Server is Offline:



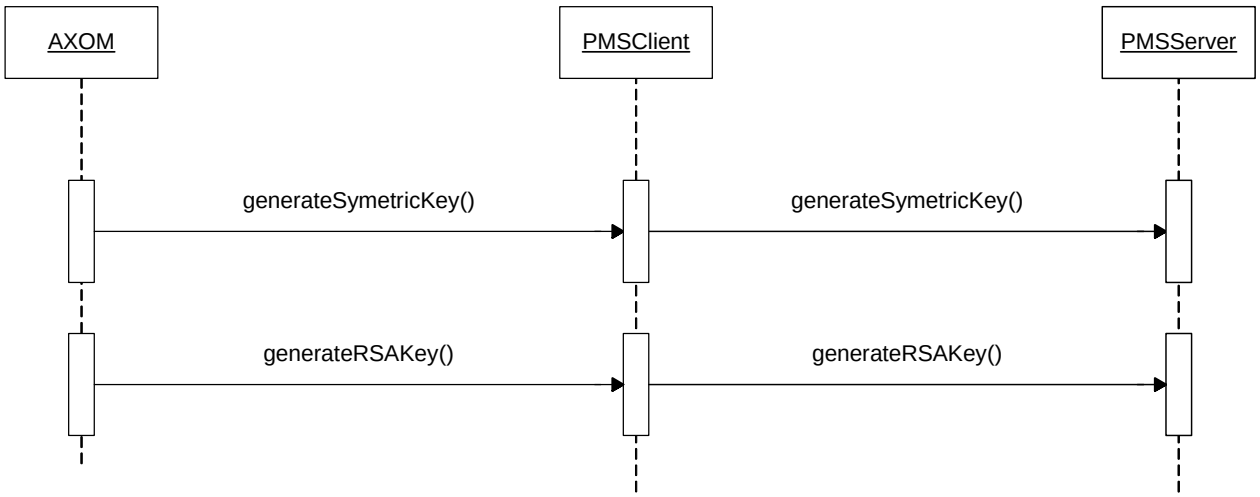
Authorisation diagram when PMS Server is Online but AXCS is Offline:



Authorisation diagram when PMS Server is Online and AXCS is Online:


AXOM
PMSClient
SecureCache
PMSServer
Authoriser
AXCS
Authorise(ActionLogData, ProtInfoRequired?
ConstructContext
Online<-- PingPMS
Online<-- PingAXCS
GetAllActionLog()
ThereAreActionLogsPending
Authorise(ActionLog,Cntx,ProtInfoRequired)
GetProtectionInformation(Object, Version, ProtStamp)
ProtInfo
Authorise
VerifyPMSActionLog(ActionLog)
AuthResut + ProtInfo + ActionLog
getHistoryHash()
CalculateNewHistoryHash(ActionLog,LastHH)
StoreHistoryHash(LastHistoryHash)
StoreProtectionInfo(Object,Version,ProtStamp,ProtInfo)
UpdateContext
AuthResut

30 Formal description of Key Generation



31 WSDL for Protection Manager Support

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="urn:PMS" xmlns:intf="urn:PMS"
xmlns:tns1="http://AXCV" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:PMS">
        <wsdl:types>
                <schema elementFormDefault="qualified" targetNamespace="urn:PMS"
xmlns="http://www.w3.org/2001/XMLSchema">
                        <import namespace="http://AXCV"/>
                        <element name="initLicenseEndUser">
                                <complexType>
                                        <sequence>
                                                <element name="certIssuer" type="xsd:string"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="initLicenseEndUserResponse">
                                <complexType>
                                        <sequence>
                                                <element name="initLicenseEndUserReturn" type="xsd:string"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="addGrantEndUser">
                                <complexType>
                                        <sequence>
                                                <element name="licenseTmpId" type="xsd:string"/>
                                                <element name="certPrincipal" type="xsd:string"/>
                                                <element name="diID" type="xsd:string"/>
                                                <element name="diType" type="xsd:int"/>
                                                <element name="diSubType" type="xsd:int"/>
                                                <element name="right" type="xsd:string"/>
                                                <element name="validityInterval" type="xsd:boolean"/>
                                                <element name="notBefore" type="xsd:string"/>
                                                <element name="notAfter" type="xsd:string"/>
                                                <element name="countLimit" type="xsd:boolean"/>
                                                <element name="limit" type="xsd:int"/>
                                                <element name="validityRegion" type="xsd:boolean"/>
                                                <element name="country" type="xsd:string"/>
                                                <element name="region" type="xsd:string"/>
                                                <element name="feeType" type="xsd:int"/>
                                                <element name="fee" type="xsd:float"/>
                                                <element name="currency" type="xsd:string"/>
                                                <element name="bankAccount" type="xsd:string"/>
                                                <element name="adaptationRules" type="xsd:string"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="addGrantEndUserResponse">
                                <complexType>
                                        <sequence>
                                                <element name="addGrantEndUserReturn" type="xsd:string"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="finaliseLicenseEndUser">
                                <complexType>
                                        <sequence>
                                                <element name="licenseTmpId" type="xsd:string"/>
                                                <element name="context" type="tns1:contextData"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="finaliseLicenseEndUserResponse">
                                <complexType>
                                        <sequence>
                                                <element name="finaliseLicenseEndUserReturn" type="xsd:string"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="initLicenseDistributor">
                                <complexType>
                                        <sequence>
```

```
                                        <element name="certIssuer" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="initLicenseDistributorResponse">
                        <complexType>
                                <sequence>
                                        <element name="initLicenseDistributorReturn" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="addGrantforDistributor">
                        <complexType>
                                <sequence>
                                        <element name="licenseTmpId" type="xsd:string"/>
                                        <element name="certPrincipal" type="xsd:string"/>
                                        <element name="diID" type="xsd:string"/>
                                        <element name="diType" type="xsd:int"/>
                                        <element name="diSubType" type="xsd:int"/>
                                        <element name="validityInterval" type="xsd:boolean"/>
                                        <element name="notBefore" type="xsd:string"/>
                                        <element name="notAfter" type="xsd:string"/>
                                        <element name="countLimit" type="xsd:boolean"/>
                                        <element name="limit" type="xsd:int"/>
                                        <element name="validityRegion" type="xsd:boolean"/>
                                        <element name="country" type="xsd:string"/>
                                        <element name="region" type="xsd:string"/>
                                        <element name="feeType" type="xsd:int"/>
                                        <element name="fee" type="xsd:float"/>
                                        <element name="currency" type="xsd:string"/>
                                        <element name="bankAccount" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="addGrantforDistributorResponse">
                        <complexType>
                                <sequence>
                                        <element name="addGrantforDistributorReturn" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="addGrantforEndUser">
                        <complexType>
                                <sequence>
                                        <element name="licenseTmpId" type="xsd:string"/>
                                        <element name="distGrantId" type="xsd:string"/>
                                        <element name="right" type="xsd:string"/>
                                        <element name="validityInterval" type="xsd:boolean"/>
                                        <element name="notBefore" type="xsd:string"/>
                                        <element name="notAfter" type="xsd:string"/>
                                        <element name="countLimit" type="xsd:boolean"/>
                                        <element name="limit" type="xsd:int"/>
                                        <element name="validityRegion" type="xsd:boolean"/>
                                        <element name="country" type="xsd:string"/>
                                        <element name="region" type="xsd:string"/>
                                        <element name="feeType" type="xsd:int"/>
                                        <element name="fee" type="xsd:float"/>
                                        <element name="currency" type="xsd:string"/>
                                        <element name="bankAccount" type="xsd:string"/>
                                        <element name="adaptationRules" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="addGrantforEndUserResponse">
                        <complexType>
                                <sequence>
                                        <element name="addGrantforEndUserReturn" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="finaliseLicenseDistributor">
                        <complexType>
                                <sequence>
                                        <element name="licenseTmpId" type="xsd:string"/>
                                        <element name="context" type="tns1:contextData"/>
```

