**Automating Production of Cross Media Content for Multi-channel Distribution**

**www.AXMEDIS.org**

# DE3.1.2.2.13 Specification of AXMEDIS Certifier and Supervisor and networks of AXCS, first update of part of DE3.1.2

**Version:** 1.5
**Date:** 09-05-2006
**Responsible: DSI (chellini@dsi.unifi.it, martini@dsi.unifi.it)**
**(verified and closed by coordinator)**

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

Deliverable Number: DE3.1.2.2.13
Contractual Date of Delivery: M18
Actual Date of Delivery: 15/04/2006
Title of Deliverable: Specification of AXMEDIS Certifier and Supervisor and networks of AXCS, first update of part of DE3.1.2
Work-Package contributing to the Deliverable: WP3.1
Task contributing to the Deliverable: WP3, WP2
Nature of the Deliverable: report
Author(s): DSI, FUPF

**Abstract:** this part includes the specification of components, formats, databases and protocol related to the AXMEDIS Framework area of protection Certification and suerpvision including AXCS

**Keyword List:** Certification and supervision, event reporting, action log, user registration, device registration and certification, requests of actions logs, statistical data on content rights exploitation, object registration, object identification.

**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 | respons ible |
|---|---|---|
| DE3.1.2.2.1 | Specification of General Aspects of AXMEDIS framework, first update of DE3.1.2 part A<br><br>AXMEDIS-DE3-1-2-2-1-Spec-of-AX-Gen-Asp-of-AXMEDIS-framework-upA-v1-0.doc | DSI |
| DE3.1.2.2.2 | Specification of AXMEDIS Command Manager, first update of DE3.1.2 part B<br><br>AXMEDIS- DE3-1-2-2-2-Spec-of-AX-Cmd-Man-upB-v1-0.doc | DSI |
| DE3.1.2.2.3 | Specification of AXMEDIS Object Manager and Protection Processor, first update of DE3.1.2 part B<br><br>AXMEDIS-DE3-1-2-2-3-Spec-of-AXOM-and-ProtProc-upB-v1-0.doc | DSI |
| DE3.1.2.2.4 | Specification of AXMEDIS Editors and Viewers, first update of DE3.1.2 part B<br><br>AXMEDIS-DE3-1-2-2-4-Spec-of-AX-Editors-and-Viewers-upB-v1-0.doc | DSI |
| DE3.1.2.2.5 | Specification of External AXMEDIS Editors/Viewers and Players, first update of DE3.1.2 part B<br><br>AXMEDIS-DE3-1-2-2-5-Spec-of-External-Editors-Viewers-Players-upB-v1-0.doc | EPFL |
| DE3.1.2.2.6 | Specification of AXMEDIS Content Processing, first update of DE3.1.2 part C<br><br>AXMEDIS-DE3-1-2-2-6-Spec-of-AX-Content-Processing-upC-v1-0.doc | DSI |
| DE3.1.2.2.7 | Specification of AXMEDIS External Processing Algorithms<br><br>AXMEDIS-DE3-1-2-2-7-Spec-of-AX-External-Processing-Algorithms-v1-0.doc | FHGIGD |
| DE3.1.2.2.8 | Specification of AXMEDIS CMS Crawling Capabilities, first update of part of DE3.1.2<br><br>AXMEDIS-DE3-1-2-2-8-Spec-of-AX-CMS-Crawling-Capab-v1-0.doc | DSI |
| DE3.1.2.2.9 | Specification of AXMEDIS database and query support, first update of part of DE3.1.2<br><br>AXMEDIS-DE3-1-2-2-9-Spec-of-AX-database-and-query-support-v1-0.doc | EXITEC H |
| DE3.1.2.2.10 | Specification of AXMEDIS P2P tools, AXEPTool and AXMEDIS, first update of part of DE3.1.2<br><br>AXMEDIS-DE3-1-2-2-10-Spec-of-AXEPTool-and-AXMEDIA-tools-v1-0.doc | CRS4 |
| DE3.1.2.2.11 | Specification of AXMEDIS Programme and Publication tools, first update of part of DE3.1.2<br><br>AXMEDIS-DE3-1-2-2-11-Spec-of-AX-Progr-and-Pub-tool-v1-0.doc | UNIVLE EDS |
| DE3.1.2.2.12 | Specification of AXMEDIS Workflow Tools, first update of part of DE3.1.2<br><br>AXMEDIS-DE3-1-2-2-12-Spec-of-AX-Workflow-Tools-v1-0.doc | IRC |
| DE3.1.2.2.13 | Specification of AXMEDIS Certifier and Supervisor and networks of AXCS, first update of part of DE3.1.2<br><br>AXMEDIS-DE3-1-2-2-13-Spec-of-AXCS-and-networks-v1-0.doc | DSI |
| DE3.1.2.2.14 | Specification of AXMEDIS Protection Support, first update of part of DE3.1.2<br><br>AXMEDIS-DE3-1-2-2-14-Spec-of-AX-Protection-Support-v1-0.doc | FUPF |
| DE3.1.2.2.15 | Specification of AXMEDIS accounting and reporting, first update of part of DE3.1.2<br><br>AXMEDIS-DE3-1-2-2-15-Spec-of-AX-Accounting-and-Reporting-v1-0.doc | EXITEC H |

## 1.1 This document concerns (DSI)

AXMEDIS Certifier and Supervisor (AXCS) and network of AXCSs.

AXCS is the authority put in charge of supervising the certification process over all its phases. Its own proper tasks concern about user and tool registration, certification and managements, object identifier generation and manipulation, object metadata collection, object usage data registration and managements. All the data collected, managed and elaborated by AXCS have to be available for other subjects in the system entitled to get pertinent information.

Please note that:

- AXCSs do not contain detailed data about users: they know only users ID and all related data about objects usage. Users registration data is collected exclusively by Distributors and/or AXMEDIS Portal.
- AXCSs do not contain any AXMEDIS object, rather they contain metadata about them, such as IDs, Dublin Core metadata, business dependent metadata and so on.

In order to perform its own work, AXCS has to handle a lot of data and to manage a huge number of connections from clients, PMSs and any other subject entitled to deal whith AXCS. Moreover, there must be an AXCS for every single distribution channel. Therefore the AXCS cannot be a unique entity but a set of AXCSs capable of sharing data and satisfying system requests is needed. In order to realize this purpose a network of AXCS has been conceived.

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

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

| Module/tool Name | Module/Tool Description and purpose, state also in which other AXMEDIS area is used | Standards exploited if any |
|---|---|---|
| AXMEDIS Certification and Verification (AXCV) | This module is in charge of certifying and verifying users and tools, providing the appropriate certificates and blocking functionalities | |
| AMEDIS Supervisor (AXS) | This module keeps track of the information related to user actions in the AXMEDIS system. It also provides access to the security information | |
| AXCS Users Registration Web Service | Web Service used to register new users in the AXMEDIS system. It is also used to update data already registered.<br>It is used by Distributors supporting directly users registration and by the AXMEDIS Registration Portal | UUID, WSDL, SOAP |
| AXCS Objects Registration Web Service | Web Service used to register new objects in the AXMEDIS system. It is also used to update data already registered.<br>It is used by Creators, Content Aggregators, etc.<br><br>PLEASE NOTE: registering a new object does not mean to insert it (the file itself) in the AXMEDIS database (AXDB). It means only to insert in the AXCS databases the object related metadata; the object itself is not involved and it is neither transferred to the AXCS | UUID, Dublin Core, WSDL, SOAP |
| AXCS Reporting Web Service | Web Service used to gather data about object usages. It gives back complete information on usages but only about object related to the requestor.<br>It is used by CAMART upon requests received from Collecting Societies, Distributors, Creators, etc. | UUID, WSDL, SOAP |
| AXCS Statistic Web Service | Web Service used to gather data about object usages. It gives back anonymous information on usages about all the objects statisfying the specified criteria.<br>It is used by CAMART upon requests received from Collecting Societies, Distributors, Creators, etc. | WSDL, SOAP |
| AXCS Database Interface | Abstraction layer to give uniform access to the AXCS databases, regardless of the specific RDBMS engine used to realise the databases | SQL |

| | | |
|---|---|---|
| AXCS Tool Off-line Registration | Web application to let tool producers submit their software in order to be examined and tested to become an AXMEDIS compliant tool | |
| AXCS Manager User Interface | This is a grphic interface to easily administer AXCS databases. | |
| AXMEDIS User Registration Portal | Web Portal to let users register in AXMEDIS. | |
| AXCS Object List Web Service | Web service used to get information about all the objects present in the AXMEDIS system. It can be queried by Distributors, Creators, Integrators, and so on (in general B2B Users). | UUID, WSDL, SOAP |

## 1.3 List of Formats Specified in this document (DSI, FUPF)

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 |
|---|---|---|
| AXMEIDS prefixes | Coded three character strings. Codes are reported in section 22. | |
| AXMEDIS ID | An URN composed of the "axmedis" namespace identifier, an AXCS identifier an AXMEDIS prefix and an UUID | UUID |
| AXS formats | Action log | Based on and expanding MPEG-21 Event Reporting |
| AXV formats | X.509 certificates | |

## 1.4 List of Databases Specified in this document (DSI)

| Database Name | database Description and purpose, state also in which other AXMEDIS area is using | Standards exploited if any |
|---|---|---|
| AXCS Registration and Certification Database | It stores data about users and tools.<br>User data are organized grouping generic data (needed for all kinds of user) and separating the others distinguishing among varios business categories (Creators, Distributors, Collecting Societies, Tool Producers)<br>Tool data are organised distinguishing between registered and certified tools. | |
| AXCS ObjectsID Database | It stores objects metadata, both standard ones (Dublin Core) and business model specific ones. | Dublin Core |
| AXCS Accounting Database | It stores data about object usages | |

## 1.5 List of Protocols Specified in this document (DSI, FUPF)

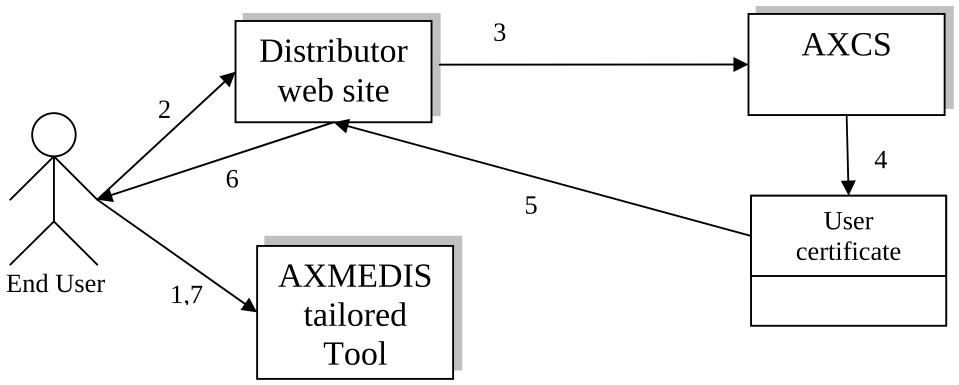
A protocol is a communication modality among distinct processes that can be located or not on different computers.

| 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 |
|---|---|---|---|
| AXCSUserRegistration | The client sends non personal data (nickname, password, email, affiliation, etc.) about the user to be registered. The web service returns an object containing indication about the operation outcome and either the result in case of success (i.e. the assigned AXUID) or an error string in case of failure | Master is the web service, slave or clients are other modules of the framework, such as kiosks, distributors, AXMEDIS Portal, etc. | UUID, WSDL |

| | | | |
|---|---|---|---|
| AXCSObjectRegistration | The client sends metadata (Dublin Core metadata, business model dependant etadata, protection information, etc.) about the object to be registered. The web service returns an object containing indication about the operation outcome and either the result in case of success (the assigned AXOID) or an error string in case of failure | Master is the web service, slave or clients are other modules of the framework (in particular editors) used by users such as creators, etc. | UUID, Dublin Core, WSDL |
| AXCSReporting | The client sends nickname, password and some eventual criteria to filter action log data. The web service returns an object containing indication about the operation outcome and either the result in case of success (requested data) or an error string in case of failure | Master is the web service, slave or client is CAMART | UUID, WSDL |
| AXCSStatistics | The client sends nickname, password and some eventual criteria to filter action log data. The web service returns an object containing indication about the operation outcome and either the result in case of success (requested data in anonymous format) or an error string in case of failure | Master is the web service, slave or client is CAMART | UUID, WSDL |
| AXS web services | Web service for storing user actions information and retrieving protection information data | Master is AXCV (called by protection processor via PMS) | Event reporting |
| AXCV web services | Web service for certifying and verifying users and tools. | Master is protection processor (which performs the calls via PMS) | |

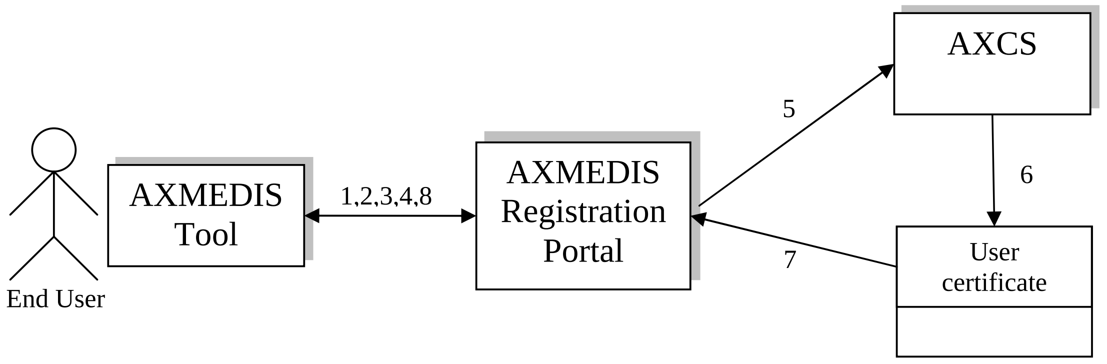
# 2 General use cases and scenarios (DSI, FUPF)

## 2.1 User registration through a distributor



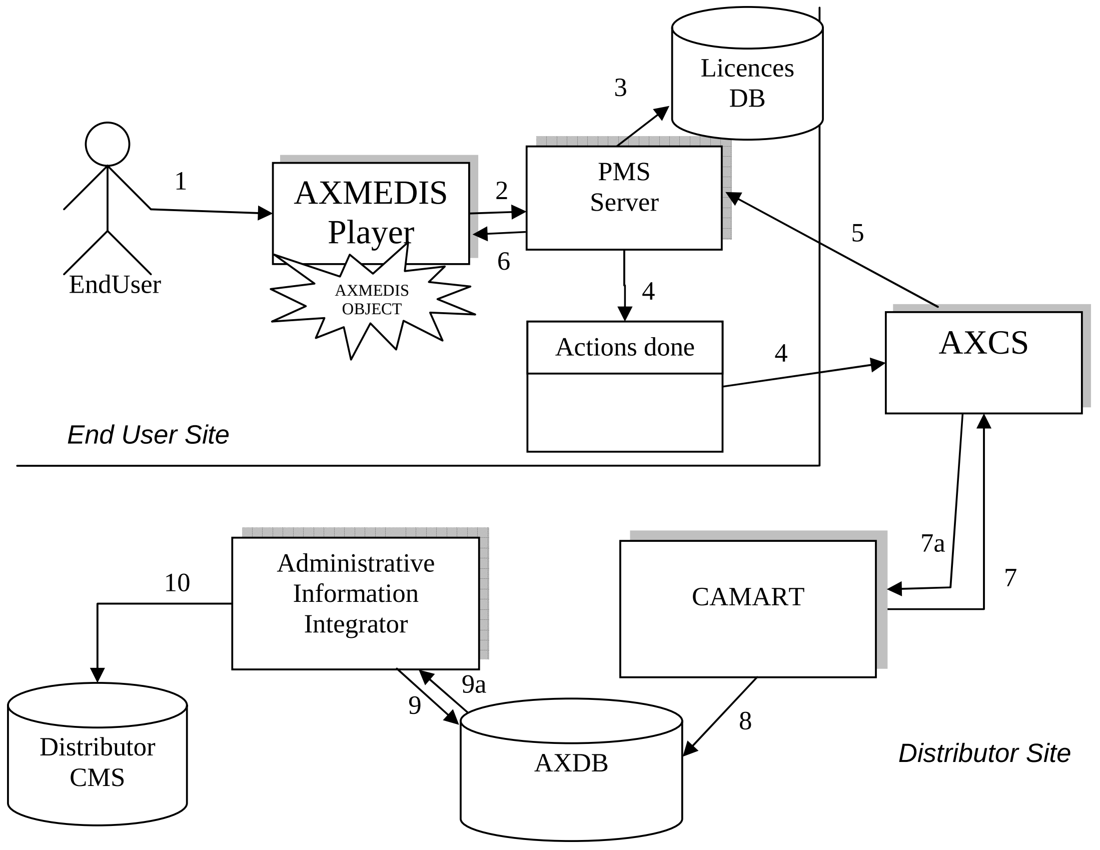
1. Upon first use the AXMEDIS tool (tailored by a distributor) checks for an user certificate. Assuming no certificate is found, the tool opens a web browser connecting to the web site of a distributor supporting registration of new users;
2. End user asks for registration in AXMEDIS, giving his personal data (name, city, address, phone number, etc.) to the distributor;
3. Distributor registers the new user using the user registration web service provided by AXCS: only data needed for registration are sent, such as email, nickname, password, nationality; no personal data are communicated to the AXCS; the information is stored in the AXCS database
4. AXCS generate a new AXUID, inserts it (with a new couple of private/public keys) in a specific certificate;
5. AXCS gives the certificate back to the distributor;
6. The distributor sends the certificate to the user via email;
7. The user import the certificate in the tool;