```
                        </sequence>
                </complexType>
        </element>
        <element name="finaliseLicenseDistributorResponse">
                <complexType>
                        <sequence>
                                <element name="finaliseLicenseDistributorReturn" type="xsd:string"/>
                        </sequence>
                </complexType>
        </element>
        <element name="getLicense">
                <complexType>
                        <sequence>
                                <element name="licenseId" type="xsd:string"/>
                        </sequence>
                </complexType>
        </element>
        <element name="getLicenseResponse">
                <complexType>
                        <sequence>
                                <element name="getLicenseReturn" type="xsd:string"/>
                        </sequence>
                </complexType>
        </element>
        <element name="sendLicense">
                <complexType>
                        <sequence>
                                <element name="licenseXML" type="xsd:string"/>
                                <element name="context" type="tns1:contextData"/>
                        </sequence>
                </complexType>
        </element>
        <element name="sendLicenseResponse">
                <complexType>
                        <sequence>
                                <element name="sendLicenseReturn" type="xsd:string"/>
                        </sequence>
                </complexType>
        </element>
        <element name="initPAREndUser">
                <complexType>
                        <sequence>
                                <element name="x" type="xsd:int"/>
                        </sequence>
                </complexType>
        </element>
        <element name="initPAREndUserResponse">
                <complexType>
                        <sequence>
                                <element name="initPAREndUserReturn" type="xsd:string"/>
                        </sequence>
                </complexType>
        </element>
        <element name="addPARGrantEndUser">
                <complexType>
                        <sequence>
                                <element name="PARTmpId" type="xsd:string"/>
                                <element name="diID" type="xsd:string"/>
                                <element name="diType" type="xsd:int"/>
                                <element name="diSubType" type="xsd:int"/>
                                <element name="right" type="xsd:string"/>
                                <element name="validityInterval" type="xsd:boolean"/>
                                <element name="notBefore" type="xsd:string"/>
                                <element name="notAfter" type="xsd:string"/>
                                <element name="countLimit" type="xsd:boolean"/>
                                <element name="limit" type="xsd:int"/>
                                <element name="validityRegion" type="xsd:boolean"/>
                                <element name="country" type="xsd:string"/>
                                <element name="region" type="xsd:string"/>
                                <element name="feeType" type="xsd:int"/>
                                <element name="fee" type="xsd:float"/>
                                <element name="currency" type="xsd:string"/>
                                <element name="bankAccount" type="xsd:string"/>
                                <element name="adaptationRules" type="xsd:string"/>
                        </sequence>
```

```
                </complexType>
        </element>
        <element name="addPARGrantEndUserResponse">
                <complexType>
                        <sequence>
                                <element name="addPARGrantEndUserReturn" type="xsd:string"/>
                        </sequence>
                </complexType>
        </element>
        <element name="finalisePAREndUser">
                <complexType>
                        <sequence>
                                <element name="PARTmpId" type="xsd:string"/>
                        </sequence>
                </complexType>
        </element>
        <element name="finalisePAREndUserResponse">
                <complexType>
                        <sequence>
                                <element name="finalisePAREndUserReturn" type="xsd:string"/>
                        </sequence>
                </complexType>
        </element>
        <element name="initPARDistributor">
                <complexType>
                        <sequence>
                                <element name="x" type="xsd:int"/>
                        </sequence>
                </complexType>
        </element>
        <element name="initPARDistributorResponse">
                <complexType>
                        <sequence>
                                <element name="initPARDistributorReturn" type="xsd:string"/>
                        </sequence>
                </complexType>
        </element>
        <element name="addPARGrantforDistributor">
                <complexType>
                        <sequence>
                                <element name="PARTmpId" type="xsd:string"/>
                                <element name="diID" type="xsd:string"/>
                                <element name="diType" type="xsd:int"/>
                                <element name="diSubType" type="xsd:int"/>
                                <element name="validityInterval" type="xsd:boolean"/>
                                <element name="notBefore" type="xsd:string"/>
                                <element name="notAfter" type="xsd:string"/>
                                <element name="countLimit" type="xsd:boolean"/>
                                <element name="limit" type="xsd:int"/>
                                <element name="validityRegion" type="xsd:boolean"/>
                                <element name="country" type="xsd:string"/>
                                <element name="region" type="xsd:string"/>
                                <element name="feeType" type="xsd:int"/>
                                <element name="fee" type="xsd:float"/>
                                <element name="currency" type="xsd:string"/>
                                <element name="bankAccount" type="xsd:string"/>
                        </sequence>
                </complexType>
        </element>
        <element name="addPARGrantforDistributorResponse">
                <complexType>
                        <sequence>
                                <element name="addPARGrantforDistributorReturn" type="xsd:string"/>
                        </sequence>
                </complexType>
        </element>
        <element name="addPARGrantforEndUser">
                <complexType>
                        <sequence>
                                <element name="PARTmpId" type="xsd:string"/>
                                <element name="distGrantId" type="xsd:string"/>
                                <element name="right" type="xsd:string"/>
                                <element name="validityInterval" type="xsd:boolean"/>
                                <element name="notBefore" type="xsd:string"/>
                                <element name="notAfter" type="xsd:string"/>
```

```
                                        <element name="countLimit" type="xsd:boolean"/>
                                        <element name="limit" type="xsd:int"/>
                                        <element name="validityRegion" type="xsd:boolean"/>
                                        <element name="country" type="xsd:string"/>
                                        <element name="region" type="xsd:string"/>
                                        <element name="feeType" type="xsd:int"/>
                                        <element name="fee" type="xsd:float"/>
                                        <element name="currency" type="xsd:string"/>
                                        <element name="bankAccount" type="xsd:string"/>
                                        <element name="adaptationRules" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="addPARGrantforEndUserResponse">
                        <complexType>
                                <sequence>
                                        <element name="addPARGrantforEndUserReturn" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="finalisePARDistributor">
                        <complexType>
                                <sequence>
                                        <element name="PARTmpId" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="finalisePARDistributorResponse">
                        <complexType>
                                <sequence>
                                        <element name="finalisePARDistributorReturn" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="getPAR">
                        <complexType>
                                <sequence>
                                        <element name="PARId" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="getPARResponse">
                        <complexType>
                                <sequence>
                                        <element name="getPARReturn" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="sendPAR">
                        <complexType>
                                <sequence>
                                        <element name="PARXML" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="sendPARResponse">
                        <complexType>
                                <sequence>
                                        <element name="sendPARReturn" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="authorise">
                        <complexType>
                                <sequence>
                                        <element name="constructingAL" type="tns1:ActionLog"/>
                                        <element name="context" type="tns1:contextData"/>
                                        <element name="protectionInfoRequired" type="xsd:boolean"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="authoriseResponse">
                        <complexType>
                                <sequence>
                                        <element name="authoriseReturn" type="tns1:AuthorResult"/>
```

```
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="certify">
                                <complexType>
                                        <sequence>
                                                <element name="axid" type="xsd:string"/>
                                                <element name="axrtid" type="xsd:string"/>
                                                <element name="axdom" type="xsd:string"/>
                                                <element name="toolFingerprint" type="xsd:string"/>
                                                <element name="regDeadline" type="xsd:string"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="certifyResponse">
                                <complexType>
                                        <sequence>
                                                <element name="certifyReturn" type="tns1:CertificationResult"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="reverify">
                                <complexType>
                                        <sequence>
                                                <element name="axid" type="xsd:string"/>
                                                <element name="axtid" type="xsd:string"/>
                                                <element name="axdom" type="xsd:string"/>
                                                <element name="toolFingerprint" type="xsd:string"/>
                                                <element name="lastFPPA" type="xsd:base64Binary"/>
                                                <element maxOccurs="unbounded" name="listOfPA"
type="tns1:ActionLog"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="reverifyResponse">
                                <complexType>
                                        <sequence>
                                                <element name="reverifyReturn" type="tns1:VerificationResult"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="verifyUser">
                                <complexType>
                                        <sequence>
                                                <element name="axid" type="xsd:string"/>
                                                <element name="axdom" type="xsd:string"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="verifyUserResponse">
                                <complexType>
                                        <sequence>
                                                <element name="verifyUserReturn" type="tns1:VerificationResult"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="certifyForMobile">
                                <complexType>
                                        <sequence>
                                                <element name="axid" type="xsd:string"/>
                                                <element name="axrtid" type="xsd:string"/>
                                                <element name="axdom" type="xsd:string"/>
                                                <element name="toolFingerprint" type="xsd:string"/>
                                                <element name="regDeadline" type="xsd:string"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="certifyForMobileResponse">
                                <complexType>
                                        <sequence>
                                                <element name="certifyForMobileReturn"
type="tns1:CertificationResultForMobile"/>
                                        </sequence>
                                </complexType>
                        </element>
```

```
                        <element name="verifyForMobile">
                                <complexType>
                                        <sequence>
                                                <element name="axid" type="xsd:string"/>
                                                <element name="axtid" type="xsd:string"/>
                                                <element name="axdom" type="xsd:string"/>
                                                <element name="toolFingerprintDigest" type="xsd:string"/>
                                                <element name="lastFPPA" type="xsd:string"/>
                                                <element maxOccurs="unbounded" name="listOfPA"
type="tns1:ActionLog"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="verifyForMobileResponse">
                                <complexType>
                                        <sequence>
                                                <element name="verifyForMobileReturn" type="tns1:VerificationResult"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="reverifyForMobile">
                                <complexType>
                                        <sequence>
                                                <element name="axid" type="xsd:string"/>
                                                <element name="axtid" type="xsd:string"/>
                                                <element name="axdom" type="xsd:string"/>
                                                <element name="toolFingerprint" type="xsd:string"/>
                                                <element name="lastFPPA" type="xsd:string"/>
                                                <element maxOccurs="unbounded" name="listOfPA"
type="tns1:ActionLog"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="reverifyForMobileResponse">
                                <complexType>
                                        <sequence>
                                                <element name="reverifyForMobileReturn" type="tns1:VerificationResult"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="updateProtectionInfo">
                                <complexType>
                                        <sequence>
                                                <element name="id" type="xsd:string"/>
                                                <element name="version" type="xsd:string"/>
                                                <element name="protstamp" type="xsd:string"/>
                                                <element name="protinfo" type="xsd:string"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="updateProtectionInfoResponse">
                                <complexType>
                                        <sequence>
                                                <element name="updateProtectionInfoReturn" type="xsd:int"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="doUserRegistration">
                                <complexType>
                                        <sequence>
                                                <element name="AXUID" type="xsd:string"/>
                                                <element name="AXDOM" type="xsd:string"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="doUserRegistrationResponse">
                                <complexType>
                                        <sequence>
                                                <element name="doUserRegistrationReturn" type="xsd:string"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="doUserUnregistration">
                                <complexType>
                                        <sequence>
```

```
                                        <element name="AXUID" type="xsd:string"/>
                                        <element name="AXDOM" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="doUserUnregistrationResponse">
                        <complexType>
                                <sequence>
                                        <element name="doUserUnregistrationReturn" type="xsd:int"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="isUserAlreadyRegistered">
                        <complexType>
                                <sequence>
                                        <element name="AXUID" type="xsd:string"/>
                                        <element name="AXDOM" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="isUserAlreadyRegisteredResponse">
                        <complexType>
                                <sequence>
                                        <element name="isUserAlreadyRegisteredReturn" type="xsd:boolean"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="retrieveRegisteredUsers">
                        <complexType>
                                <sequence>
                                        <element name="AXDOM" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="retrieveRegisteredUsersResponse">
                        <complexType>
                                <sequence>
                                        <element maxOccurs="unbounded" name="retrieveRegisteredUsersReturn"
type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="RetrieveDomains">
                        <complexType>
                                <sequence>
                                        <element name="xValue" type="xsd:int"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="RetrieveDomainsResponse">
                        <complexType>
                                <sequence>
                                        <element maxOccurs="unbounded" name="RetrieveDomainsReturn"
type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="Ping">
                        <complexType>
                                <sequence>
                                        <element name="x" type="xsd:int"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="PingResponse">
                        <complexType>
                                <sequence>
                                        <element name="PingReturn" type="xsd:int"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="generateTranslation">
                        <complexType>
                                <sequence>
                                        <element name="_license" type="xsd:string"/>
```