## 2.2 User registration through the AXMEDIS Registration Portal



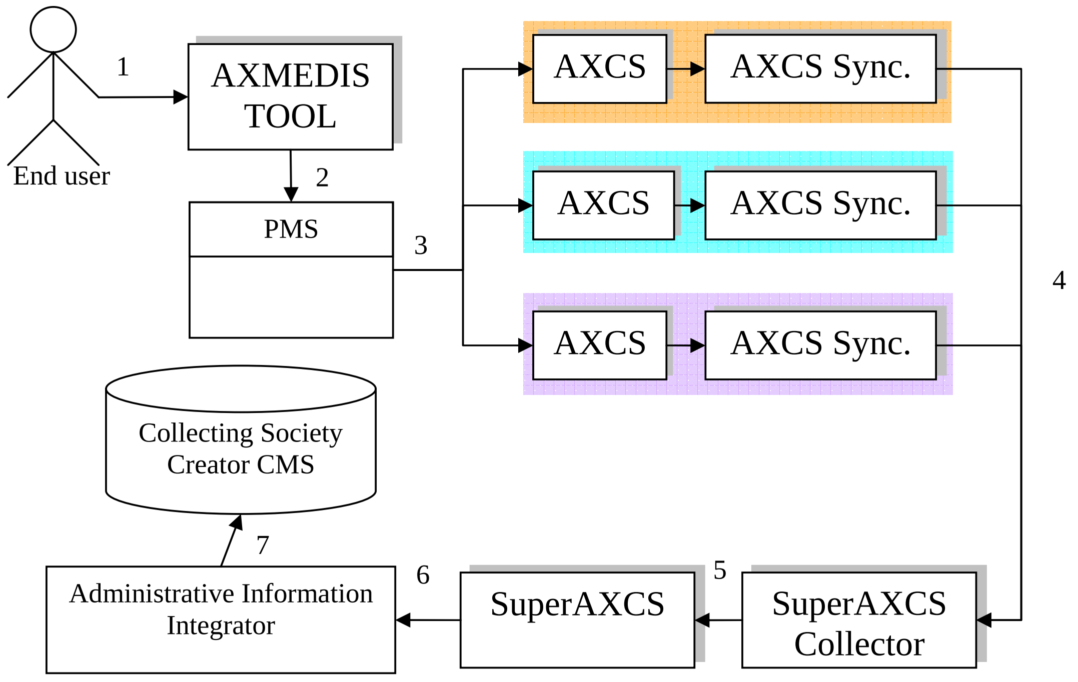
1. Upon first use the AXMEDIS tool checks for an user certificate. Assuming no certificate is found, the tool redirects the user to the AXMEDIS Portal
2. End user visits the web site of AXMEDIS Portal through the AXMEDIS Tool (not via web browser) and asks for registration in AXMEDIS, giving his personal data (name, city, address, phone number, etc.);
3. AXMEDIS Portal temporarily stores user personal data and sends a confirmation request via email to the user;
4. The user has to confirm his registration willing clicking on a link inserted in the received email; this brings him back to the AXMEDIS Portal to finally confirm the registration;
5. The AXMEDIS Portal registers the new user using the user registration web service provided by AXCS: only data needed for registration are sent, such as email, nickname, password, nationality; no personal data are communicated to the AXCS;
6. AXCS generate a new AXUID, inserts it (with a new couple of private/public keys) in a specific certificate;
7. AXCS gives the certificate back to the AXMEDIS Portal, which consolidates user personal data
8. AXMEDIS Portal returns the user certificate directly back to the tool, so that the certificate gets automatically imported.

## 2.3 Business Model (Distributor point of view)



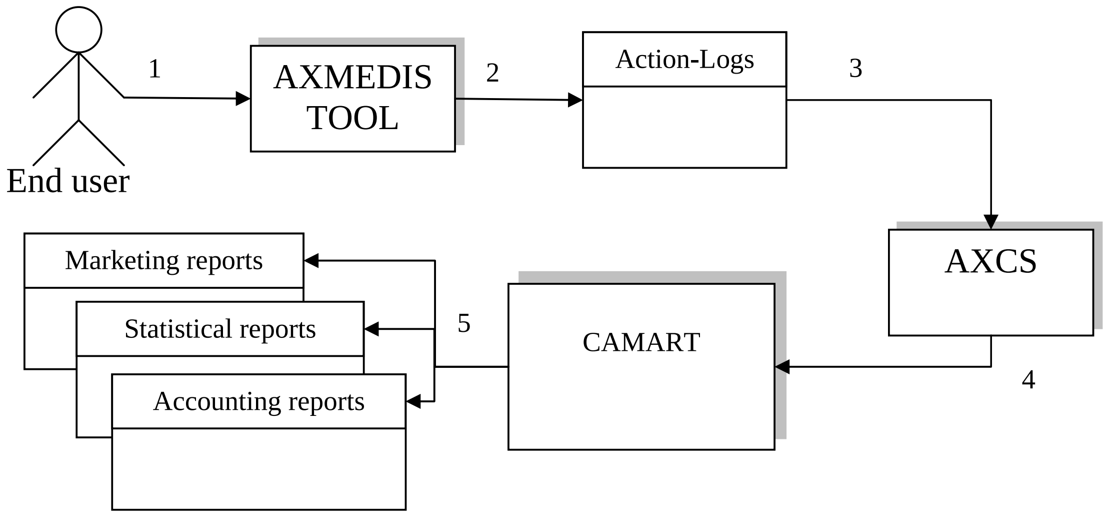
1. End User requests to perform an action on an AXMEDIS Protected Object
2. AXMEDIS Player asks PMS to perform an Action (assuming client has been already certified)
3. PMS checks in the LicenceDB if the Action is allowed (assuming OK)
4. PMS sends AXCS the action performed
5. AXCS gives back the key to access the content (if necessary)
6. PMS gives the grant to access the content and possibly the key to the AXMEDIS Player
7. CAMART retrieves from AXCS the actions performed by all the End Users on objects distributed by the distributor
8. CAMART stores the transactions in the AXDB
9. Adm. Integrator gets transactions performed from the DB
10. Administrative information are mapped into the Distributor CMS

## 2.4 Business Model (Collecting society or creator point of view)



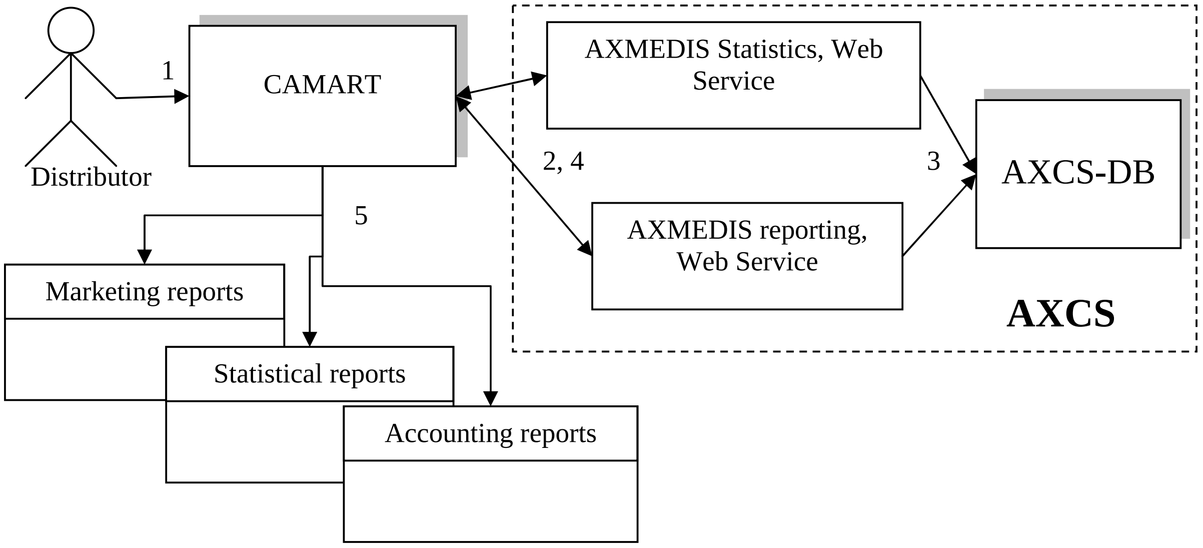
1. End User uses an AXMEDIS tool to operate on an AXMEDIS Protected Objects that are on different distribution channels
2. Protection Manager Support allow only authorized operations on the object
3. Objects are accessed on different channels and each AXCS stores its Action-Logs
4. Via the AXCS sync general information on objects or information that allow SuperAXCS to recover Action-Logs from the different AXCSs are transferred to the SuperAXCS Collector
5. SuperAXCS collects information
6. Administrative reports are created
7. Administrative Information Integrator transfer Action-Logs on CMS

## 2.5 Content Tracking and accounting (DSI, FUPF)



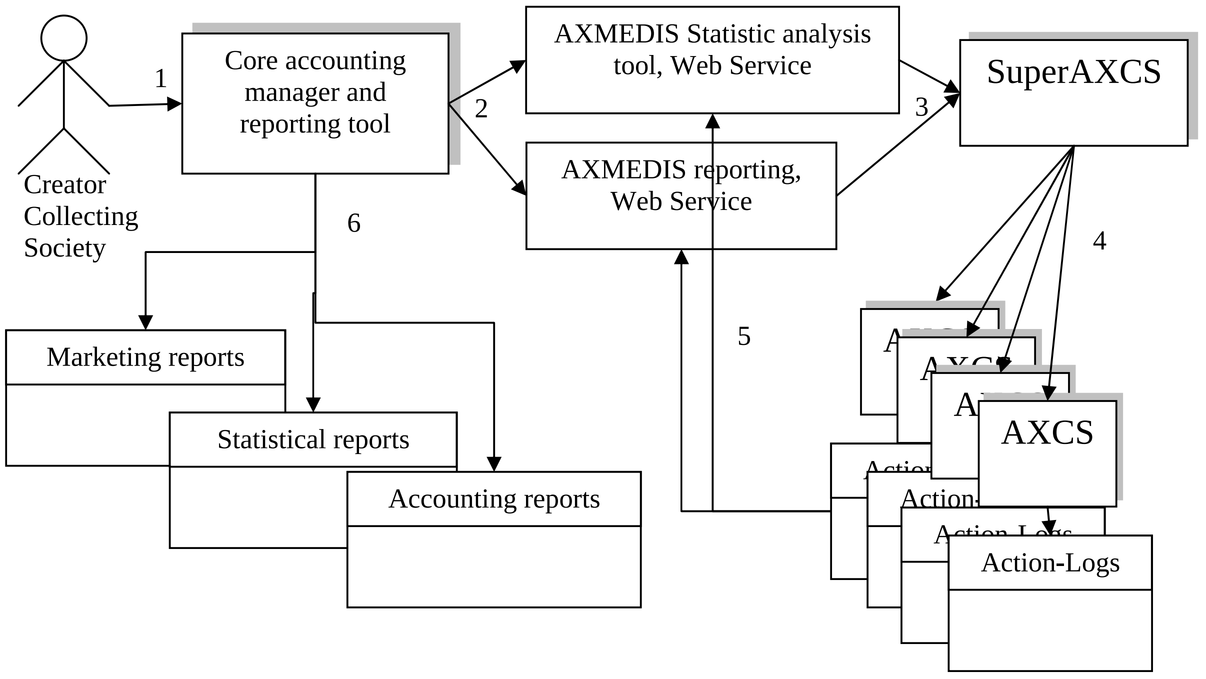
1. Distributor performs actions on objects
2. Action-Logs are generated (both on line and off line) reporting actions performed on objects
3. Action-Logs are stored by the AXCS
4. Core accounting manager and reporting tool extract information from AXCS allowing the generation of different reports type
5. Marketing reports, Statistical reports and accounting reports can be generated on demand

## 2.6 Single collection (single AXCS)



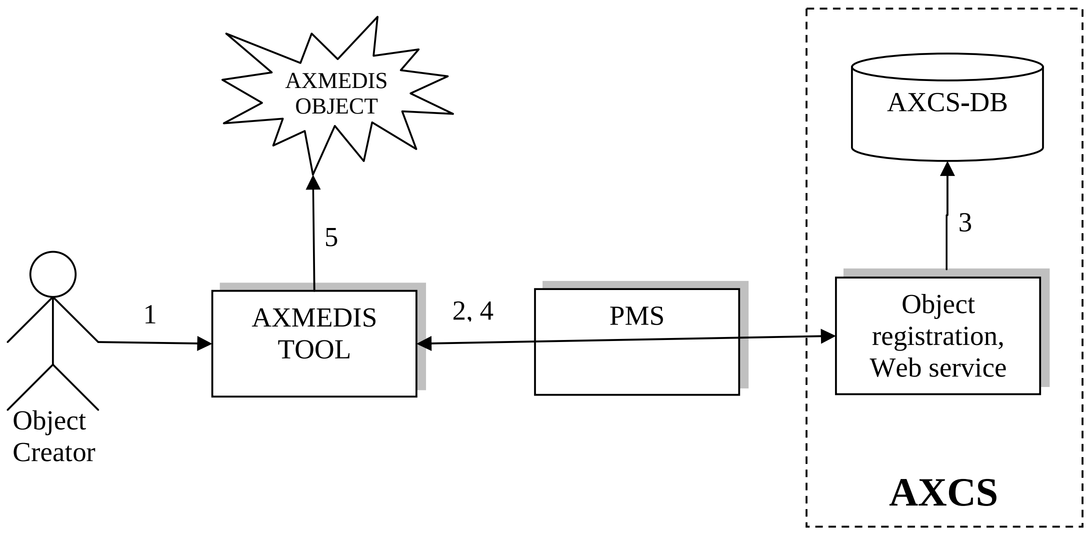
1. A Distributor wants to recover information on actions performed on the objects he has rights.
2. CAMART queries the correct tool for obtaining the Action-Logs in the correct form (anonymous or not, aggregated or not, etc)
3. AXMEDIS Statistic or reporting tools query AXCS-DB and extract the required Action-Logs
4. AXMEDIS Statistic or reporting tools communicate required Action Logs to CAMART to return results in the desired form
5. . Different reports are generated on the basis of the information collected.

## 2.7 Hierarchical collection of information from the several AXCS, the Super AXCS


Creator
Collecting
Society
1
Core accounting manager and reporting tool
2
AXMEDIS Statistic analysis tool, Web Service
AXMEDIS reporting, Web Service
3
SuperAXCS
4
AXCS
5
Action-Logs
6
Marketing reports
Statistical reports
Accounting reports

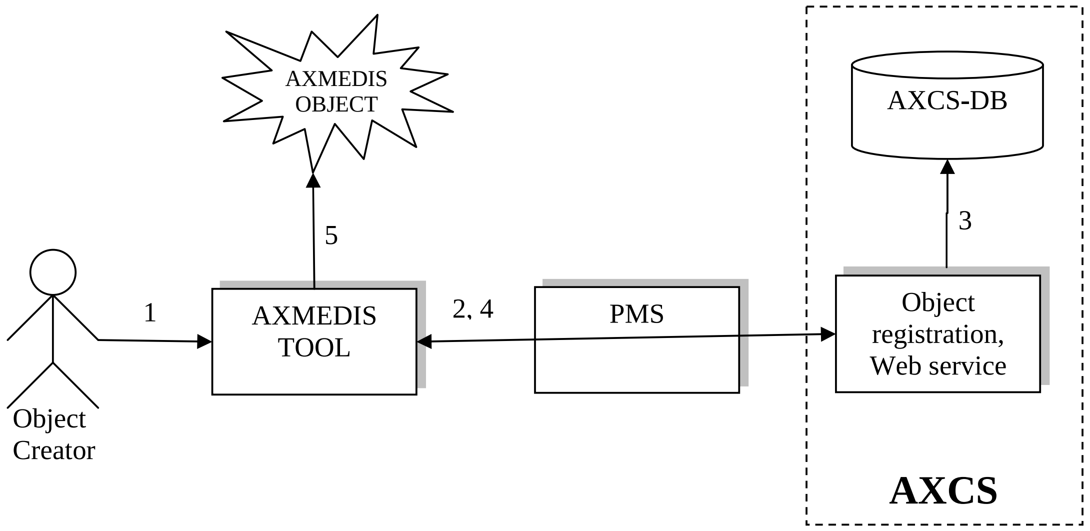
1. An Actor, that is collecting society or creator, wants to recover information on actions performed on the objects he has rights.
2. Core accounting manager and reporting tool query the correct tool for obtaining the Action-Logs in the correct form (anonymous or not, aggregated or not, etc)
3. AXMEDIS Statistic or reporting tools query the SuperAXCS
4. SuperAXCS recover information from the different AXCSs
5. The different AXCSs extract the required Action-Logs and communicate them to the tools that perform actions to return results in the desired form
6. Different reports are generated on the basis of the information collected.