```
                                        <element name="_originalRel" type="xsd:string"/>
                                        <element name="_destinationRel" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="generateTranslationResponse">
                        <complexType>
                                <sequence>
                                        <element name="generateTranslationReturn" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="verifyTempDistLicenseAgainstPAR">
                        <complexType>
                                <sequence>
                                        <element name="distrLicense" type="xsd:string"/>
                                        <element name="par" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="verifyTempDistLicenseAgainstPARResponse">
                        <complexType>
                                <sequence>
                                        <element name="verifyTempDistLicenseAgainstPARReturn"
type="xsd:boolean"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="verifyTempDistLicenseAgainstPARDatabase">
                        <complexType>
                                <sequence>
                                        <element name="distrLicense" type="xsd:string"/>
                                        <element name="pardatabase" type="xsd:string"/>
                                        <element name="pardatabasehost" type="xsd:string"/>
                                        <element name="pardatabaseuser" type="xsd:string"/>
                                        <element name="pardatabasepass" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="verifyTempDistLicenseAgainstPARDatabaseResponse">
                        <complexType>
                                <sequence>
                                        <element name="verifyTempDistLicenseAgainstPARDatabaseReturn"
type="xsd:boolean"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="verifyLicense">
                        <complexType>
                                <sequence>
                                        <element name="license" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="verifyLicenseResponse">
                        <complexType>
                                <sequence>
                                        <element name="verifyLicenseReturn" type="xsd:boolean"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="verifyPAR">
                        <complexType>
                                <sequence>
                                        <element name="par" type="xsd:string"/>
                                </sequence>
                        </complexType>
                </element>
                <element name="verifyPARResponse">
                        <complexType>
                                <sequence>
                                        <element name="verifyPARReturn" type="xsd:boolean"/>
                                </sequence>
                        </complexType>
                </element>
```

```
                        <element name="verify">
                                <complexType>
                                        <sequence>
                                                <element name="axid" type="xsd:string"/>
                                                <element name="axtid" type="xsd:string"/>
                                                <element name="axdom" type="xsd:string"/>
                                                <element name="toolFingerprintDigest" type="xsd:base64Binary"/>
                                                <element name="lastFPPA" type="xsd:base64Binary"/>
                                                <element maxOccurs="unbounded" name="listOfPA"
type="tns1:ActionLog"/>
                                        </sequence>
                                </complexType>
                        </element>
                        <element name="verifyResponse">
                                <complexType>
                                        <sequence>
                                                <element name="verifyReturn" type="tns1:VerificationResult"/>
                                        </sequence>
                                </complexType>
                        </element>
                </schema>
                <schema elementFormDefault="qualified" targetNamespace="http://AXCV"
xmlns="http://www.w3.org/2001/XMLSchema">
                        <complexType name="contextData">
                                <sequence>
                                        <element name="timesUsed" type="xsd:int"/>
                                        <element name="territoryOfEmission" nillable="true" type="xsd:string"/>
                                </sequence>
                        </complexType>
                        <complexType name="ActionLog">
                                <sequence>
                                        <element name="AXCID" nillable="true" type="xsd:string"/>
                                        <element name="AXCSID" nillable="true" type="xsd:string"/>
                                        <element name="AXDID" nillable="true" type="xsd:string"/>
                                        <element name="AXDOM" nillable="true" type="xsd:string"/>
                                        <element name="AXLID" nillable="true" type="xsd:string"/>
                                        <element name="AXOID" nillable="true" type="xsd:string"/>
                                        <element name="AXTID" nillable="true" type="xsd:string"/>
                                        <element name="AXUID" nillable="true" type="xsd:string"/>
                                        <element name="AXWID" nillable="true" type="xsd:string"/>
                                        <element name="estimatedHwFingerprint" nillable="true" type="xsd:string"/>
                                        <element name="executionTimestamp" nillable="true" type="xsd:string"/>
                                        <element name="histVerSuccess" nillable="true" type="xsd:string"/>
                                        <element name="location" nillable="true" type="xsd:string"/>
                                        <element name="logID" nillable="true" type="xsd:string"/>
                                        <element name="objectVersion" nillable="true" type="xsd:string"/>
                                        <element name="operation" nillable="true" type="xsd:string"/>
                                        <element name="operationDetails" nillable="true" type="xsd:string"/>
                                        <element name="ownerName" nillable="true" type="xsd:string"/>
                                        <element name="protectionStamp" nillable="true" type="xsd:string"/>
                                        <element name="registrationTimestamp" nillable="true" type="xsd:string"/>
                                </sequence>
                        </complexType>
                        <complexType name="AuthorResult">
                                <sequence>
                                        <element name="resultAuth" type="xsd:int"/>
                                        <element name="constructingAL" nillable="true" type="tns1:ActionLog"/>
                                        <element name="protectionKey" nillable="true" type="xsd:string"/>
                                </sequence>
                        </complexType>
                        <complexType name="CertificationResult">
                                <sequence>
                                        <element name="axtid" nillable="true" type="xsd:string"/>
                                        <element name="certificationResult" type="xsd:int"/>
                                        <element name="enablingCode" nillable="true" type="xsd:string"/>
                                        <element name="toolBase64PKCS12" nillable="true" type="xsd:base64Binary"/>
                                </sequence>
                        </complexType>
                        <complexType name="VerificationResult">
                                <sequence>
                                        <element name="storeListActionLogResult" type="xsd:int"/>
                                        <element name="verificationResult" type="xsd:int"/>
                                </sequence>
                        </complexType>
                        <complexType name="CertificationResultForMobile">
```