## 2.8 Object ID generation



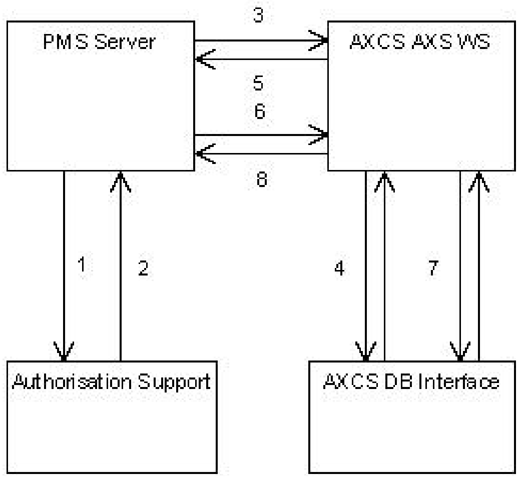
1. A Creator wants to create a new AXMEDIS Object.
2. The tool with whom the user is creating the object requests to the AXCS (via PMS) a new object ID
3. Object registration generates a new AXOID and stores it in the AXCS database (assuming success)
4. Object registration returns back to requestor the generated ID
5. A new AXMEDIS object is created with the assigned ID

## 2.9 Object metadata registration



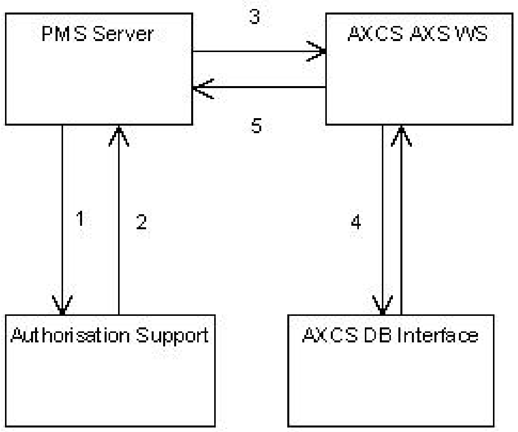
1. A Creator wants to register metadata about a new AXMEDIS object
2. The tool with whom the user is editing the object metadata requests to the AXCS (via PMS) for registration
3. Object registration stores received metadata in the AXCS database (assuming success)
4. Object registration returns back a successful value to the requestor

## 2.10 Successful authorisation scenario (FUPF)



1. PMS Server asks Authorisation Support to authorise an action
2. Authorisation Support determines that the user is granted to perform that action
3. PMS Server sends a supervisorInputData to AXCS-AXS to notify the successful authorisation
4. AXCS-AXS stores the supervisorInputData in AXCS database
5. AXCS-AXS notifies the successful storage of the received supervisorInputData
6. PMS Server requests AXCS-AXS the object keys or protection information
7. AXCS-AXS retrieves the keys or protection information from AXCS database
8. AXCS-AXS returns the keys or protection information

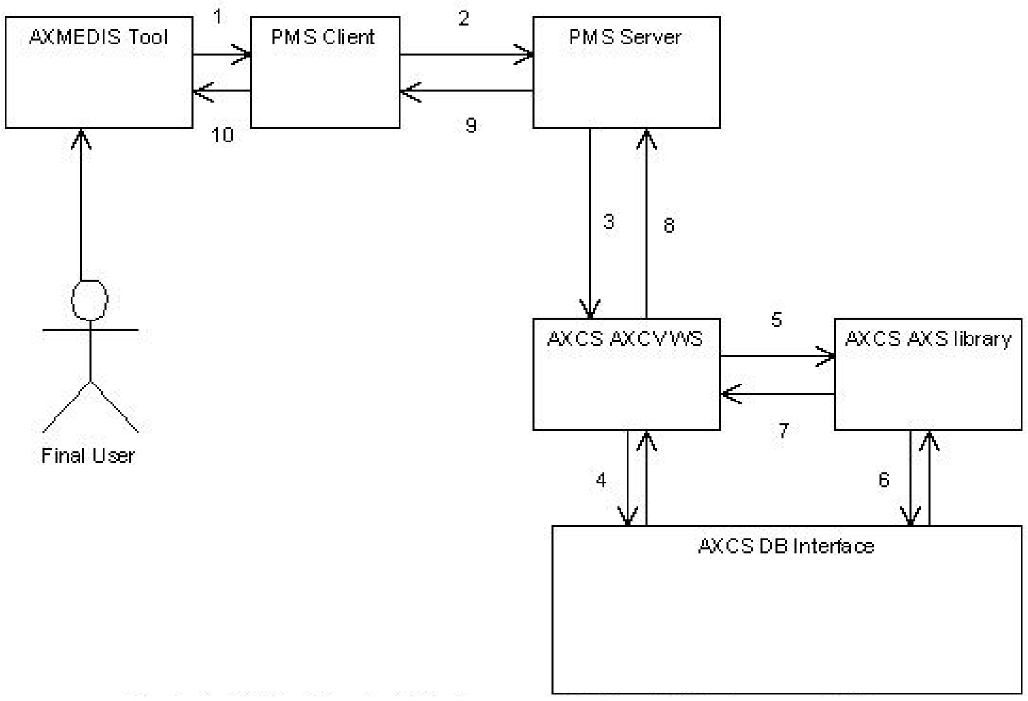
## 2.11 Unsuccessful authorisation scenario (FUPF)



1. PMS Server asks Authorisation Support to authorise an action
2. Authorisation Support determines that the user is not granted to perform that action
3. PMS Server sends a supervisorInputData to AXCS-AXS to notify the unsuccessful authorisation
4. AXCS-AXS stores the supervisorInputData in AXCS database
5. AXCS-AXS notifies the successful storage of the received supervisorInputData

## 2.12 Certification of Tool and User scenario (FUPF)

The scenario in the next figure represents the certification of a user and the corresponding AXMEDIS tool on a device, when the user is already registered in the system.

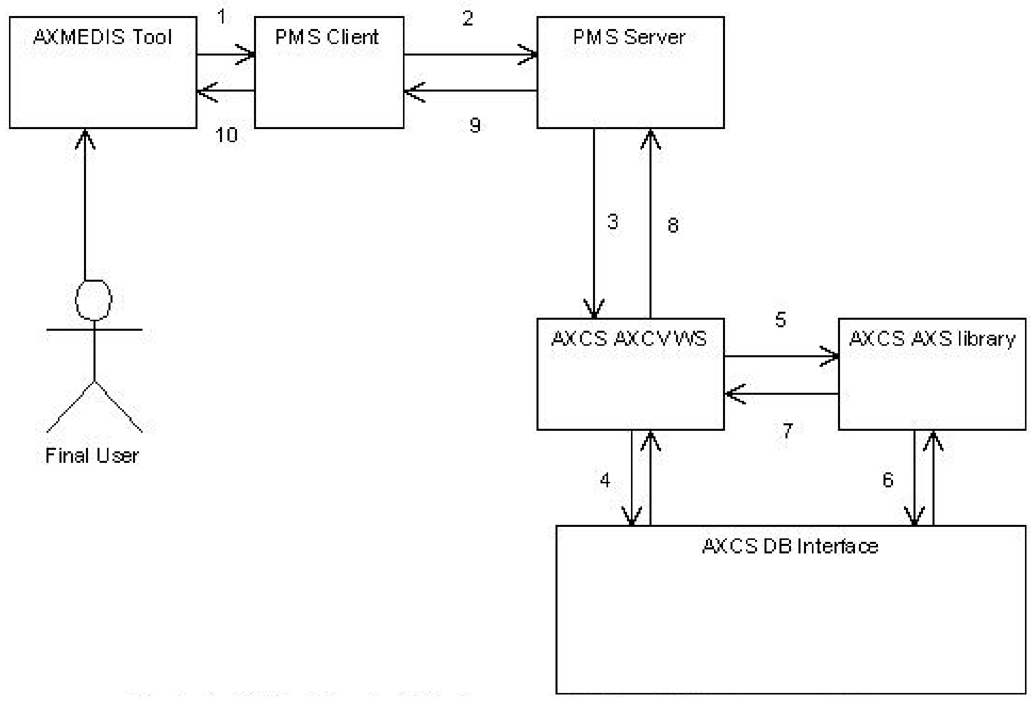


A Final User wants to use a tool for the first time

1. The tool uses PMS Client to send the PMS Server the following information: AXUID, AXRTID, tool Fingerprint, etc. in order to certify the tool.
2. PMS Client retrieves and adds domain information and contacts PMS Server
3. PMS Server contacts AXCS-AXCV
4. AXCS-AXCV checks the user status and tool integrity and determines that the tool was not already certified. AXCV generates AXTID, tool certificate and enabling code and creates a new entry in AXCS RegCert database.
5. AXCV sends AXS a SupervisorInputData (SID) which indicates that the user has certified a tool on a device.
6. AXS stores SID in the AXCS database
7. AXS confirms the storage
8. AXCV returns the result of the operation, and if the certification is successful it also returns AXTID, tool certificate and enabling code
9. PMS Server passes the received information to PMS Client
10. PMS Client gives the information to AXMEDIS Tool

## 2.13 Certification of Tool and User scenario (FUPF)

The scenario represents the relationship among the different components of AXCS when a user is not verified or tool is not certified. This scenario also shows the AXCS AXS functionality in this case.

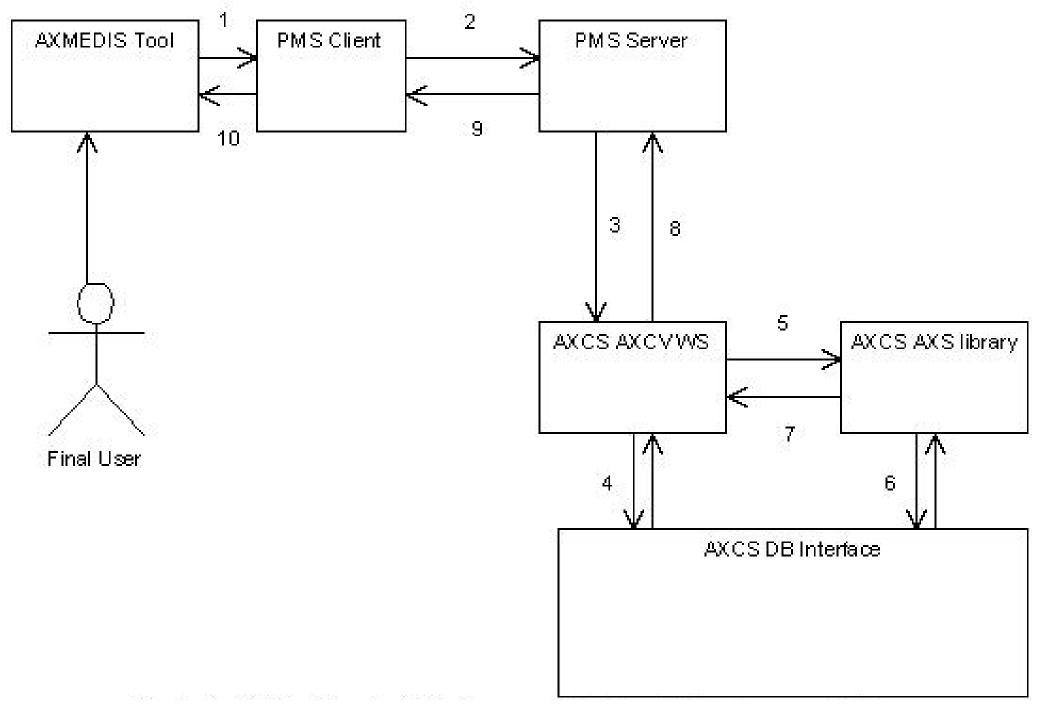


A Final User wants to use a tool for the first time

1. The tool uses PMS Client to send the PMS Server the following information: AXUID, AXRTID, tool Fingerprint (includes only software part), etc. in order to certify the tool
2. PMS Client retrieves and adds domain information and contacts PMS Server
3. PMS Server contacts AXCS-AXCV
4. AXCS-AXCV checks the user status and tool integrity and determines that user status is not correct or that the tool that the user is trying to certify does not match the data present in the database (the tool has been manipulated)
5. If the tool has been manipulated AXCV blocks the user and sends AXS a SupervisorInputData (SID) which indicates the reason why the user was blocked
6. AXS stores SID in the AXCS database
7. AXS confirms the storage
8. AXCV returns an error code that denotes the reason why the certification was not successful
9. PMS Server passes the received information to PMS Client
10. PMS Client gives the information to AXMEDIS Tool

## 2.14 Verification of AXMEDIS Users using AXMEDIS Tools on a Device during content consumption (FUPF)

The scenario represents the relation among the different components of AXCS during successful content consumption inside a Domain

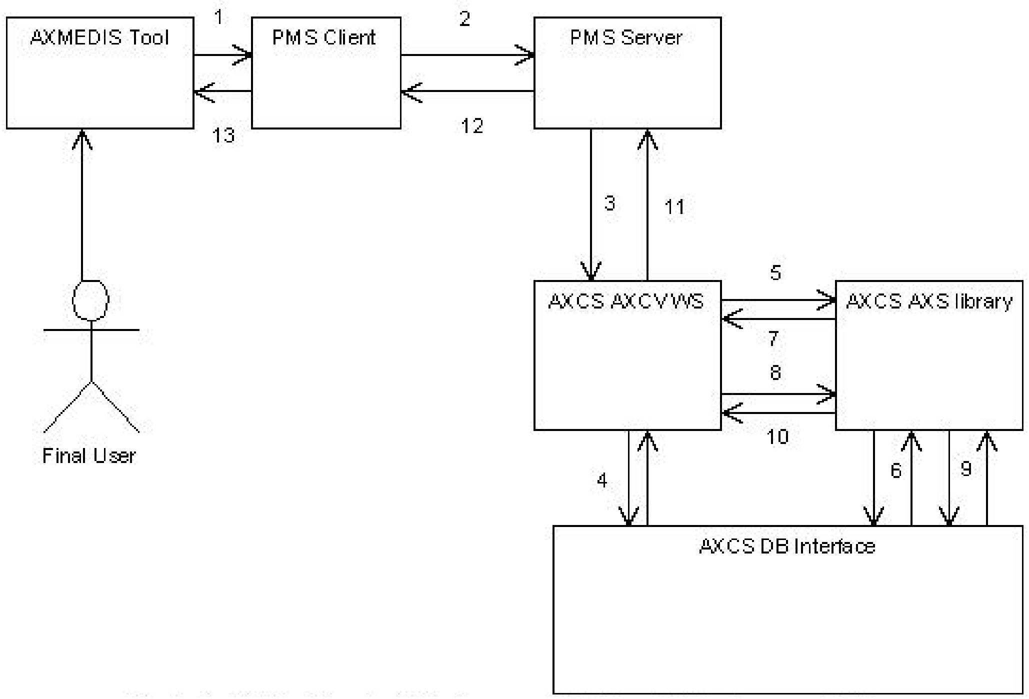


A Final User wants to consume an AXMEDIS object on a tool that has been already certified in AXMEDIS:

1. The tool uses PMS Client to send the PMS Server the following information: AXUID, AXTID, tool Fingerprint Digest (includes software and device hardware and installation information), list of performed actions to be resynchronized, etc. in order to verify user and tool status and integrity
2. PMS Client retrieves and adds domain information and contacts PMS Server
3. PMS Server contacts AXCS-AXCV
4. AXCS-AXCV checks the user status, tool certification in AXMEDIS and tool integrity
5. AXCS-AXCV sends AXCS-AXS the list of performed actions
6. AXCS-AXS checks that the list of actions is consistent in terms of fingerprint history
7. AXCS-AXS notifies AXCS-AXCV the successful storage
8. AXCV notifies PMS Server the successful verification
9. PMS Server passes the received information to PMS Client
10. PMS Client gives the information to AXMEDIS Tool

## 2.15 Verification of AXMEDIS Users using AXMEDIS Tools on a Device during content consumption (FUPF)

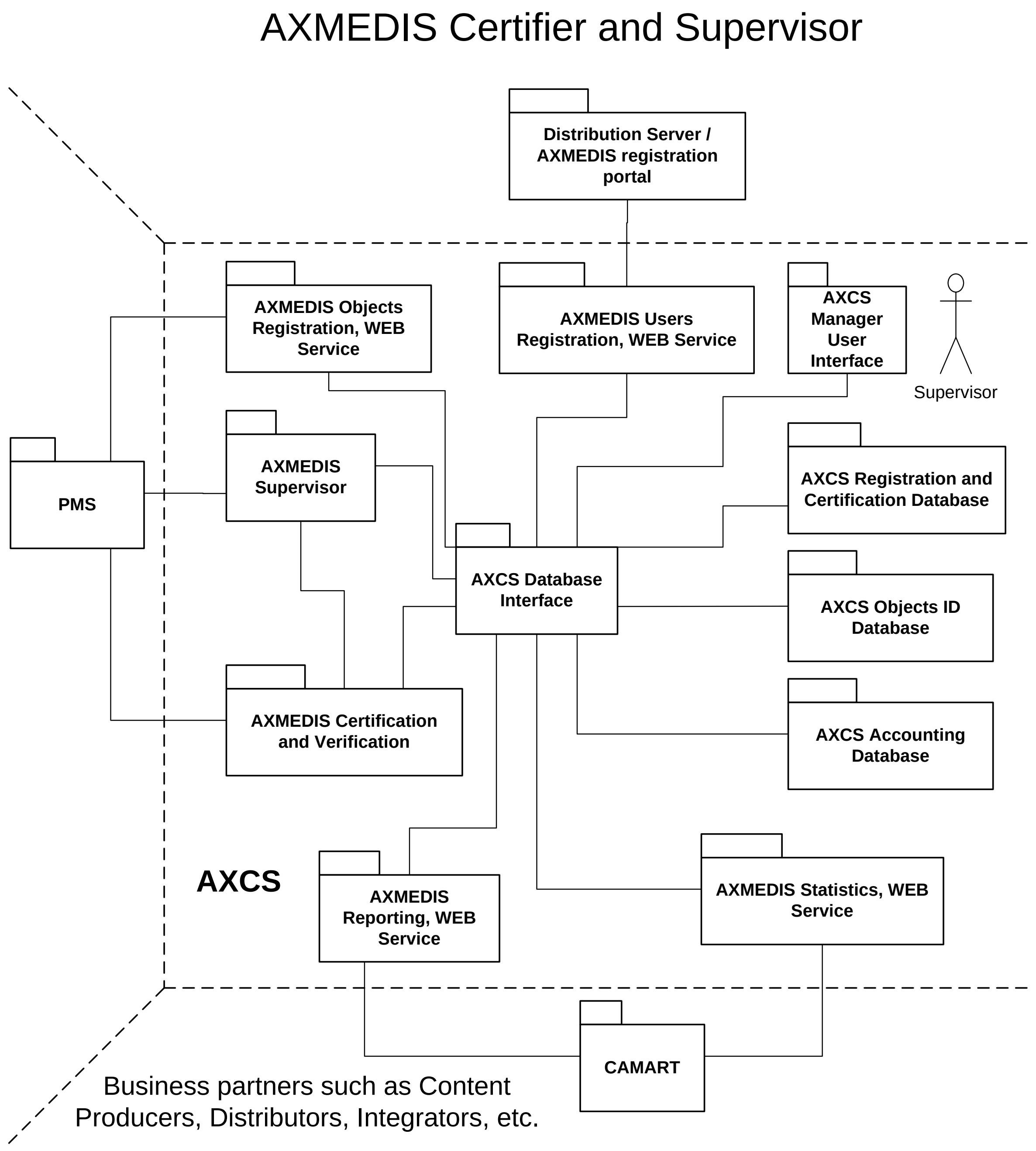
The scenario represents the relationship among the different components of AXCS during unsuccessful content consumption due to a mismatch in the verification of the history of performed actions.



A Final User wants to consume an AXMEDIS object on a tool that has been already certified in AXMEDIS:

1. The tool uses PMS Client to send the PMS Server the following information: AXUID, AXTID, tool Fingerprint Digest (includes software and device hardware and installation information), list of performed actions to be resynchronized, etc. in order to verify user and tool status and integrity
2. PMS Client retrieves and adds domain information and contacts PMS Server
3. PMS Server contacts AXCS-AXCV
4. AXCS-AXCV checks the user status, tool certification in AXMEDIS and tool integrity
5. AXCS-AXCV sends AXCS-AXS the list of performed actions
6. AXCS-AXS determines that the list of performed actions is not consistent in terms of fingerprint history, marks these actionLogs setting to “1” the histVerSuccess field and to “invalid” instantLastFPPA field of the received actionLogs and stores them in the AXCS database
7. AXCS-AXS notifies AXCS-AXCV the unsuccessful verification of the history of performed actions
8. AXCV blocks the certified tool and the user and sends a SupervisorInputData to AXS that explains the reasons why
9. AXS stores SID in the AXCS database
10. AXS confirms the storage
11. AXCV notifies PMS Server the unsuccessful verification
12. PMS Server passes the received information to PMS Client
13. PMS Client gives the information to AXMEDIS Tool

# 3 General architecture and relationships among the modules produced


AXMEDIS Certifier and Supervisor
Distribution Server / AXMEDIS registration portal
AXMEDIS Objects Registration, WEB Service
AXMEDIS Users Registration, WEB Service
AXCS Manager User Interface
Supervisor
PMS
AXMEDIS Supervisor
AXCS Registration and Certification Database
AXCS Database Interface
AXCS Objects ID Database
AXMEDIS Certification and Verification
AXCS Accounting Database
AXCS
AXMEDIS Reporting, WEB Service
AXMEDIS Statistics, WEB Service
CAMART
Business partners such as Content Producers, Distributors, Integrators, etc.

AXMEDIS Certifier and Supervisor is the AXMEDIS certification authority that provides services for Content Providers and Distributors and verify the correctness of the Clients (as "Clients" are intended also software agents, not only physics or legal people).
AXMEDIS Certifier and Supervisor database structure is designed considering the distribution of services provided with the aim of scalable architecture capable of supporting a huge amount of transactions per second. These transactions can be of various kinds:

- requests of key and/or protection information,
- requests of verification,
- requests of logs,
- Registrations, etc…

The architecture of AXMEDIS Certifier and Supervisor has to be flexible enough to support centralised Certification and Supervision as well as distributed. In the centralised version only one AXMEDIS Certifier and Supervisor is set up for the whole network, in the other case can be present more than one Certifier and Supervisor (one for each distribution channel). They could be connected as a hierarchical or a peer-to-peer structure or stand alone, limiting in this case the navigation of content.

The architecture of the AXMEDIS Certifier and Supervisor has to be scalable and the internal services should be well separable to cope with large traffic for the certification and supervision and to allow the decentralisation of some of the services in an easy and reconfigurable manner.

## 4 AXMEDIS Certification and Verification, AXCV (FUPF)

| Module/Tool Profile | | |
|---|---|---|
| **AXMEDIS Certification and Certification, AXCV** | | |
| Responsible Name | Víctor Torres | |
| Responsible Partner | FUPF | |
| Status (proposed/approved) | Approved | |
| Implemented/not implemented | Implemented | |
| Status of the implementation | First version available | |
| Executable or Library/module (Support) | Library and Web service | |
| Single Thread or Multithread | Multithread | |
| Language of Development | Java | |
| Platforms supported | | |
| Reference to the AXFW location of the source code demonstrator | https://cvs.axmedis.org/repos/Framework/source/axcs-axcv | |
| Reference to the AXFW location of the demonstrator executable tool for internal download | https://cvs.axmedis.org/repos/Framework/bin/axcs-axcv/axcv.jar<br>https://cvs.axmedis.org/repos/WebServices/axcs-axcv/bin/wsaxcv.jar | |
| 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 | http://193.145.45.173:8080/axis/AXCV<br>https://193.145.45.173:8443/axis/AXCV<br><br>http://flauto.dsi.unifi.it:8080/axis/services/AXCV | |
| Test cases (present/absent) | Present | |
| Test cases location | http://cvs.axmedis.org/repos/Framework/doc/test/axcs-axcv<br>http://cvs.axmedis.org/repos/WebServices/axcs-axcv/doc/test | |
| Usage of the AXMEDIS configuration manager (yes/no) | No | |
| Usage of the AXMEDIS Error Manager (yes/no) | No | |
| Major Problems not solved | | |
| Major pending requirements | | |
| | | |
| Interfaces API with other tools, named as | Name of the communicating tools References to other major components needed | Communication model and format (protected or not, etc.) |
| PMS Server (AXCS proxy) | | |
| AXMEDIS Supervisor | | |
| AXCS Database Interface | | |
| | | |
| 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 | | |
| AXCSRegCert | | |
| | | |
| | | |
| | | |
| | | |
| 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 |
| Bouncy Castle | bcprov-jdk15-128.jar | http://www.bouncycastle.org/licence.html |
| Doomdark | jug.jar | LGPL |
| Unrestricted policy files for the Sun JCE: | local_policy.jar<br>US_export_policy.jar | |
| MySql | mysql-connector-java-3.1.8-bin.jar | |
| | | |
| | | |

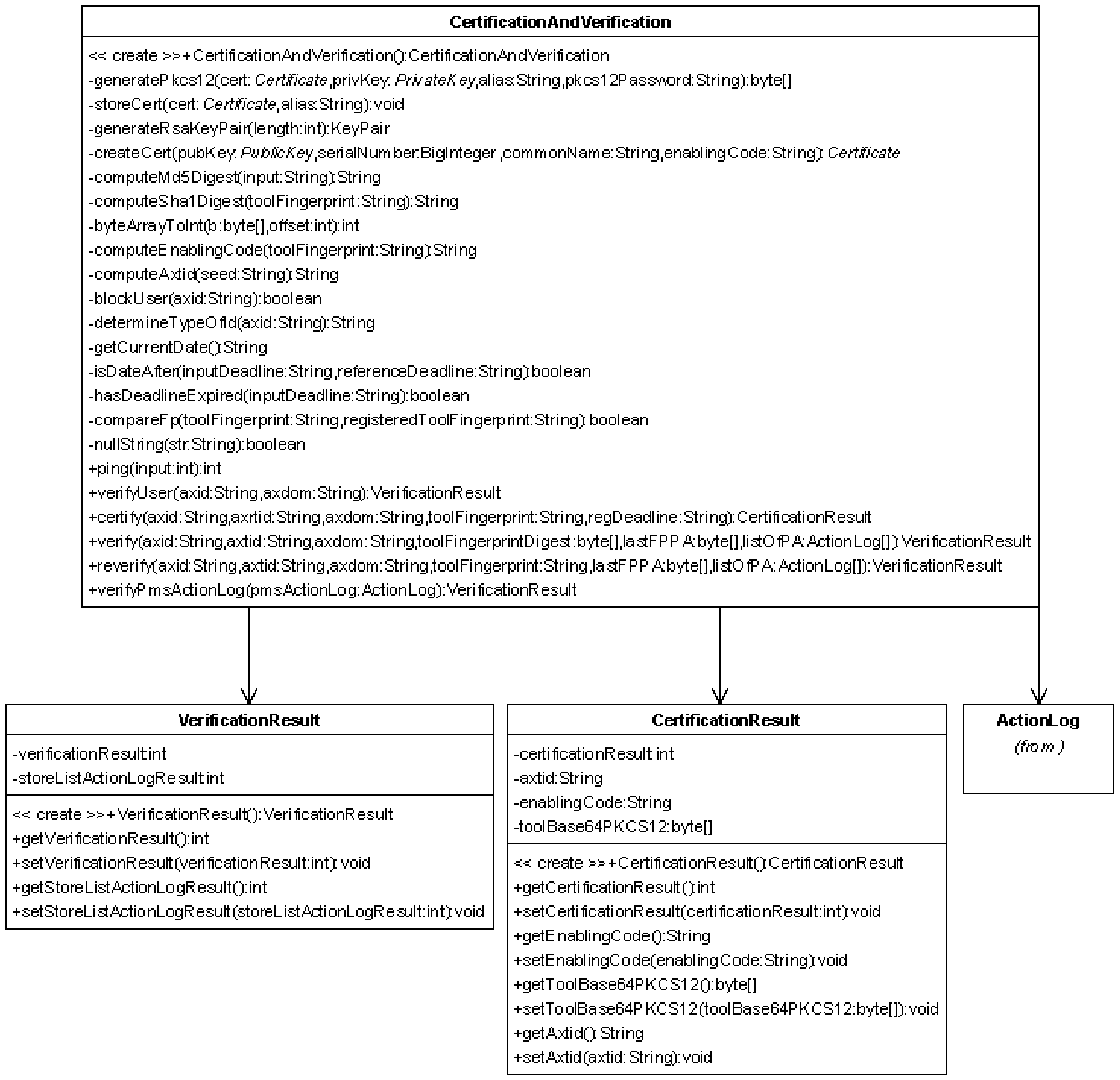
## 4.1 General Description of the Module

| AXCV | |
|---|---|
| ***Methods*** | ***Description*** |
| verifyUser | This method is called by the AXCS Proxy integrated in PMS Client, which 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 the AXCS Proxy integrated in PMS Client, which 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 AXCS. |
| verify | This method is called by the AXCS Proxy integrated in PMS Client, which 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: – |

| | |
|---|---|
| | 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. |
| verifyPmsActionLog | This method is used to store a single ActionLog after the authorisation of a user in PMS Server, if AXCS is online. The difference between this method and verify method is that in this case, it is PMS Server who partially fills and sends the single input ActionLog. Moreover, PMS does not send the lastFPPA of the client, because it is not known. Therefore, AXS will calculate the new lastFPPA and store it in AXCS database without verifying it. We can suppose that, as AXCV verify method is called just before performing the authorisation, the operation history will be already verified. |
| ping | This method is used by PMS to determine if AXCV is online. |

## 4.2 Module Design in terms of Classes

The following figure shows the UML diagram of this module, together with the definition of its operations. ActionLog class is defined in AXMEDIS Supervisor.