```
                                        <sequence>
                                                <element name="axtid" nillable="true" type="xsd:string"/>
                                                <element name="certificationResult" type="xsd:int"/>
                                                <element name="enablingCode" nillable="true" type="xsd:string"/>
                                                <element name="toolBase64PKCS12" nillable="true" type="xsd:string"/>
                                        </sequence>
                                </complexType>
                        </schema>
                </wsdl:types>
                <wsdl:message name="initLicenseDistributorRequest">
                        <wsdl:part name="parameters" element="impl:initLicenseDistributor"/>
                </wsdl:message>
                <wsdl:message name="verifyUserResponse">
                        <wsdl:part name="parameters" element="impl:verifyUserResponse"/>
                </wsdl:message>
                <wsdl:message name="verifyTempDistLicenseAgainstPARDatabaseRequest">
                        <wsdl:part name="parameters" element="impl:verifyTempDistLicenseAgainstPARDatabase"/>
                </wsdl:message>
                <wsdl:message name="updateProtectionInfoRequest">
                        <wsdl:part name="parameters" element="impl:updateProtectionInfo"/>
                </wsdl:message>
                <wsdl:message name="addPARGrantEndUserResponse">
                        <wsdl:part name="parameters" element="impl:addPARGrantEndUserResponse"/>
                </wsdl:message>
                <wsdl:message name="RetrieveDomainsResponse">
                        <wsdl:part name="parameters" element="impl:RetrieveDomainsResponse"/>
                </wsdl:message>
                <wsdl:message name="getLicenseRequest">
                        <wsdl:part name="parameters" element="impl:getLicense"/>
                </wsdl:message>
                <wsdl:message name="getPARResponse">
                        <wsdl:part name="parameters" element="impl:getPARResponse"/>
                </wsdl:message>
                <wsdl:message name="doUserRegistrationRequest">
                        <wsdl:part name="parameters" element="impl:doUserRegistration"/>
                </wsdl:message>
                <wsdl:message name="finaliseLicenseEndUserRequest">
                        <wsdl:part name="parameters" element="impl:finaliseLicenseEndUser"/>
                </wsdl:message>
                <wsdl:message name="sendLicenseResponse">
                        <wsdl:part name="parameters" element="impl:sendLicenseResponse"/>
                </wsdl:message>
                <wsdl:message name="verifyLicenseResponse">
                        <wsdl:part name="parameters" element="impl:verifyLicenseResponse"/>
                </wsdl:message>
                <wsdl:message name="finalisePAREndUserRequest">
                        <wsdl:part name="parameters" element="impl:finalisePAREndUser"/>
                </wsdl:message>
                <wsdl:message name="verifyForMobileResponse">
                        <wsdl:part name="parameters" element="impl:verifyForMobileResponse"/>
                </wsdl:message>
                <wsdl:message name="generateTranslationRequest">
                        <wsdl:part name="parameters" element="impl:generateTranslation"/>
                </wsdl:message>
                <wsdl:message name="getLicenseResponse">
                        <wsdl:part name="parameters" element="impl:getLicenseResponse"/>
                </wsdl:message>
                <wsdl:message name="finalisePARDistributorResponse">
                        <wsdl:part name="parameters" element="impl:finalisePARDistributorResponse"/>
                </wsdl:message>
                <wsdl:message name="RetrieveDomainsRequest">
                        <wsdl:part name="parameters" element="impl:RetrieveDomains"/>
                </wsdl:message>
                <wsdl:message name="initPAREndUserRequest">
                        <wsdl:part name="parameters" element="impl:initPAREndUser"/>
                </wsdl:message>
                <wsdl:message name="reverifyForMobileRequest">
                        <wsdl:part name="parameters" element="impl:reverifyForMobile"/>
                </wsdl:message>
                <wsdl:message name="finalisePAREndUserResponse">
                        <wsdl:part name="parameters" element="impl:finalisePAREndUserResponse"/>
                </wsdl:message>
                <wsdl:message name="authoriseResponse">
                        <wsdl:part name="parameters" element="impl:authoriseResponse"/>
                </wsdl:message>
```

```
<wsdl:message name="PingResponse">
	<wsdl:part name="parameters" element="impl:PingResponse"/>
</wsdl:message>
<wsdl:message name="reverifyForMobileResponse">
	<wsdl:part name="parameters" element="impl:reverifyForMobileResponse"/>
</wsdl:message>
<wsdl:message name="authoriseRequest">
	<wsdl:part name="parameters" element="impl:authorise"/>
</wsdl:message>
<wsdl:message name="addPARGrantforEndUserRequest">
	<wsdl:part name="parameters" element="impl:addPARGrantforEndUser"/>
</wsdl:message>
<wsdl:message name="initLicenseDistributorResponse">
	<wsdl:part name="parameters" element="impl:initLicenseDistributorResponse"/>
</wsdl:message>
<wsdl:message name="sendPARResponse">
	<wsdl:part name="parameters" element="impl:sendPARResponse"/>
</wsdl:message>
<wsdl:message name="verifyPARRequest">
	<wsdl:part name="parameters" element="impl:verifyPAR"/>
</wsdl:message>
<wsdl:message name="initPARDistributorResponse">
	<wsdl:part name="parameters" element="impl:initPARDistributorResponse"/>
</wsdl:message>
<wsdl:message name="initLicenseEndUserRequest">
	<wsdl:part name="parameters" element="impl:initLicenseEndUser"/>
</wsdl:message>
<wsdl:message name="sendLicenseRequest">
	<wsdl:part name="parameters" element="impl:sendLicense"/>
</wsdl:message>
<wsdl:message name="certifyRequest">
	<wsdl:part name="parameters" element="impl:certify"/>
</wsdl:message>
<wsdl:message name="verifyResponse">
	<wsdl:part name="parameters" element="impl:verifyResponse"/>
</wsdl:message>
<wsdl:message name="finaliseLicenseEndUserResponse">
	<wsdl:part name="parameters" element="impl:finaliseLicenseEndUserResponse"/>
</wsdl:message>
<wsdl:message name="certifyForMobileRequest">
	<wsdl:part name="parameters" element="impl:certifyForMobile"/>
</wsdl:message>
<wsdl:message name="verifyTempDistLicenseAgainstPARDatabaseResponse">
	<wsdl:part name="parameters" element="impl:verifyTempDistLicenseAgainstPARDatabaseResponse"/>
</wsdl:message>
<wsdl:message name="PingRequest">
	<wsdl:part name="parameters" element="impl:Ping"/>
</wsdl:message>
<wsdl:message name="isUserAlreadyRegisteredResponse">
	<wsdl:part name="parameters" element="impl:isUserAlreadyRegisteredResponse"/>
</wsdl:message>
<wsdl:message name="addGrantEndUserRequest">
	<wsdl:part name="parameters" element="impl:addGrantEndUser"/>
</wsdl:message>
<wsdl:message name="verifyTempDistLicenseAgainstPARResponse">
	<wsdl:part name="parameters" element="impl:verifyTempDistLicenseAgainstPARResponse"/>
</wsdl:message>
<wsdl:message name="addPARGrantforDistributorRequest">
	<wsdl:part name="parameters" element="impl:addPARGrantforDistributor"/>
</wsdl:message>
<wsdl:message name="retrieveRegisteredUsersRequest">
	<wsdl:part name="parameters" element="impl:retrieveRegisteredUsers"/>
</wsdl:message>
<wsdl:message name="certifyResponse">
	<wsdl:part name="parameters" element="impl:certifyResponse"/>
</wsdl:message>
<wsdl:message name="reverifyResponse">
	<wsdl:part name="parameters" element="impl:reverifyResponse"/>
</wsdl:message>
<wsdl:message name="doUserUnregistrationRequest">
	<wsdl:part name="parameters" element="impl:doUserUnregistration"/>
</wsdl:message>
<wsdl:message name="verifyLicenseRequest">
	<wsdl:part name="parameters" element="impl:verifyLicense"/>
</wsdl:message>
```