CertificationAndVerification
<< create >>+CertificationAndVerification():CertificationAndVerification
-generatePkcs12(cert: Certificate,privKey: PrivateKey,alias:String,pkcs12Password:String):byte[]
-storeCert(cert: Certificate,alias:String):void
-generateRsaKeyPair(length:int):KeyPair
-createCert(pubKey: PublicKey,serialNumber:BigInteger ,commonName:String,enablingCode:String): Certificate
-computeMd5Digest(input:String):String
-computeSha1Digest(toolFingerprint:String):String
-byteArrayToInt(b:byte[],offset:int):int
-computeEnablingCode(toolFingerprint:String):String
-computeAxtid(seed:String):String
-blockUser(axid:String):boolean
-determineTypeOfId(axid:String):String
-getCurrentDate():String
-isDateAfter(inputDeadline:String,referenceDeadline:String):boolean
-hasDeadlineExpired(inputDeadline:String):boolean
-compareFp(toolFingerprint:String,registeredToolFingerprint:String): boolean
-nullString(str:String):boolean
+ping(input:int):int
+verifyUser(axid:String,axdom:String):VerificationResult
+certify(axid:String,axrtid:String,axdom:String,toolFingerprint:String,regDeadline:String):CertificationResult
+verify(axid:String,axtid:String,axdom: String,toolFingerprintDigest:byte[],lastFPPA:byte[],listOfPA:ActionLog[]):VerificationResult
+reverify(axid:String,axtid:String,axdom:String,toolFingerprint:String,lastFPPA:byte[],listOfPA: ActionLog[]):VerificationResult
+verifyPmsActionLog(pmsActionLog:ActionLog):VerificationResult
VerificationResult
-verificationResult:int
-storeListActionLogResult:int
<< create >>+VerificationResult():VerificationResult
+getVerificationResult():int
+setVerificationResult(verificationResult:int):void
+getStoreListActionLogResult():int
+setStoreListActionLogResult(storeListActionLogResult:int):void
CertificationResult
-certificationResult:int
-axtid:String
-enablingCode:String
-toolBase64PKCS12:byte[]
<< create >>+CertificationResult():CertificationResult
+getCertificationResult():int
+setCertificationResult(certificationResult:int):void
+getEnablingCode():String
+setEnablingCode(enablingCode:String):void
+getToolBase64PKCS12():byte[]
+setToolBase64PKCS12(toolBase64PKCS12:byte[]):void
+getAxtid(): String
+setAxtid(axtid: String):void
ActionLog
(from )

## 4.3 Technical and Installation information

The following installation information refers to the web service of AXCV, which uses AXCV and other libraries:

| | |
|---|---|
| References to other major components needed | AXMEDIS AXCS AXCV library<br>AXMEDIS AXCS AXS library<br>AXCS Database Interface library |
| Problems not solved | None |
| Configuration and execution context | <u>How to deploy/undeploy the web service</u><br><br>To deploy/undeploy the AXCV AXIS web service in your Tomcat server you can use the org.apache.axis.client.AdminClient class using the provided deploy.wsdd or undeploy.wsdd files as parameters. See the example below.<br><br><u>Classes needed to use the software</u><br><br>- Client side: wsaxcv.jar library, which contains client side classes:<br>    - CertificationAndVerification<br>    - CertificationAndVerificationService<br>    - CertificationAndVerificationServiceLocator<br>    - AXCVSoapBindingStub<br>    - CertificationResult<br>    - VerificationResult<br><br>- Server side: wsaxcv.jar library, which contains server side classes:<br>    - CertificationAndVerification<br>    - AXCVSoapBindingImpl<br>    - AXCVSoapBindingSkeleton<br>    - CertificationResult<br>    - VerificationResult<br><br><u>Libraries needed to run the software</u><br><br>- Client side<br>    - bcprov-jdk15-128.jar (only for testing)<br>    - mysql-connector-java-3.1.8-bin.jar (only for testing)<br>    - junit.jar (only for testing)<br>    - AXIS related libraries:<br>        - mail.jar<br>        - activation.jar<br>        - axis.jar<br>        - jaxrpc.jar<br>        - saaj.jar<br>        - commons-logging-1.0.4.jar<br>        - commons-discovery-0.2.jar<br>        - wsdl4j-1.5.1.jar<br>    - other dependencies<br>        - AXS library<br>        - AXCS Database Interface library (only for testing) |

- Server side
    - bcprov-jdk15-128.jar
    - mysql-connector-java-3.1.8-bin.jar
    - jug.jar
    - AXIS related libraries
        - mail.jar
        - activation.jar
        - axis.jar
        - jaxrpc.jar
        - saaj.jar
        - commons-logging-1.0.4.jar
        - commons-discovery-0.2.jar
        - wsdl4j-1.5.1.jar
    - other dependencies
        - AXCV library
        - AXS library
        - AXCS Database Interface library

Files needed for the deployment of AXCV library in the Server side

The following files must be included anywhere in the local classpath where the software is executed:

- AxcsCAPkcs12.p12: This key store contains the AXCV certificate and private key used to generate the tool certificates.
- axcvToolCertStore.p12: This key store contains the tool certificates that have been generated by the AXCV. It does not contain the associated private keys.
- axcv.properties: This java properties file contains the information necessary for the initialisation of the AXCV module. It can be customised to change the names and passwords of the corresponding files and key stores that the AXCV needs to access. RegCertDSN, AxcsDbUser and AxcsDbPassword parameters are only used if the axcsdb.ini file from DSI AXCS database is not found in the system.
- nextSerial.txt: This file stores the next serial to be used for the tool certificate generation. Every time a tool certificate is generated, it is incremented by 1.

The following files must substitute the original files available in the <java-home>\lib\security directory in order to be able to use the maximum strength in the cryptographic algorithms:

- Unrestricted policy files for the Sun JCE: local_policy.jar, US_export_policy.jar. If these jar files are not installed, then the pkcs12 structures generated when certifying tools cannot be encrypted with the whole AXUID password, and a the 8 first characters of the AXUID are then used.

Note: <java-home> refers to the directory where the J2SE Runtime Environment (JRE) was installed. It is determined based on whether you are running JCE on a JRE with or without the JDK installed. The JDK contains the JRE, but at a different level in the file hierarchy. For example, if the JDK is installed in /home/user1/jdk1.5.0 on Solaris or in C:\jdk1.5.0 on Win32, then <java-home> is

- /home/user1/jdk1.5.0/jre [Solaris]
- C:\jdk1.5.0\jre [Win32]

If on the other hand the JRE is installed in /home/user1/jre1.5.0 on Solaris or in C:\jre1.5.0 on Win32, and the JDK is not installed, then <java-home> is

- /home/user1/jre1.5.0 [Solaris]
- C:\jre1.5.0 [Win32]

<u>Example of deployment in Linux server</u>

```
cd ($AXIS_HOME)
java -cp
"($AXIS_HOME)/WEB-INF/lib/wsaxcv.jar:
($AXIS_HOME)/WEB-INF/lib/axcv.jar:
($AXIS_HOME)/WEB-INF/lib/axs.jar:
($AXIS_HOME)/WEB-INF/lib/axcs-db-interface.jar:
($AXIS_HOME)/WEB-INF/lib/mysql-connector-java-3.1.8-bin.jar:
($AXIS_HOME)/WEB-INF/lib/bcprov-jdk15-128.jar:
($AXIS_HOME)/WEB-INF/lib/jug.jar:
($AXIS_HOME)/WEB-INF/lib/activation.jar:
($AXIS_HOME)/WEB-INF/lib/mail.jar:
($AXIS_HOME)/WEB-INF/lib/xerces.jar:
($AXIS_HOME)/WEB-INF/lib/axis.jar:
($AXIS_HOME)/WEB-INF/lib/axis-ant.jar:
($AXIS_HOME)/WEB-INF/lib/jaxrpc.jar:
($AXIS_HOME)/WEB-INF/lib/saaj.jar:
($AXIS_HOME)/WEB-INF/lib/commons-logging-1.0.4.jar:
($AXIS_HOME)/WEB-INF/lib/commons-discovery-0.2.jar:
($AXIS_HOME)/WEB-INF/lib/wsdl4j-1.5.1.jar:
($AXIS_HOME)/WEB-INF/lib/log4j-1.2.8.jar:
($AXIS_HOME)/WEB-INF/classes/"
org.apache.axis.client.AdminClient -p80 "($AXIS_HOME)/WEB-
INF/($WSDD_PATH)/deploy.wsdd"
```

In the above example "($AXIS_HOME)/WEB-INF/lib/" contains all necessary libraries and dependencies, and "($AXIS_HOME)/WEB-INF/classes/" contains the files needed by dependent libraries:
- AXCV library needed files:
    - AxcsCAPkcs12.p12
    - axcvToolCertStore.p12
    - nextSerial.txt
    - axcv.properties
- AXS library needed files (see AXMEDIS Supervisor, AXS):
    - axs.properties

<u>How to configure the server to have secure AXIS Web Services using Tomcat 5.5</u>

You can configure your Tomcat server so that it provides AXIS web services over a secure channel (SSL).

First of all you must configure server.xml file in your Tomcat server:

- Uncomment the code:

```
<-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->
<!--
<Connector
        port="8443" minProcessors="5" maxProcessors="75"
        enableLookups="true" disableUploadTimeout="true"
        acceptCount="100" debug="0" scheme="https" secure="true";
        clientAuth="false" sslProtocol="TLS"/>
-->
```

- Customise it to request for client authentication:

```
 <Connector
        port="8443" maxHttpHeaderSize="8192" maxThreads="150"
        minSpareThreads="25" maxSpareThreads="75"
        enableLookups="false" disableUploadTimeout="true"
        acceptCount="100" scheme="https" secure="true"
        clientAuth="true" sslProtocol="TLS"
        keystoreFile="C:\path\to\keystore\AxcsCAPkcs12.p12"
        keystorePass="AXCSpwd" keystoreType="PKCS12"
        truststoreFile="C:\path\to\trustore\AxcsCAPkcs12.p12"
        truststorePass="AXCSpwd" truststoreType="PKCS12"
/>
```

Where,

- keystoreFile: JKS or PKCS12 keystore file that contains the server certificate and private key used to authenticate the server.

- truststoreFile: JKS or PKCS12 keystore file that contains the certificates of the CAs that the server will accept when a client presents a certificate signed by one of them.

When using a self-signed or CA certificate for the server, both keystore files can be the same, if necessary.

For more details on how to configure the options in server.xml file refer to http://tomcat.apache.org/tomcat-5.5-doc/ssl-howto.html.

How to configure the client to have secure AXIS Web Services

You can configure your AXIS client so that it connects to a secure web service using a secure channel (SSL). You can also configure it to perform client authentication by presenting his own certificate

In the Java code of your client you can set, for example:

```
//add BouncyCastle JCE provider
Security.addProvider(new BouncyCastleProvider());

//Secure Web Service initialisation
CertificationAndVerificationServiceLocator    service    =    new
CertificationAndVerificationServiceLocator();
java.net.URL              secureURL              =              new
java.net.URL("https://localhost:8443/axis/services/AXCV");
CertificationAndVerification axcv = service.getAXCV(secureURL);

//System settings to present a client certificate (client authentication)
System.setProperty("javax.net.ssl.keyStore", “C:\\path\\to\\keystore.p12");
System.setProperty("javax.net.ssl.keyStorePassword", "password");
System.setProperty("javax.net.ssl.keyStoreType", "pkcs12");
System.setProperty("javax.net.ssl.keyStoreProvider", "BC");

//System settings to trust on the server certificate (server authentication)
System.setProperty("javax.net.ssl.trustStore",
“C:\\path\\to\\truststore.p12");
System.setProperty("javax.net.ssl.trustStorePassword", "password");
System.setProperty("javax.net.ssl.trustStoreType", "pkcs12");
System.setProperty("javax.net.ssl.trustStoreProvider", "BC");

//Other System settings
System.setProperty("java.protocol.handler.pkgs",
"com.sun.net.ssl.internal.www.protocol");
System.setProperty("javax.net.debug","all");
System.setProperty("java.security.debug","all");
Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
```

Where,

- keyStore: keystore file that contains the client certificate and private key used to authenticate the client.

- trustStore: keystore file that contains the certificates of the CAs that the client will trust on, when a server presents a certificate signed by one of them.

You can choose between any different provider available in your machine and also customise the keystore types and SSL provider.

## 4.4 Examples of usage

A Junit-based Test is provided to perform a full test of AXCV web service functionalities. It shows how the AXS web service can be invoked to obtain many different results. It is available at: http://cvs.axmedis.org/repos/WebServices/axcs-axcv/doc/test/wsAXCVAXSTest.java

A simple test is provided to show how a simple invocation of the method can be performed. It is available at: http://cvs.axmedis.org/repos/WebServices/axcs-axcv/doc/test/wsAXCVAXSTest_simple.java

## 4.5 Integration and compilation issues

You can deploy the provided web service libraries on any device, which supports AXIS and Java. For other Web Services platforms you must generate the corresponding classes and files from the AXCV web service wsdl file.

## 4.6 Configuration Parameters

The following parameters can be configured in axcv.properties file

| Config parameter | Possible values |
|---|---|

| | |
|---|---|
| axcsCertStore | PKCS12 store file name where AXCV certificate and private key are stored<br>E.g: AxcsCAPkcs12.p12 |
| axcsCertStorePasswd | Password for PKCS12 store where AXCV certificate and private key are stored<br>E.g.: password |
| axcvToolCertStore | PKCS12 store file name where the generated tool certificates are stored (private key not stored)<br>E.g.: axcvToolCertStore.p12 |
| axcvToolCertStorePasswd | Password for PKCS12 store where the generated tool certificates are stored<br>E.g.: password |
| nextSerial | File name where serial number for next tool Certificate is stored<br>E.g.: nextSerial.txt |
| RegCertDSN | AXCS Registration and Certification database DSN. This parameter will be only used if the axcsdb.ini file from DSI database is not found in the system.<br>Possible values:<br>If DSI database is present in the system: foo<br>If DSI database is not present in the system: jdbc:mysql://193.145.45.173/axcsregcert |
| AxcsDbUser | AXCS Database User. This parameter will be only used if the axcsdb.ini file from DSI database is not found in the system.<br>Possible values:<br>If DSI database is present in the system: foo<br>If DSI database is not present in the system: databaseUser |
| AxcsDbPassword | AXCS Database Password. This parameter will be only used if the axcsdb.ini file from DSI database is not found in the system.<br>Possible values:<br>If DSI database is present in the system: foo<br>If DSI database is not present in the system: databasePassword |

## 4.7 Formal description of AXCV algorithms

| **AXCV library and Web Service** | |
|---|---|
| Method | verifyUser |
| Description | This method is called by the AXCS Proxy integrated in PMS Client, which 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 or B2BUser (AXUID)<br>xsd:string **axdom**: AXMEDIS domain of certified user (if any) |
| Output parameters | VerificationResult complex type formed by sequence of:<br>xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration:<br>0: Verification OK<br>-1: invalid AXID<br>-2: user is not registered<br>-3: user is blocked<br>-4: user domain mismatch<br>-5: user registration deadline expired<br><br>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). |

| **AXCV library and Web Service** | |
|---|---|
| Method | certify |
| Description | This method is called by the AXCS Proxy integrated in PMS Client, which 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 AXCS. |
| Input parameters | xsd:string axid: identifier of the AXMEDIS final user or B2BUser (AXUID)<br>xsd:string **axrtid**: identifier of the registered AXMEDIS tool<br>xsd:string **axdom**: domain where the user is registered.<br>xsd:string **toolFingerprint**: fingerprint (software and hardware parts) of the installed tool<br>xsd:string **regDeadline**: registration deadline of the installed tool. |
| Output parameters | CertificationResult complex type formed by sequence of:<br>xsd:string **axtid**, the identifier of the installed tool associated to a user and device.<br>xsd:int **certificationResult**, which indicates the result of the certification, according to the following numeration:<br><br>0: OK<br>-1: invalid AXID<br>-2: user not registered<br>-3: user blocked<br>-4: user domain mismatch<br>-5: user registration deadline expired<br>-6: tool not registered (RegTools table)<br>-7: registered tool is blocked<br>-8: received tool deadline exceeds registered tool deadline (user and tool have been blocked)<br>-9: received tool deadline has expired<br>-10: registered tool fingerprint mismatch. Tool has been manipulated (user and tool have been blocked)<br>-11: user-tool-device had already been certified. New tool certificate should be created<br>-20: error updating user status in database<br>-21: error inserting new entry in CerTools table<br>-22: error in AXSupervisor when communicating with database<br>-30: internal AXCV error<br>xsd:string **enablingCode**, AXMEDIS tool activation code sent to the Protection Processor (bytes are encoded using Base64).<br>xsd:string **toolBase64PKCS12**, PKCS12 structure bytes encoded in Base. It includes 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<br>AXMEDIS tool has been certified and can be used in the AXMEDIS framework<br><br>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). |

| **AXCV library and Web Service** | |
|---|---|
| Method | verify |
| Description | This method is called by the AXCS Proxy integrated in PMS Client, which 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 or B2BUser (AXUID)<br>xsd:string **axtid**: identifier of the certified tool (the single instance of the tool installed on a device).<br>xsd:string **axdom**: domain where the user is registered.<br>xsd:string **toolFingerprintDigest**: hash of the full fingerprint (software and hardware parts) of the installed tool.<br>xsd:string **regDeadline**: registration deadline of the installed tool.<br>xsd:string **LastFPPA**: fingerprint actions 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:<br>xsd:int **verificationResult**, which indicates the result of the verification, according to the following numeration:<br><br>0: OK<br>-1: invalid AXID<br>-2: user not registered<br>-3: user blocked<br>-4: user domain mismatch<br>-5: user registration deadline expired<br>-6: AXTID does not exist<br>-7: installed (and certified) tool is blocked<br>-8: tool deadline has expired<br>-9: toolFingerprintDigest (toolFingerprint hash) mismatch<br>-10: toolFingerprint mismatch (user and tool have been blocked)<br>-11: registered tool is blocked<br>-12: user has been blocked and installed tool has been blocked again<br>-13: tool has been blocked<br>-20: error updating user status in database<br>-21: error updating tool status in database<br>-22: error updating LastFPPA in database<br>-23: error retrieving regtool data from database<br>-24: error in AXSupervisor when communicating with AXCS accounting database in storeListActionLog or storePMSActionLog<br>-25: error in AXSupervisor when communicating with AXCS accounting database in storeSID<br>-30: internal AXCV error<br><br>xsd:int **storeListActionLogResult**, which indicates the result of the storage of the received ActionLog list, which is performed by AXS:<br><br>0: ActionLog(s) has been stored: it includes the case of empty list<br>-1: ActionLog(s) has been stored: tool should have been already blocked<br>-2: ActionLog(s) has been stored: tool operation history hash (LastFPPA) is not consistent<br>-3: ActionLog(s) has not been stored: error in AXSupervisor when communicating with AXCS database<br>-4: ActionLog(s) has not been stored: input actionLog(s) do not refer to the same AXTID<br>-5: ActionLog(s) has not been stored: input actionLog(s) have some non-nillable null fields<br>-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). |

| **AXCV library and Web Service** | |
|---|---|
| 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 or B2BUser (AXUID)<br>xsd:string axtid: identifier of the certified tool (the single instance of the tool installed on a device).<br>xsd:string **axdom**: domain where the user is registered.<br>xsd:string **toolFingerprint**: full fingerprint (software and hardware parts) of the installed tool.<br>xsd:string **regDeadline**: registration deadline of the installed tool.<br>xsd:string **LastFPPA**: fingerprint actions performed during the offline operation.<br>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 | The results are the same as for the verify method. |

| **AXCV library and Web Service** | |
|---|---|
| Method | verifyPmsActionLog |
| Description | This method is used to store a single ActionLog after the authorisation of a user in PMS Server, if AXCS is online. The difference between this method and verify method is that in this case, it is PMS Server who partially fills and sends the single input ActionLog. Moreover, PMS does not send the lastFPPA of the client, because it is not known. Therefore, AXS will calculate the new lastFPPA and store it in AXCS database without verifying it. We can suppose that, as AXCV verify method is called just before performing the authorisation, the operation history will be already verified. |
| Input parameters | tns2:ActionLog **pmsActionLog**: single ActionLog, which is a complex type defined in AXMEDIS Supervisor, including the action performed during the online operation. |
| Output parameters | The same as in previous methods: verify and reverify |

| **AXCV Web Service** | |
|---|---|
| Method | ping |
| Description | This method is used by PMS to determine if AXCV is online. |
| Input parameters | xsd:int **input**: any integer |
| Output parameters | xsd:int pingReturn: an integer with value 2 if AXCV web service is online |

# 5 AXMEDIS Supervisor, AXS (FUPF)

| Module/Tool Profile | | |
|---|---|---|
| AXMEDIS Supervisor, AXS | | |
| Responsible Name | Víctor Torres | |
| Responsible Partner | FUPF | |
| Status (proposed/approved) | Approved | |
| Implemented/not implemented | Implemented | |
| Status of the implementation | First version available | |
| Executable or Library/module (Support) | Library and Web service | |
| Single Thread or Multithread | Multithread | |
| Language of Development | Java | |
| Platforms supported | All | |
| Reference to the AXFW location of the source code demonstrator | https://cvs.axmedis.org/repos/Framework/source/axcs-axs<br>https://cvs.axmedis.org/repos/WebServices/axcs-axs/source | |
| Reference to the AXFW location of the demonstrator executable tool for internal download | https://cvs.axmedis.org/repos/Framework/bin/axcs-axs/axs.jar<br>https://cvs.axmedis.org/repos/WebServices/axcs-axs/bin/wsaxs.jar | |
| 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://193.145.45.173:8080/axis/AXS<br>https://193.145.45.173:8443/axis/AXS<br>http://flauto.dsi.unifi.it:8080/axis/services/AXS | |
| Test cases (present/absent) | present | |
| Test cases location | http://cvs.axmedis.org/repos/Framework/doc/test/axcs-axs<br>http://cvs.axmedis.org/repos/WebServices/axcs-axs/doc/test | |
| Usage of the AXMEDIS configuration manager (yes/no) | No | |
| Usage of the AXMEDIS Error Manager (yes/no) | No | |
| Major Problems not solved | | |
| Major pending requirements | | |
| | | |
| Interfaces API with other tools, named as | Name of the communicating tools References to other major components needed | Communication model and format (protected or not, etc.) |
| PMS Server (AXCS proxy) | | |
| AXMEDIS Certification And Verification | | |
| AXCS Database Interface | | |
| | | |
| 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 | | |
| AXCSAccounting | | |
| | | |
| | | |
| | | |
| | | |
| 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 |
| Bouncy Castle | bcprov-jdk15-128.jar | http://www.bouncycastle.org/licence.html |
| MySql | mysql-connector-java-3.1.8-bin.jar | |
| | | |
| | | |

## 5.1 General Description of the Module

| **AXS** | |
|---|---|
| ***Methods*** | ***Description*** |
| storeSID | This function is used to receive SupervisorInputData and to store them in the AXCS database.<br>1) AXCV uses this method to store SupervisorInputData when auser is blocked during certification or verification: to be able to know the reason why it was blocked<br>2) PMS Server uses this method to store SupervisorInputData when:<br>2.1 an AXMEDIS user requests a license to consume an AXMEDIS object (on-line or off-line). The information about the license request is sent to Supervisor.<br>2.2 an end user requests permission to perform an action on an AXMEDIS Object, PMS Server does not positively authorize the user and sends Supervisor the information about the negative authorization of the user. Then, Supervisor stores that information in AXCS Database. |
| storeListActionLog | This method is used by AXCV 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. |
| | |
| getProtectionInfo | This method is used to retrieve the protection information related to an object from the Objects Table of the AXCS Objects ID Database. |
| saveProtectionInfo | 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. |
| storePMSActionLog | This method is used to store a single ActionLog after the authorisation of a user in PMS Server, if AXCS is online.<br>The difference between this method and storeListActionLog is that in this case, it is PMS Server who partially fills and sends the full ActionLog. Moreover, PMS does not send the lastFPPA of the client, because it is not known. Therefore, AXS will calculate the new lastFPPA and store it in AXCS database without verifying it. We can suppose that, as AXCV verify method is called just before performing the authorisation, the operation history will be already verified. |
| getProtectionInfo | This method 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 used to insert or update the protection information related to an AXMEDIS object in the Objects Table of the AXCS Objects ID Database. |

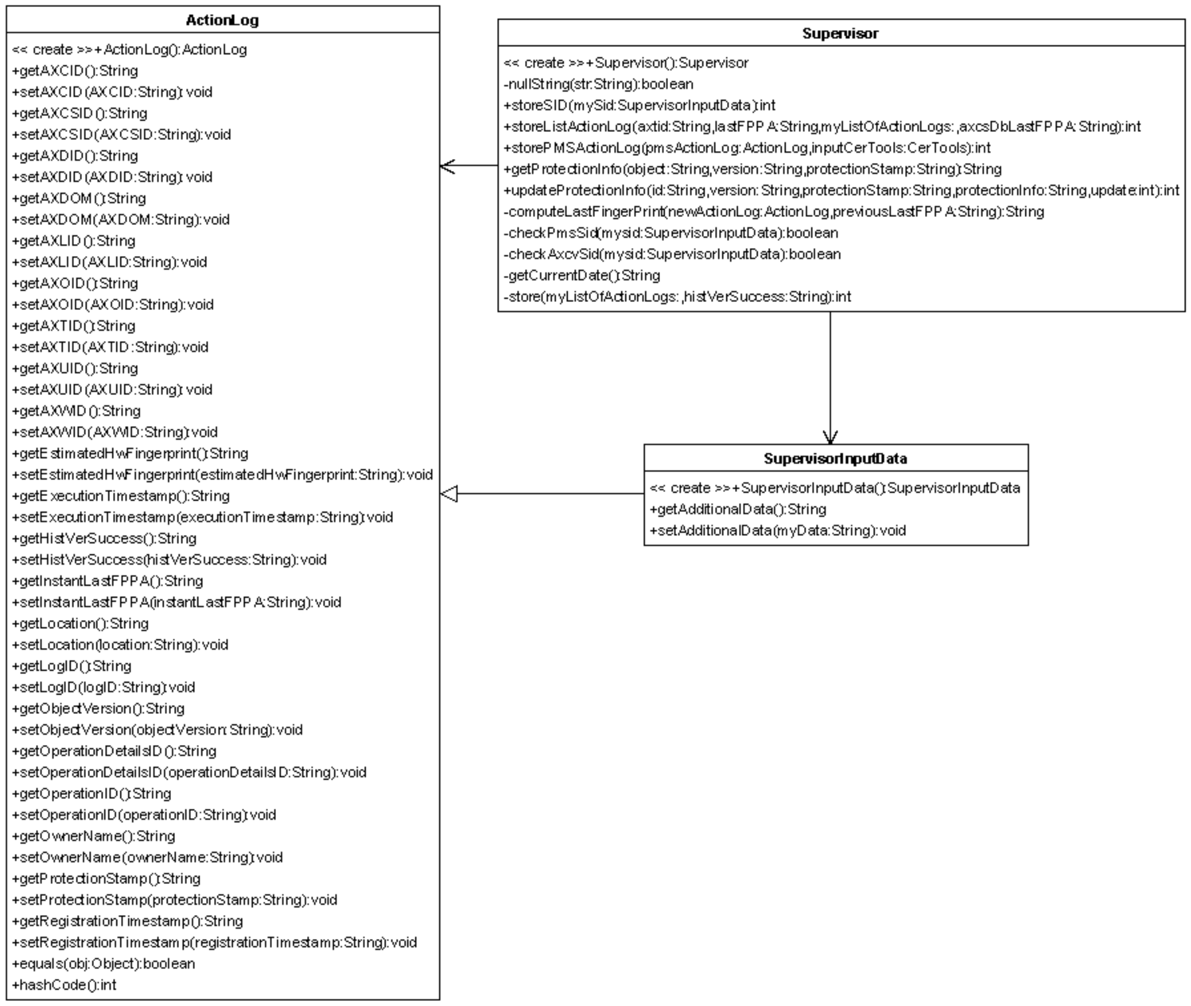
**Difference between Action Log and Supervisor Input Data:**

The difference between Action Log and Supervisor Input Data is that the first one is directly related with the AXMEDIS Object (it is created when any action over the object is done) and Supervisor Input Data is not directly related to the object (it is not created when an action over the object is done, but it can refers to the object). The Supervisor Input Data is created to notify to Supervisor any event in (or between) modules.

Mainly both of them have the structure of the MPEG-21 Event Report, however the Supervisor Input Data does not use some fields and there is an additional data field added.

The objective of this difference is to clarify when the notifications come from the object (actions done over the object) and when they are between modules.

## 5.2 Module Design in terms of Classes


ActionLog
<< create >>+ActionLog():ActionLog
+getAXCID():String
+setAXCID(AXCID:String):void
+getAXCSID():String
+setAXCSID(AXCSID:String):void
+getAXDID():String
+setAXDID(AXDID:String):void
+getAXDOM():String
+setAXDOM(AXDOM:String):void
+getAXLID():String
+setAXLID(AXLID:String):void
+getAXOID():String
+setAXOID(AXOID:String):void
+getAXTID():String
+setAXTID(AXTID:String):void
+getAXUID():String
+setAXUID(AXUID:String):void
+getAXWID():String
+setAXWID(AXWID:String):void
+getEstimatedHwFingerprint():String
+setEstimatedHwFingerprint(estimatedHwFingerprint:String):void
+getExecutionTimestamp():String
+setExecutionTimestamp(executionTimestamp:String):void
+getHistVerSuccess():String
+setHistVerSuccess(histVerSuccess:String):void
+getInstantLastFPPA():String
+setInstantLastFPPA(instantLastFPPA:String):void
+getLocation():String
+setLocation(location:String):void
+getLogID():String
+setLogID(logID:String):void
+getObjectVersion():String
+setObjectVersion(objectVersion:String):void
+getOperationDetailsID():String
+setOperationDetailsID(operationDetailsID:String):void
+getOperationID():String
+setOperationID(operationID:String):void
+getOwnerName():String
+setOwnerName(ownerName:String):void
+getProtectionStamp():String
+setProtectionStamp(protectionStamp:String):void
+getRegistrationTimestamp():String
+setRegistrationTimestamp(registrationTimestamp:String):void
+equals(obj:Object):boolean
+hashCode():int
Supervisor
<< create >>+Supervisor():Supervisor
-nullString(str:String):boolean
+storeSID(mySid:SupervisorInputData):int
+storeListActionLog(axtid:String,lastFPPA:String,myListOfActionLogs:,axcsDbLastFPPA:String):int
+storePMSActionLog(pmsActionLog:ActionLog,inputCerTools:CerTools):int
+getProtectionInfo(object:String,version:String,protectionStamp:String):String
+updateProtectionInfo(id:String,version:String,protectionStamp:String,protectionInfo:String,update:int):int
-computeLastFingerPrint(newActionLog:ActionLog,previousLastFPPA:String):String
-checkPmsSid(mysid:SupervisorInputData):boolean
-checkAxcvSid(mysid:SupervisorInputData):boolean
-getCurrentDate():String
-store(myListOfActionLogs:,histVerSuccess:String):int
SupervisorInputData
<< create >>+SupervisorInputData():SupervisorInputData
+getAdditionalData():String
+setAdditionalData(myData:String):void

## 5.3 Technical and Installation information

| | |
|---|---|
| References to other major components needed | AXMEDIS AXS (Supervisor) library<br>AXCS Database Interface library |
| Problems not solved | None |
| Configuration and execution context | <u>How to deploy/undeploy the web service</u><br><br>To deploy/undeploy the AXS AXIS web service in your Tomcat server you can use the org.apache.axis.client.AdminClient class using the provided deploy.wsdd or undeploy.wsdd files as parameters. See the example below.<br><br><u>Classes needed to use the software</u><br><br>- Client side: wsaxs.jar library, which contains client side classes:<br>- es.fupf.dmag.axmedis.wsaxs.Supervisor<br>- es.fupf.dmag.axmedis.wsaxs.SupervisorService<br>- es.fupf.dmag.axmedis.wsaxs.SupervisorServiceLocator |

- es.fupf.dmag.axmedis.wsaxs.AxsSoapBindingStub
- es.fupf.dmag.axmedis.wsaxs.SupervisorInputData
- es.fupf.dmag.axmedis.wsaxs.ActionLog

- Server side: wsaxs.jar library, which contains server side classes:
    - es.fupf.dmag.axmedis.wsaxs.Supervisor
    - es.fupf.dmag.axmedis.wsaxs.AxsSoapBindingImpl
    - es.fupf.dmag.axmedis.wsaxs.AxsSoapBindingSkeleton
    - es.fupf.dmag.axmedis.wsaxs.SupervisorInputData
    - es.fupf.dmag.axmedis.wsaxs.ActionLog

<u>Libraries needed to run the software</u>

- Client side
    - bcprov-jdk15-128.jar (only for testing)
    - mysql-connector-java-3.1.8-bin.jar (only for testing)
    - junit.jar (only for testing)
    - AXIS related libraries:
        - mail.jar
        - activation.jar
        - axis.jar
        - jaxrpc.jar
        - saaj.jar
        - commons-logging-1.0.4.jar
        - commons-discovery-0.2.jar
        - wsdl4j-1.5.1.jar
    - other dependencies
        - AXS library
        - AXCS Database Interface library (only for testing)

- Server side
    - bcprov-jdk15-128.jar
    - mysql-connector-java-3.1.8-bin.jar
    - AXIS related libraries
        - mail.jar
        - activation.jar
        - axis.jar
        - jaxrpc.jar
        - saaj.jar
        - commons-logging-1.0.4.jar
        - commons-discovery-0.2.jar
        - wsdl4j-1.5.1.jar
    - other dependencies
        - AXS library
        - AXCS Database Interface library

<u>Files needed for the deployment of AXS library in the Server side</u>

The following files must be included anywhere in the local classpath where the software is executed:

- axs.properties: RegCertDSN, AccountingDSN, ObjectsIdDSN, AxcsDbUser and AxcsDbPassword parameters can be customised but are only used if the axcsdb.ini file from DSI database is not found in the system.

| | |
|---|---|
| | Example of deployment in Linux server<br><br>cd ($AXIS_HOME)<br>java -cp<br>"($AXIS_HOME)/WEB-INF/lib/wsaxs.jar:<br>($AXIS_HOME)/WEB-INF/lib/axs.jar:<br>($AXIS_HOME)/WEB-INF/lib/axcs-db-interface.jar:<br>($AXIS_HOME)/WEB-INF/lib/mysql-connector-java-3.1.8-bin.jar:<br>($AXIS_HOME)/WEB-INF/lib/bcprov-jdk15-128.jar:<br>($AXIS_HOME)/WEB-INF/lib/activation.jar:<br>($AXIS_HOME)/WEB-INF/lib/mail.jar:<br>($AXIS_HOME)/WEB-INF/lib/xerces.jar:<br>($AXIS_HOME)/WEB-INF/lib/axis.jar:<br>($AXIS_HOME)/WEB-INF/lib/axis-ant.jar:<br>($AXIS_HOME)/WEB-INF/lib/jaxrpc.jar:<br>($AXIS_HOME)/WEB-INF/lib/saaj.jar:<br>($AXIS_HOME)/WEB-INF/lib/commons-logging-1.0.4.jar:<br>($AXIS_HOME)/WEB-INF/lib/commons-discovery-0.2.jar:<br>($AXIS_HOME)/WEB-INF/lib/wsdl4j-1.5.1.jar:<br>($AXIS_HOME)/WEB-INF/lib/log4j-1.2.8.jar:<br>($AXIS_HOME)/WEB-INF/classes/"<br>org.apache.axis.client.AdminClient -p80 "($AXIS_HOME)/WEB-INF/($WSDD_PATH)/deploy.wsdd"<br><br>In the above example "($AXIS_HOME)/WEB-INF/lib/" contains all necessary libraries and dependencies, and "($AXIS_HOME)/WEB-INF/classes/" contains the files needed by dependent libraries:<br><br>- AXS library needed files (see AXMEDIS Supervisor, AXS):<br>    - axs.properties<br>How to configure the server to have secure AXIS Web Services using Tomcat 5.5<br><br>Refer to section 4.4 for details<br><br>How to configure the client to have secure AXIS Web Services<br><br>Refer to section 4.4 for details |

## 5.4 Examples of usage

A JUnit-based Test is provided to perform a full test of AXS web service functionalities. It shows how the AXS web service can be invoked to obtain many different results. It is available at: http://cvs.axmedis.org/repos/WebServices/axcs-axs/doc/test/wsAXSTest.java

## 5.5 Integration and compilation issues

You can deploy the provided web service libraries on any device, which supports AXIS and Java. For other Web Services platforms you must generate the corresponding classes and files from the AXCV web service wsdl file.