```
<wsdl:message name="finalisePARDistributorRequest">
	<wsdl:part name="parameters" element="impl:finalisePARDistributor"/>
</wsdl:message>
<wsdl:message name="verifyPARResponse">
	<wsdl:part name="parameters" element="impl:verifyPARResponse"/>
</wsdl:message>
<wsdl:message name="reverifyRequest">
	<wsdl:part name="parameters" element="impl:reverify"/>
</wsdl:message>
<wsdl:message name="verifyRequest">
	<wsdl:part name="parameters" element="impl:verify"/>
</wsdl:message>
<wsdl:message name="finaliseLicenseDistributorRequest">
	<wsdl:part name="parameters" element="impl:finaliseLicenseDistributor"/>
</wsdl:message>
<wsdl:message name="updateProtectionInfoResponse">
	<wsdl:part name="parameters" element="impl:updateProtectionInfoResponse"/>
</wsdl:message>
<wsdl:message name="retrieveRegisteredUsersResponse">
	<wsdl:part name="parameters" element="impl:retrieveRegisteredUsersResponse"/>
</wsdl:message>
<wsdl:message name="doUserUnregistrationResponse">
	<wsdl:part name="parameters" element="impl:doUserUnregistrationResponse"/>
</wsdl:message>
<wsdl:message name="doUserRegistrationResponse">
	<wsdl:part name="parameters" element="impl:doUserRegistrationResponse"/>
</wsdl:message>
<wsdl:message name="addGrantforEndUserRequest">
	<wsdl:part name="parameters" element="impl:addGrantforEndUser"/>
</wsdl:message>
<wsdl:message name="addGrantforDistributorResponse">
	<wsdl:part name="parameters" element="impl:addGrantforDistributorResponse"/>
</wsdl:message>
<wsdl:message name="isUserAlreadyRegisteredRequest">
	<wsdl:part name="parameters" element="impl:isUserAlreadyRegistered"/>
</wsdl:message>
<wsdl:message name="certifyForMobileResponse">
	<wsdl:part name="parameters" element="impl:certifyForMobileResponse"/>
</wsdl:message>
<wsdl:message name="initPAREndUserResponse">
	<wsdl:part name="parameters" element="impl:initPAREndUserResponse"/>
</wsdl:message>
<wsdl:message name="getPARRequest">
	<wsdl:part name="parameters" element="impl:getPAR"/>
</wsdl:message>
<wsdl:message name="addPARGrantEndUserRequest">
	<wsdl:part name="parameters" element="impl:addPARGrantEndUser"/>
</wsdl:message>
<wsdl:message name="initLicenseEndUserResponse">
	<wsdl:part name="parameters" element="impl:initLicenseEndUserResponse"/>
</wsdl:message>
<wsdl:message name="addPARGrantforEndUserResponse">
	<wsdl:part name="parameters" element="impl:addPARGrantforEndUserResponse"/>
</wsdl:message>
<wsdl:message name="finaliseLicenseDistributorResponse">
	<wsdl:part name="parameters" element="impl:finaliseLicenseDistributorResponse"/>
</wsdl:message>
<wsdl:message name="addGrantforEndUserResponse">
	<wsdl:part name="parameters" element="impl:addGrantforEndUserResponse"/>
</wsdl:message>
<wsdl:message name="verifyForMobileRequest">
	<wsdl:part name="parameters" element="impl:verifyForMobile"/>
</wsdl:message>
<wsdl:message name="addGrantEndUserResponse">
	<wsdl:part name="parameters" element="impl:addGrantEndUserResponse"/>
</wsdl:message>
<wsdl:message name="initPARDistributorRequest">
	<wsdl:part name="parameters" element="impl:initPARDistributor"/>
</wsdl:message>
<wsdl:message name="addGrantforDistributorRequest">
	<wsdl:part name="parameters" element="impl:addGrantforDistributor"/>
</wsdl:message>
<wsdl:message name="sendPARRequest">
	<wsdl:part name="parameters" element="impl:sendPAR"/>
</wsdl:message>
```

```
	<wsdl:message name="addPARGrantforDistributorResponse">
		<wsdl:part name="parameters" element="impl:addPARGrantforDistributorResponse"/>
	</wsdl:message>
	<wsdl:message name="verifyUserRequest">
		<wsdl:part name="parameters" element="impl:verifyUser"/>
	</wsdl:message>
	<wsdl:message name="verifyTempDistLicenseAgainstPARRequest">
		<wsdl:part name="parameters" element="impl:verifyTempDistLicenseAgainstPAR"/>
	</wsdl:message>
	<wsdl:message name="generateTranslationResponse">
		<wsdl:part name="parameters" element="impl:generateTranslationResponse"/>
	</wsdl:message>
	<wsdl:portType name="PMS">
		<wsdl:operation name="initLicenseEndUser">
			<wsdl:input name="initLicenseEndUserRequest" message="impl:initLicenseEndUserRequest"/>
			<wsdl:output name="initLicenseEndUserResponse" message="impl:initLicenseEndUserResponse"/>
		</wsdl:operation>
		<wsdl:operation name="addGrantEndUser">
			<wsdl:input name="addGrantEndUserRequest" message="impl:addGrantEndUserRequest"/>
			<wsdl:output name="addGrantEndUserResponse" message="impl:addGrantEndUserResponse"/>
		</wsdl:operation>
		<wsdl:operation name="finaliseLicenseEndUser">
			<wsdl:input name="finaliseLicenseEndUserRequest" message="impl:finaliseLicenseEndUserRequest"/>
			<wsdl:output name="finaliseLicenseEndUserResponse"
message="impl:finaliseLicenseEndUserResponse"/>
		</wsdl:operation>
		<wsdl:operation name="initLicenseDistributor">
			<wsdl:input name="initLicenseDistributorRequest" message="impl:initLicenseDistributorRequest"/>
			<wsdl:output name="initLicenseDistributorResponse" message="impl:initLicenseDistributorResponse"/>
		</wsdl:operation>
		<wsdl:operation name="addGrantforDistributor">
			<wsdl:input name="addGrantforDistributorRequest" message="impl:addGrantforDistributorRequest"/>
			<wsdl:output name="addGrantforDistributorResponse"
message="impl:addGrantforDistributorResponse"/>
		</wsdl:operation>
		<wsdl:operation name="addGrantforEndUser">
			<wsdl:input name="addGrantforEndUserRequest" message="impl:addGrantforEndUserRequest"/>
			<wsdl:output name="addGrantforEndUserResponse" message="impl:addGrantforEndUserResponse"/>
		</wsdl:operation>
		<wsdl:operation name="finaliseLicenseDistributor">
			<wsdl:input name="finaliseLicenseDistributorRequest"
message="impl:finaliseLicenseDistributorRequest"/>
			<wsdl:output name="finaliseLicenseDistributorResponse"
message="impl:finaliseLicenseDistributorResponse"/>
		</wsdl:operation>
		<wsdl:operation name="getLicense">
			<wsdl:input name="getLicenseRequest" message="impl:getLicenseRequest"/>
			<wsdl:output name="getLicenseResponse" message="impl:getLicenseResponse"/>
		</wsdl:operation>
		<wsdl:operation name="sendLicense">
			<wsdl:input name="sendLicenseRequest" message="impl:sendLicenseRequest"/>
			<wsdl:output name="sendLicenseResponse" message="impl:sendLicenseResponse"/>
		</wsdl:operation>
		<wsdl:operation name="initPAREndUser">
			<wsdl:input name="initPAREndUserRequest" message="impl:initPAREndUserRequest"/>
			<wsdl:output name="initPAREndUserResponse" message="impl:initPAREndUserResponse"/>
		</wsdl:operation>
		<wsdl:operation name="addPARGrantEndUser">
			<wsdl:input name="addPARGrantEndUserRequest" message="impl:addPARGrantEndUserRequest"/>
			<wsdl:output name="addPARGrantEndUserResponse"
message="impl:addPARGrantEndUserResponse"/>
		</wsdl:operation>
		<wsdl:operation name="finalisePAREndUser">
			<wsdl:input name="finalisePAREndUserRequest" message="impl:finalisePAREndUserRequest"/>
			<wsdl:output name="finalisePAREndUserResponse" message="impl:finalisePAREndUserResponse"/>
		</wsdl:operation>
		<wsdl:operation name="initPARDistributor">
			<wsdl:input name="initPARDistributorRequest" message="impl:initPARDistributorRequest"/>
			<wsdl:output name="initPARDistributorResponse" message="impl:initPARDistributorResponse"/>
		</wsdl:operation>
		<wsdl:operation name="addPARGrantforDistributor">
			<wsdl:input name="addPARGrantforDistributorRequest"
message="impl:addPARGrantforDistributorRequest"/>
			<wsdl:output name="addPARGrantforDistributorResponse"
message="impl:addPARGrantforDistributorResponse"/>
```