## 5.6 Configuration Parameters

The following parameters can be configured in axs.properties file.

| Config parameter | Possible values |
|---|---|
| | |
| AccountingDSN | AXCS Accounting database DSN. This parameter will be only used if the axcsdb.ini file from DSI database is not found in the system.<br>Possible values:<br>If DSI database is present in the system: foo<br>If DSI database is not present in the system: jdbc:mysql://193.145.45.173/axcsaccounting |
| ObjectsIdDSN | AXCS Objects ID database DSN. This parameter will be only used if the axcsdb.ini file from DSI database is not found in the system.<br>Possible values:<br>If DSI database is present in the system: foo<br>If DSI database is not present in the system: jdbc:mysql:// 193.145.45.173/axcsobjectsid |
| AxcsDbUser | AXCS Database User. This parameter will be only used if the axcsdb.ini file from DSI database is not found in the system.<br>Possible values:<br>If DSI database is present in the system: foo<br>If DSI database is not present in the system: databaseUser |
| AxcsDbPassword | AXCS Database Password. This parameter will be only used if the axcsdb.ini file from DSI database is not found in the system.<br>Possible values:<br>If DSI database is present in the system: foo<br>If DSI database is not present in the system: databasePassword |

## 5.7 Formal description of AXS Algorithms

| AXS Library and Web Service | |
|---|---|
| Method | storeSID |
| Description | This function is used to receive SupervisorInputData and to store them in the AXCS database.<br>1) AXCV uses this method to store SupervisorInputData when a user is blocked during certification or verification: to be able to know the reason why it was blocked<br>2) PMS Server uses this method to store SupervisorInputData when:<br>2.1 an AXMEDIS user requests a license to consume an AXMEDIS object (on-line or off-line). The information about the license request is sent to Supervisor.<br>2.2 an end user requests permission to perform an action on an AXMEDIS Object, PMS Server does not positively authorize the user and sends Supervisor the information about the negative authorization of the user. Then, Supervisor stores that information in AXCS Database. |
| Input parameters | axs:SupervisorInputData complex type formed by sequence of:<br>additionalData type="xsd:string"<br>axs:SupervisorInputData extends axs:ActionLog. axs:ActionLog is a complex type f formed by sequence of:<br>AXCID type="xsd:string"<br>AXCSID type="xsd:string"<br>AXDID type="xsd:string"<br>AXDOM type="xsd:string"<br>AXLID type="xsd:string"<br>AXOID type="xsd:string"<br>AXTID type="xsd:string"<br>AXUID type="xsd:string"<br>AXWID type="xsd:string"<br>estimatedHwFingerprint type="xsd:string"<br>executionTimestamp type="xsd:string"<br>histVerSuccess type="xsd:string"<br>instantLastFPPA type="xsd:string" |

| | |
|---|---|
| | location type="xsd:string"<br>logID type="xsd:string"<br>objectVersion type="xsd:string"<br>operationDetailsID type="xsd:string"<br>operationID type="xsd:string"<br>ownerName type="xsd:string"<br>registrationTimestamp type="xsd:string" |
| Output parameters | **StoreSIDReturn** of type="xsd:int", which indicates the result of this request, according to the following numeration:<br><br>0: ok<br>-1: SupervisorInputData has some non-nillable null fields<br>-2: operationID should be set to: "PMS SupervisorInputData" or "AXCV SupervisorInputData"<br>-3: error in AXSupervisor when communicating with AXCS accounting database |

| **AXS Library and Web Service** | |
|---|---|
| Method | storeListActionLog |
| Description | This method is used by AXCV 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. |
| Input parameters | Array of axs:ActionLog, which is a complex type previously defined. |
| Output parameters | type="xsd:int" **storeListActionLogReturn**, which indicates the result of this request, according to the following numeration:<br>0: ActionLog(s) has been stored: it includes the case of empty list<br>-1: ActionLog(s) has been stored: tool should have been already blocked<br>-2: ActionLog(s) has been stored: tool operation history hash (LastFPPA) is not consistent<br>-3: ActionLog(s) has not been stored: error in AXSupervisor when communicating with AXCS database<br>-4: ActionLog(s) has not been stored: input actionLog(s) do not refer to the same Axtid<br>-5: ActionLog(s) has not been stored: input actionLog(s) have some non-nillable null fields |

| **AXS Library and Web Service** | |
|---|---|
| Method | getProtectionInfo |
| Description | This method is used by PMS Server 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:<br>type="xsd:string" **AXOID**, AXMEDIS object identifier<br>type="xsd:string" **ObjectVersion**, object version<br>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 |

| **AXS Library and Web Service** | |
|---|---|
| Method | updateProtectionInfo |
| Description | This method is used by PMS Server to insert or update the protection information related to an AXMEDIS object in the Objects Table of the AXCS Objects ID Database. |
| Input | The following fields of the Objects table in the AXCS Objects ID database: |

| | |
|---|---|
| parameters | type="xsd:string" **AXOID**, AXMEDIS object identifier<br>type="xsd:string" **ObjectVersion**, object version<br>type="xsd:string" **ProtectionStamp**, protection stamp<br>type="xsd:string" **ProtectionInfo**, protection information to be updated<br>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:<br>0: OK<br>-1: there is not any entry in AXCS Objects database that matches the input information<br>-2: error in AXSupervisor when updating ProtectionInfo in AXCS Objects database |

## 5.8 Formal description of algorithm to calculate the history of Action Logs fingerprint

This algorithm in used by AXS to calculate the history of Action Logs fingerprint (history hash or lastFPPA). The same algorithm must be used in the client side so that the appropriate checks can be performed, as explained in the next section.

$FP_1$ = MD5 ( $AXOID_1$ | $ObjectVersion_1$ | $ProtectionStamp_1$ | $AXUID_1$ | $AXTID_1$ | $OperationID_1$ | $ExecutionTimestamp_1$ | $estimatedHWFingerprint_1$ | $AXLID_1$ | $AXDOM_1$)

$FP_2$ = MD5 ( $FP_1$ | $AXOID_2$ | $ObjectVersion_2$ | $ProtectionStamp_2$ | $AXUID_2$ | $AXTID_2$ | $OperationID_2$ | $ExecutionTimestamp_2$ | $estimatedHWFingerprint_2$ | $AXLID_2$ | $AXDOM_2$), where $FP_2$ is the fingerprint to be inserted in the secod Action Log of a user.

…

$FP_n$ = MD5 ( $FP_{n-1}$ | $AXOID_n$ | $ObjectVersion_n$ | $ProtectionStamp_n$ | $AXUID_n$ | $AXTID_n$ | $OperationID_n$ | $ExecutionTimestamp_n$ | $estimatedHWFingerprint_n$ | $AXLID_n$ | $AXDOM_n$)

Where,

$FP_1$ is the fingerprint to be inserted in the first Action Log of a user on a tool
$FP_n$ is the fingerprint to be inserted in the nth Action Log of a user on a tool
$AXOID_n$, $ObjectVersion_n$, etc.denote the AXOID, ObjectVersion, etc. involved in the nth Action Log of a user on a tool denotes the concatenation of strings

MD5 denotes the calculation of the MD5 hash which is finally encoded in Base64 and returned as a byte[].

## 5.9 Formal description of algorithm to check the consistency of the history of Action Logs fingerprint

This algorithm in used by AXS to check if the fingerprint of the history of Action Logs is consistent regarding the information that is stored in AXCS database and the list of action logs and lastFPPA received in storeListActionLog method. It has been revised according to the adopted solution: 2) To have a single LastFPPA common for all users in the same tool.

1. Get the last history hash (lastFPPA) of the certified tool from AXCS CerTools table of AXCS RegCert database.

2a. If it is not present, set tmpLastFingerprint to “” (blank). Go to step 3.

2b. If it is present, and its value is not “invalid”, set tmpLastFingerprint to lastFPPA value. Go to step 3.

2c. If it is present, and its value is “invalid”, block the certified tool (this case should not happen, as it should have been already blocked). A SupervisorInputData is stored to explain why. End.

3. For each element of the ActionLog input array, beginning by element 0 until i-th, do step 4.

4. Calculate $FP_n$ as explained in the previous section, where $FP_{n-1}$ will be tmpLastFingerprint value. Set tmpLastFingerprint to $FP_{n.}$

5. Compare tmpLastFingerprint obtained at the end of the whole iteration in step 4 to LastFPPA field received as an input parameter (this parameter is set in the client side).

6a. If they match, the history is consistent. The received ActionLogs histVerSuccess field is set to “1” (true) and they are stored in the AXCS Accounting database. LastFPPA is updated in CerTools table of AXCS RegCert database to the value of tmpLastFingerprint. End.

6b. If they do not match, the history is not consistent. The received histVerSuccess field of all the received action logs is set to “0” (false) and they are stored in the AXCS Accounting database. LastFPPA is updated in CerTools table of AXCS RegCert database to the value “invalid”. The certified tool is blocked. A SupervisorInputData is stored to explain why. End.

Note: The received list of Action Logs must be in a predefined order. That is, element 0 must be the oldest or least recent performed action in the certified tool.

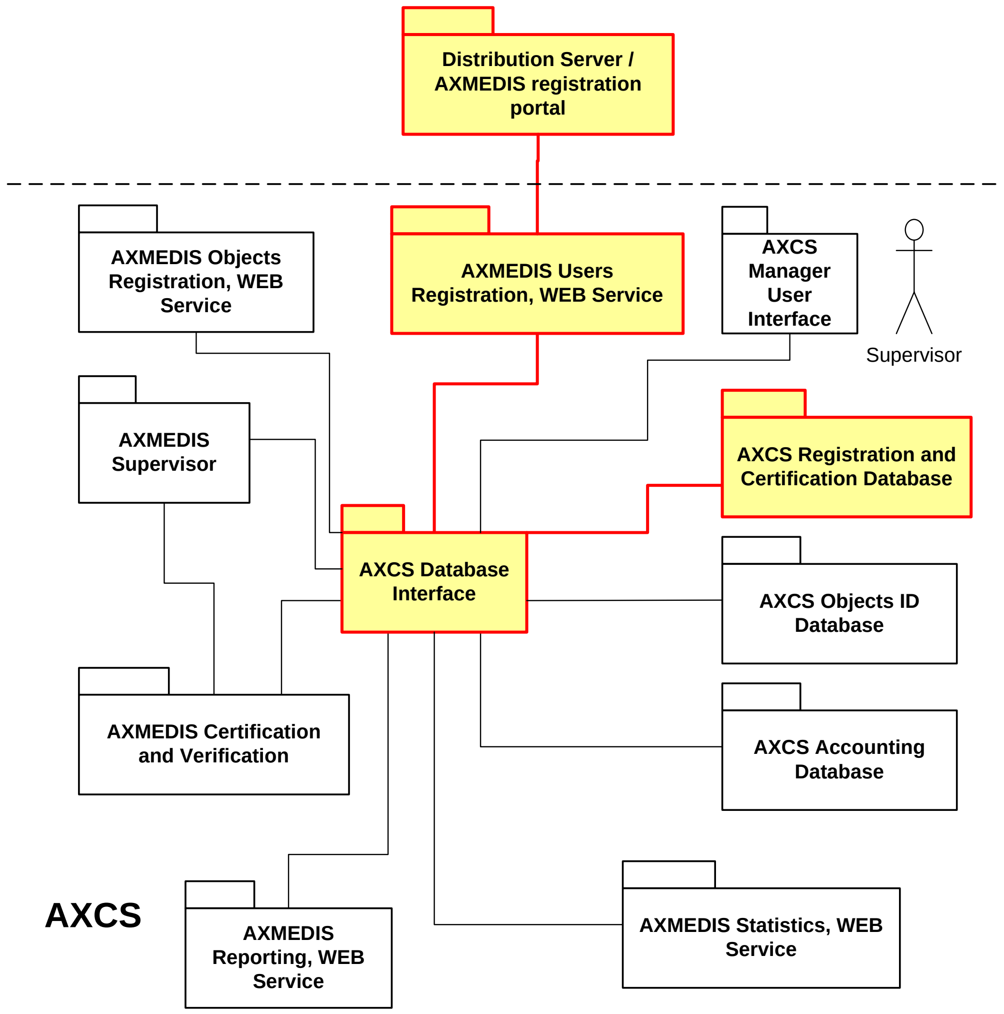
# 6 AXCS Users Registration Web Service (DSI)

| Module/Tool Profile | | |
|---|---|---|
| AXCS Users Registration Web Service | | |
| Responsible Name | Chellini, Martini | |
| Responsible Partner | DSI | |
| Status (proposed/approved) | Proposed | |
| Implemented/not implemented | Implemented | |
| Status of the implementation | In refinement | |
| Executable or Library/module (Support) | Web service | |
| Single Thread or Multithread | Multithread | |
| Language of Development | Java | |
| Platforms supported | Anyone on which can be installed a servlet container and a JRE 1.5.0 (5.0). It has been tested using Tomcat 5.5, Axis 1.3, JRE 1.5.0_06 | |
| Reference to the AXFW location of the source code demonstrator | https://cvs.axmedis.org/repos/Framework/source/axcs/axcs-framework/src/org/axmedis/axcs/services | |
| Reference to the AXFW location of the demonstrator executable tool for internal download | https://cvs.axmedis.org/repos/WebServices/axcs-registration | |
| 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://flauto.dsi.unifi.it:8080/axis/services/AXCSUserRegistrator | |
| Test cases (present/absent) | present | |
| Test cases location | https://cvs.axmedis.org/repos/WebServices/axcs-registration/bin/org/axmedis/axcs/ws/registration/ClientRegistration.class | |
| 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 | User data updates are not possible, status of the requestor is not checked, communication is not using a protected channel | |
| | | |
| 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.) |
| registration | axcs-db-interface | Protected (SSL) |
| | | |
| | | |
| | | |
| Formats Used | Shared with | format name or reference to a section |
| AXMEDIS prefixes | All AXCS Web services | See section 20 |
| AXMEDIS ID | All AXCS Web services | See section 23 |

| | | |
|---|---|---|
| | | |
| Protocol Used | Shared with | Protocol name or reference to a section |
| AXCS user registration protocol | | AXCSUserRegistration (see section 36) |
| | | |
| | | |
| | | |
| Used Database name | | |
| AXCSRegCert | | |
| | | |
| | | |
| | | |
| | | |
| 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 |
| UUID generation | Jug (Doomdark) | |
| MySQL JDBC driver | Connector/J 3.1.10 | GPL, but commercial licenses are available |
| | | |
| | | |
| | | |
| | | |

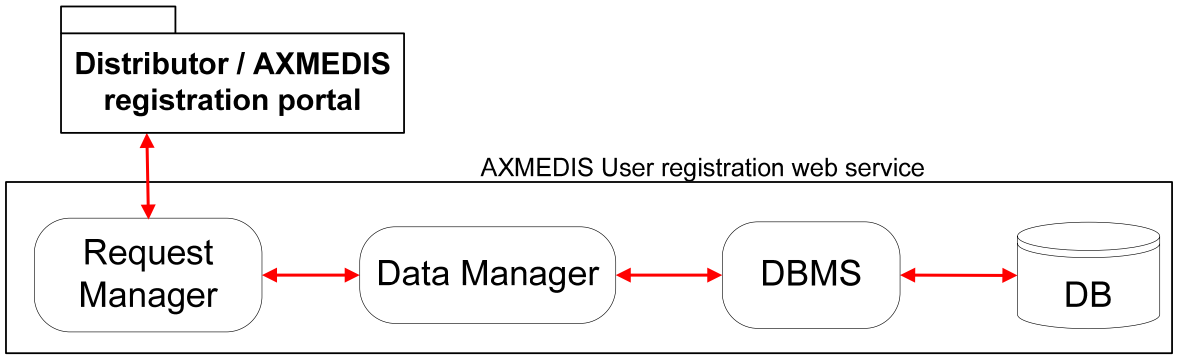
## 6.1 General Description of the Module