```
                </wsdl:operation>
                <wsdl:operation name="addPARGrantforEndUser">
                        <wsdl:input name="addPARGrantforEndUserRequest"
message="impl:addPARGrantforEndUserRequest"/>
                        <wsdl:output name="addPARGrantforEndUserResponse"
message="impl:addPARGrantforEndUserResponse"/>
                </wsdl:operation>
                <wsdl:operation name="finalisePARDistributor">
                        <wsdl:input name="finalisePARDistributorRequest" message="impl:finalisePARDistributorRequest"/>
                        <wsdl:output name="finalisePARDistributorResponse" message="impl:finalisePARDistributorResponse"/>
                </wsdl:operation>
                <wsdl:operation name="getPAR">
                        <wsdl:input name="getPARRequest" message="impl:getPARRequest"/>
                        <wsdl:output name="getPARResponse" message="impl:getPARResponse"/>
                </wsdl:operation>
                <wsdl:operation name="sendPAR">
                        <wsdl:input name="sendPARRequest" message="impl:sendPARRequest"/>
                        <wsdl:output name="sendPARResponse" message="impl:sendPARResponse"/>
                </wsdl:operation>
                <wsdl:operation name="authorise">
                        <wsdl:input name="authoriseRequest" message="impl:authoriseRequest"/>
                        <wsdl:output name="authoriseResponse" message="impl:authoriseResponse"/>
                </wsdl:operation>
                <wsdl:operation name="certify">
                        <wsdl:input name="certifyRequest" message="impl:certifyRequest"/>
                        <wsdl:output name="certifyResponse" message="impl:certifyResponse"/>
                </wsdl:operation>
                <wsdl:operation name="reverify">
                        <wsdl:input name="reverifyRequest" message="impl:reverifyRequest"/>
                        <wsdl:output name="reverifyResponse" message="impl:reverifyResponse"/>
                </wsdl:operation>
                <wsdl:operation name="verifyUser">
                        <wsdl:input name="verifyUserRequest" message="impl:verifyUserRequest"/>
                        <wsdl:output name="verifyUserResponse" message="impl:verifyUserResponse"/>
                </wsdl:operation>
                <wsdl:operation name="certifyForMobile">
                        <wsdl:input name="certifyForMobileRequest" message="impl:certifyForMobileRequest"/>
                        <wsdl:output name="certifyForMobileResponse" message="impl:certifyForMobileResponse"/>
                </wsdl:operation>
                <wsdl:operation name="verifyForMobile">
                        <wsdl:input name="verifyForMobileRequest" message="impl:verifyForMobileRequest"/>
                        <wsdl:output name="verifyForMobileResponse" message="impl:verifyForMobileResponse"/>
                </wsdl:operation>
                <wsdl:operation name="reverifyForMobile">
                        <wsdl:input name="reverifyForMobileRequest" message="impl:reverifyForMobileRequest"/>
                        <wsdl:output name="reverifyForMobileResponse" message="impl:reverifyForMobileResponse"/>
                </wsdl:operation>
                <wsdl:operation name="updateProtectionInfo">
                        <wsdl:input name="updateProtectionInfoRequest" message="impl:updateProtectionInfoRequest"/>
                        <wsdl:output name="updateProtectionInfoResponse" message="impl:updateProtectionInfoResponse"/>
                </wsdl:operation>
                <wsdl:operation name="doUserRegistration">
                        <wsdl:input name="doUserRegistrationRequest" message="impl:doUserRegistrationRequest"/>
                        <wsdl:output name="doUserRegistrationResponse" message="impl:doUserRegistrationResponse"/>
                </wsdl:operation>
                <wsdl:operation name="doUserUnregistration">
                        <wsdl:input name="doUserUnregistrationRequest" message="impl:doUserUnregistrationRequest"/>
                        <wsdl:output name="doUserUnregistrationResponse" message="impl:doUserUnregistrationResponse"/>
                </wsdl:operation>
                <wsdl:operation name="isUserAlreadyRegistered">
                        <wsdl:input name="isUserAlreadyRegisteredRequest"
message="impl:isUserAlreadyRegisteredRequest"/>
                        <wsdl:output name="isUserAlreadyRegisteredResponse"
message="impl:isUserAlreadyRegisteredResponse"/>
                </wsdl:operation>
                <wsdl:operation name="retrieveRegisteredUsers">
                        <wsdl:input name="retrieveRegisteredUsersRequest" message="impl:retrieveRegisteredUsersRequest"/>
                        <wsdl:output name="retrieveRegisteredUsersResponse"
message="impl:retrieveRegisteredUsersResponse"/>
                </wsdl:operation>
                <wsdl:operation name="RetrieveDomains">
                        <wsdl:input name="RetrieveDomainsRequest" message="impl:RetrieveDomainsRequest"/>
                        <wsdl:output name="RetrieveDomainsResponse" message="impl:RetrieveDomainsResponse"/>
                </wsdl:operation>
                <wsdl:operation name="Ping">
```

```
                        <wsdl:input name="PingRequest" message="impl:PingRequest"/>
                        <wsdl:output name="PingResponse" message="impl:PingResponse"/>
                </wsdl:operation>
                <wsdl:operation name="generateTranslation">
                        <wsdl:input name="generateTranslationRequest" message="impl:generateTranslationRequest"/>
                        <wsdl:output name="generateTranslationResponse" message="impl:generateTranslationResponse"/>
                </wsdl:operation>
                <wsdl:operation name="verifyTempDistLicenseAgainstPAR">
                        <wsdl:input name="verifyTempDistLicenseAgainstPARRequest"
message="impl:verifyTempDistLicenseAgainstPARRequest"/>
                        <wsdl:output name="verifyTempDistLicenseAgainstPARResponse"
message="impl:verifyTempDistLicenseAgainstPARResponse"/>
                </wsdl:operation>
                <wsdl:operation name="verifyTempDistLicenseAgainstPARDatabase">
                        <wsdl:input name="verifyTempDistLicenseAgainstPARDatabaseRequest"
message="impl:verifyTempDistLicenseAgainstPARDatabaseRequest"/>
                        <wsdl:output name="verifyTempDistLicenseAgainstPARDatabaseResponse"
message="impl:verifyTempDistLicenseAgainstPARDatabaseResponse"/>
                </wsdl:operation>
                <wsdl:operation name="verifyLicense">
                        <wsdl:input name="verifyLicenseRequest" message="impl:verifyLicenseRequest"/>
                        <wsdl:output name="verifyLicenseResponse" message="impl:verifyLicenseResponse"/>
                </wsdl:operation>
                <wsdl:operation name="verifyPAR">
                        <wsdl:input name="verifyPARRequest" message="impl:verifyPARRequest"/>
                        <wsdl:output name="verifyPARResponse" message="impl:verifyPARResponse"/>
                </wsdl:operation>
                <wsdl:operation name="verify">
                        <wsdl:input name="verifyRequest" message="impl:verifyRequest"/>
                        <wsdl:output name="verifyResponse" message="impl:verifyResponse"/>
                </wsdl:operation>
        </wsdl:portType>
        <wsdl:binding name="PMSSoapBinding" type="impl:PMS">
                <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
                <wsdl:operation name="initLicenseEndUser">
                        <wsdlsoap:operation/>
                        <wsdl:input>
                                <wsdlsoap:body use="literal"/>
                        </wsdl:input>
                        <wsdl:output>
                                <wsdlsoap:body use="literal"/>
                        </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="addGrantEndUser">
                        <wsdlsoap:operation/>
                        <wsdl:input>
                                <wsdlsoap:body use="literal"/>
                        </wsdl:input>
                        <wsdl:output>
                                <wsdlsoap:body use="literal"/>
                        </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="finaliseLicenseEndUser">
                        <wsdlsoap:operation/>
                        <wsdl:input>
                                <wsdlsoap:body use="literal"/>
                        </wsdl:input>
                        <wsdl:output>
                                <wsdlsoap:body use="literal"/>
                        </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="initLicenseDistributor">
                        <wsdlsoap:operation/>
                        <wsdl:input>
                                <wsdlsoap:body use="literal"/>
                        </wsdl:input>
                        <wsdl:output>
                                <wsdlsoap:body use="literal"/>
                        </wsdl:output>
                </wsdl:operation>
                <wsdl:operation name="addGrantforDistributor">
                        <wsdlsoap:operation/>
                        <wsdl:input>
                                <wsdlsoap:body use="literal"/>
                        </wsdl:input>
```

```
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="addGrantforEndUser">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="finaliseLicenseDistributor">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="getLicense">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="sendLicense">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="initPAREndUser">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="addPARGrantEndUser">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="finalisePAREndUser">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="initPARDistributor">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
```

```
		</wsdl:operation>
		<wsdl:operation name="addPARGrantforDistributor">
			<wsdlsoap:operation/>
			<wsdl:input>
				<wsdlsoap:body use="literal"/>
			</wsdl:input>
			<wsdl:output>
				<wsdlsoap:body use="literal"/>
			</wsdl:output>
		</wsdl:operation>
		<wsdl:operation name="addPARGrantforEndUser">
			<wsdlsoap:operation/>
			<wsdl:input>
				<wsdlsoap:body use="literal"/>
			</wsdl:input>
			<wsdl:output>
				<wsdlsoap:body use="literal"/>
			</wsdl:output>
		</wsdl:operation>
		<wsdl:operation name="finalisePARDistributor">
			<wsdlsoap:operation/>
			<wsdl:input>
				<wsdlsoap:body use="literal"/>
			</wsdl:input>
			<wsdl:output>
				<wsdlsoap:body use="literal"/>
			</wsdl:output>
		</wsdl:operation>
		<wsdl:operation name="getPAR">
			<wsdlsoap:operation/>
			<wsdl:input>
				<wsdlsoap:body use="literal"/>
			</wsdl:input>
			<wsdl:output>
				<wsdlsoap:body use="literal"/>
			</wsdl:output>
		</wsdl:operation>
		<wsdl:operation name="sendPAR">
			<wsdlsoap:operation/>
			<wsdl:input>
				<wsdlsoap:body use="literal"/>
			</wsdl:input>
			<wsdl:output>
				<wsdlsoap:body use="literal"/>
			</wsdl:output>
		</wsdl:operation>
		<wsdl:operation name="authorise">
			<wsdlsoap:operation/>
			<wsdl:input>
				<wsdlsoap:body use="literal"/>
			</wsdl:input>
			<wsdl:output>
				<wsdlsoap:body use="literal"/>
			</wsdl:output>
		</wsdl:operation>
		<wsdl:operation name="certify">
			<wsdlsoap:operation/>
			<wsdl:input>
				<wsdlsoap:body use="literal"/>
			</wsdl:input>
			<wsdl:output>
				<wsdlsoap:body use="literal"/>
			</wsdl:output>
		</wsdl:operation>
		<wsdl:operation name="reverify">
			<wsdlsoap:operation/>
			<wsdl:input>
				<wsdlsoap:body use="literal"/>
			</wsdl:input>
			<wsdl:output>
				<wsdlsoap:body use="literal"/>
			</wsdl:output>
		</wsdl:operation>
		<wsdl:operation name="verifyUser">
			<wsdlsoap:operation/>
```

```
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="certifyForMobile">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="verifyForMobile">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="reverifyForMobile">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="updateProtectionInfo">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="doUserRegistration">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="doUserUnregistration">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="isUserAlreadyRegistered">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="retrieveRegisteredUsers">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
```

```
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="RetrieveDomains">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="Ping">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="generateTranslation">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="verifyTempDistLicenseAgainstPAR">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="verifyTempDistLicenseAgainstPARDatabase">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="verifyLicense">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="verifyPAR">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="verify">
                <wsdlsoap:operation/>
                <wsdl:input>
                        <wsdlsoap:body use="literal"/>
                </wsdl:input>
                <wsdl:output>
                        <wsdlsoap:body use="literal"/>
                </wsdl:output>
```

```
                </wsdl:operation>
        </wsdl:binding>
        <wsdl:service name="PMSService">
                <wsdl:port name="PMS" binding="impl:PMSSoapBinding">
                        <wsdlsoap:address location="http://localhost:8502/PMS"/>
                </wsdl:port>
        </wsdl:service>
        <!--WSDL created by Apache Axis version: 1.2.1on Jun 14, 2005 (09:15:57 EDT)-->
</wsdl:definitions>
```

32 Bibliography

[1] ISO/IEC. ISO/IEC IS 21000-5 - MPEG-21 - Rights Expression Language
[2] Open Mobile Alliance (OMA). DRM Rights Expression Language. http://www.openmobilealliance.org/release_program/docs/DRM/V2_0-20050825-C/OMA-TS-DRM-REL-V2_0-20050825-C.pdf
[3] ISO/IEC. ISO/IEC 21000-5/FPDAM 1- MPEG-21 - Part 5: Rights Expression Language, Amendment 1: MPEG-21 REL profiles.