All AXMEDIS users must be registered by an AXCS. Several information collected by Distributors or the AXMEDIS registration portal in the registration phase has to be transferred to an AXCS. The User Registration Web Service is the AXCS module that receives this information and stores it in the AXCS Registration and Certification Database. Once the user is registered in the system and the related data are stored in the AXCS Registration and Certification Database, other AXCS modules can access this database to retrieve user information to perform their work (please note that each date is considered in the format yyyy-mm-ddThh:mm:ss, where "T" is used as separator between date and time and the time is assumed to be referred to GMT+0).
The following figure shows which AXCS modules are involved in the user registration process:


Distribution Server / AXMEDIS registration portal
AXMEDIS Objects Registration, WEB Service
AXMEDIS Users Registration, WEB Service
AXCS Manager User Interface
Supervisor
AXMEDIS Supervisor
AXCS Registration and Certification Database
AXCS Database Interface
AXCS Objects ID Database
AXMEDIS Certification and Verification
AXCS Accounting Database
AXCS
AXMEDIS Reporting, WEB Service
AXMEDIS Statistics, WEB Service

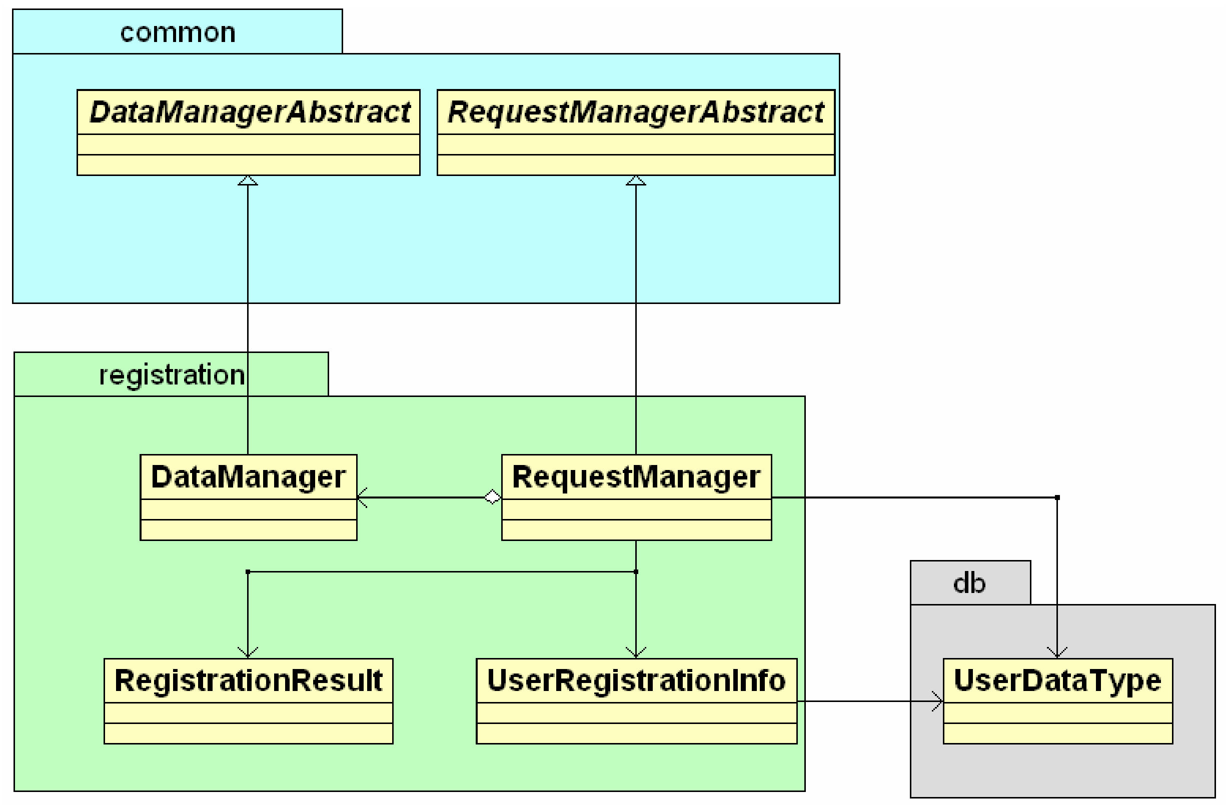
It can be identified the following logical decomposition in the structure of the User Registration Web Service:

- **Request Manager**: this component receives registration requests from Distributor or the AXMEDIS registration portal and prepares them to be processed by the Data Manager. The communication channel connecting Distributor and Request Manager is protected using a secure protocol (for instance SSL). It implements the interface with requestors and manage the whole application.
- **Data Manager**: this component receives data from Request Manager and inserts it in the AXCS Registration and Certification Database. If necessary elaborates and fits data before insert it in the database. The database management is performed using the related API (AXCS-DB-INTERFACE API).


Distributor / AXMEDIS registration portal
AXMEDIS User registration web service
Request Manager
Data Manager
DBMS
DB

## 6.2 Module Design in terms of Classes

The logic at the base of the registration mechanism has been subdivided in some classes in order to separate concerns and to make it easier to reuse some functionalities in other modules. The following figure shows the class diagram of the AXCS User Registration web service (the name of the packages have been truncated to keep the figure clear):



## 6.3 User interface description

No user interface is available.

## 6.4 Technical and Installation information

Since the AXCS Users Registration is a web service, it has to be deployed on the chosen platform (ie Tomcat/AXIS, Tomcat/JWSDP, SAS, etc).

| | |
|---|---|
| References to other major components needed | Axcs-db-interface, AXCSRegCert database |
| Problems not solved | NONE |
| Configuration and execution context | DEPLOY PROCEDURE:<br>1. Use the AdminClient utility that comes with Axis distribution to deploy this web services with the deploy_regs.wsdd file supplied.<br>2. You need also to create a directory called "axcs" (lower case). Place this directory in one of the following location (depending on the OS):<br><br>WINDOWS: C:\axcs |

*NIX: /axcs

3. In the axcs directory create a new text file named "axcsdb.ini" (lower case) in which you have to specify location of the AXCS databases and authentication data to access these databases.
   The file has to be written in the following format.

   ```
   AXCSAccounting =
   jdbc:mysql://<location_of_AXCSAccouning_db>/AXCSAccounting
   AXCSRegCert =
   jdbc:mysql://<location_of_AXCSRegCert_db>/AXCSRegCert
   AXCSObjectsID =
   jdbc:mysql://<location_of_AXCSObjectsID_db>/AXCSObjectsID

   AXCSDbUser = <user_name>
   AXCSDbPassword = <password>

   [dbLog = <log_file_name_with_path>]
   ```

   The last line is optional. It lets you specify a logging file in which errors can be written during execution. This file can be placed anywhere you like, but the complete path has to be specified, escaping backslash characters (escape character is a backslash itself). If the last line is not supplied, no log will be created.
   All data can be specified in any order and white spaces do not matter.
   Please note that the autentication data have to be the same for all the three databases.

   For instance, if the databases would be hosted on a machine called "MyHost", autentication data would be username: "User" and password: "Pw" and logging file would be "C:\axcs\log_file.txt", the file would appear something similar to:

   ```
   AXCSAccounting = jdbc:mysql://MyHost/AXCSAccounting
   AXCSRegCert = jdbc:mysql://MyHost/AXCSRegCert
   AXCSObjectsID = jdbc:mysql://MyHost/AXCSObjectsID

   AXCSDbUser = User
   AXCSDbPassword = Pw

   dbLog = C:\\axcs\\log_file.txt
   ```

UNDEPLOY PROCEDURE
Use the AdminClient utility that comes with Axis distribution to undeploy this web services with the undeploy_regs.wsdd file supplied.

| | |
|---|---|
| | Platform used in development:Tomcat 5.5 / Axis 1.3, JRE 5.0 |

## 6.5 Draft User Manual

Since the AXCS User Registration is a web service, the public methods it provides should be called using a web service client application. The web service client should call the needed method giving the necessary input parameters and receiving the results. The web service client applications should be implement to use the "Document" style and "Literal" use.
The requestor calls the "registration" method, providing his user name and password and the user data he is requesting registration for. If his credentials are correct, he obtains the access to the system and the sent user data are checked and stored in the appropriate database tables. At the end the user is registered in the system. Note that the requestor can be a distributor (which is a granting subject for the user) or the AXMEDIS Portal (in this case the user has no granting subject).

## 6.6 Examples of usage

A simple Java client application is provided as example of usage. It is available at https://cvs.axmedis.org/repos/WebServices/axcs-registration/source/org/axmedis/axcs/ws/registration/ClientRegistration.java

## 6.7 Integration and compilation issues

Since the AXCS Users Registration is a web service, it should work on every supported platform. It has been developed and tested with Tomcat 5.5 with AXIS 1.3 and Java 5. Interoperability is granted by the respect of the specification included in the related WSDL, which is compliant with the WS-I Basic Profile. This has been achieved realising the web service in "Document" style and "Literal" use.
To deploy the web service it is enough to place Jug and Connector/J libraries in the "lib" directory included in the Axis subtree.

## 6.8 Configuration Parameters

| Config parameter | Possible values |
|---|---|
| NONE | NONE |

## 6.9 Errors reported and that may occur

| Error code | Description and rationales |
|---|---|
| 1 | The required operation failed. More details are provided in the error message returned by the service |

## 6.10 Formal description of algorithm User Registration

The **RequestManage**r class implements the following functions:
- Accepting authentication data, needful to verify Distributors credentials and make them access the system
- Accepting Users registration data and preparing them to be transferred to Data Manager
- Reply to requestor with the most appropriate message (on the basis of Data Manager responses)

The **DataManager** class implements the following functions:
- Receiving verification requests and data from the Request Manager

- Accessing the database (using the AXCS-DB-INTERFACE API) to verify Distributors Credential information
- Accessing the database to store received Users registration data
- Reply to Request Manager according to the performed actions

| **RequestManager** | |
|---|---|
| Method | registration |
| Description | It is the only public method of this web service. It collects Distributor credentials needful to access the system and uses the verifyLogin() (a DataManager method) to verify requestor credentials. It collects also registration data (regInfo) provided by the requesting distributor and uses the other methods (described below) to insert them in database and to provide the result to the requesting distributor. |
| Input parameters | UserRegistrationInfo regInfo – a data structure containing the nickname and password of the requesting distributor and a nested data structure containing the data to be inserted/updated |
| Output parameters | RegistrationResult – The result output parameter is a data structure containing three fields:<br>• "resultStatus": it is set to 0 if the registration is successful otherwise is set to 1.<br>• "definitiveUID": this field returns the definitive AXUID in case of success, a *null* value otherwise<br>• "errorMessage": this filed returns the error message in case of failure, a *null* value otherwise |

| **RequestManager** | |
|---|---|
| Method | dataPrepare |
| Description | It prepares the User registration data received from the Distributor by the web service to be stored in DB. It distinguishes between Users and B2B Users (like Creators, Distributors and so on) on the basis of received data. It also uses an UUID generator to get the definitive ID (AXUID) to be inserted in the database and to be sent to requestor |
| Input parameters | UserDataType data – the data to be prepared |
| Output parameters | UserDataType – data prepared to be stored in the database |

| **RequestManager** | |
|---|---|
| Method | encryptComm |
| Description | It encrypts clear data with the provided asymmetric key (asymKey) |
| Input parameters | String asymKey – the key to be used for encryption<br>String clearData – data to be encrypted |
| Output parameters | String – encrypted data |

| **RequestManager** | |
|---|---|
| Method | decryptComm |
| Description | It decrypts clear data with the provided asymmetric key (asymKey) |
| Input parameters | String asymKey – the key to be used for encryption<br>String clearData – data to be encrypted |
| Output parameters | String – decrypted data |

| **DataManager** | |
|---|---|
| Method | verifyLogin |
| Description | It accesses AXCS Registration and Certification Database to verify if the couple NickName and Password is the same provided through the registration process. It also retrieves the user |

| | |
|---|---|
| | public key stored in the DB.<br>If this method returns *null* it means the user is not trusted (is not present in the database pertinent table).<br>Note: the password is encrypted both the one received as input from the method and the one stored in DB. The equality check is made between encrypted strings |
| Input parameters | String nickName – the nickname of a business user entitled to register new users or to update users data<br>String password – the password of a business user entitled to register new users or to update users data |
| Output parameters | UserLoginData – a data structure containing the AXUID and the public key of the user requesting the registration/update operation in case of success, a *null* value otherwise |

| **DataManager** | |
|---|---|
| Method | storeData |
| Description | Stores received data in the AXCS Registration and Certification Database |
| Input parameters | Boolean replace – it specifies whether to perform an insert or an update<br>UserDataType userData – a data structure containing data to be rgistered/updated |
| Output parameters | Boolean – *true* if the operation succeeds, *false* otherwise |

Note that two encrypting/decrypting methods (DecryptComm, EncryptComm) have been introduced to enforce the encrypting robustness. In fact we can suppose to use an encrypted protocol (like SSL), but we can enforce encryption robustness (and therefore security) encrypting ourselves data too using a Public/Private key paradigm. You have to remember that the Distributor public key is stored in AXCS Registration and Certification Database.

In order to generate a user ID that is unique in the whole system, it has been considered to use a standard algorithm: the UUID generator algorithm.
UUID is an identifier that is unique across both space and time, with respect to the space of all UUIDs. A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network. The generation of UUIDs does not require a registration authority for each single identifier. Instead, it requires a unique value over space for each UUID generator. This spatially unique value is specified as an IEEE 802 address, which is usually already applied to network-connected systems. This 48-bit address can be assigned based on an address block obtained through the IEEE registration authority. This UUID specification assumes the availability of an IEEE 802 address. The UUID consists of a record of 16 octets and must not contain padding between fields. The total size is 128 bits.

More information about UUID generation can be found at the following web address:

http://www.opengroup.org/onlinepubs/9629399/apdxa.htm

# 7 AXCS Objects Registration Web Service (DSI)

| Module/Tool Profile | | |
|---|---|---|
| AXCS Objects Registration Web Service | | |
| Responsible Name | Chellini, Martini | |
| Responsible Partner | DSI | |
| Status (proposed/approved) | Proposed | |
| Implemented/not implemented | Implemented | |
| Status of the implementation | In refinement | |
| Executable or Library/module (Support) | Web service | |
| Single Thread or Multithread | Multithread | |
| Language of Development | Java | |
| Platforms supported | Anyone on which can be installed a servlet container and a JRE 1.5.0 (5.0). It has been tested using Tomcat 5.5, Axis 1.3, JRE 1.5.0_06 | |
| Reference to the AXFW location of the source code demonstrator | https://cvs.axmedis.org/repos/Framework/source/axcs/axcs-framework/src/org/axmedis/axcs/services | |
| Reference to the AXFW location of the demonstrator executable tool for internal download | https://cvs.axmedis.org/repos/WebServices/axcs-oid-generator | |
| 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://flauto.dsi.unifi.it:8080/axis/services/AXCSObjectRegistrator | |
| Test cases (present/absent) | present | |
| Test cases location | https://cvs.axmedis.org/repos/WebServices/axcs-oid-generator/bin/web_service/original_code/org/axmedis/axcs/ws/objectsmetadatamanager/ClientRegistrator.class | |
| Usage of the AXMEDIS configuration manager (yes/no) | No | |
| Usage of the AXMEDIS Error Manager (yes/no) | No | |
| Major Problems not solved | No | |
| Major pending requirements | Object data updates are not possible, registration of metadata about composed objects is not possible, status of the requestor is not checked, object hash is not signed, communication is not using a protected channel | |
| | | |
| 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.) |
| generateAxoid | axcs-db-interface | Protected (SSL) |
| registration | | |
| | | |
| | | |
| Formats Used | Shared with | format name or reference to a |

| | | |
|---|---|---|
| | | section |
| AXMEDIS prefixes | All AXCS Web Services | See section 20 |
| AXMEDIS ID | All AXCS Web services | See section 23 |
| | | |
| | | |
| Protocol Used | Shared with | Protocol name or reference to a section |
| AXCS object registration protocol | | AXCSObjectRegistration (see section 37) |
| | | |
| | | |
| | | |
| Used Database name | | |
| AXCSRegCert | | |
| AXCSObjectsID | | |
| | | |
| | | |
| | | |
| 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 |
| UUID generation | Jug (Doomdark) | |
| MySQL JDBC driver | Connector/J 3.1.10 | GPL, but commercial licenses are available |
| | | |
| | | |
| | | |
| | | |

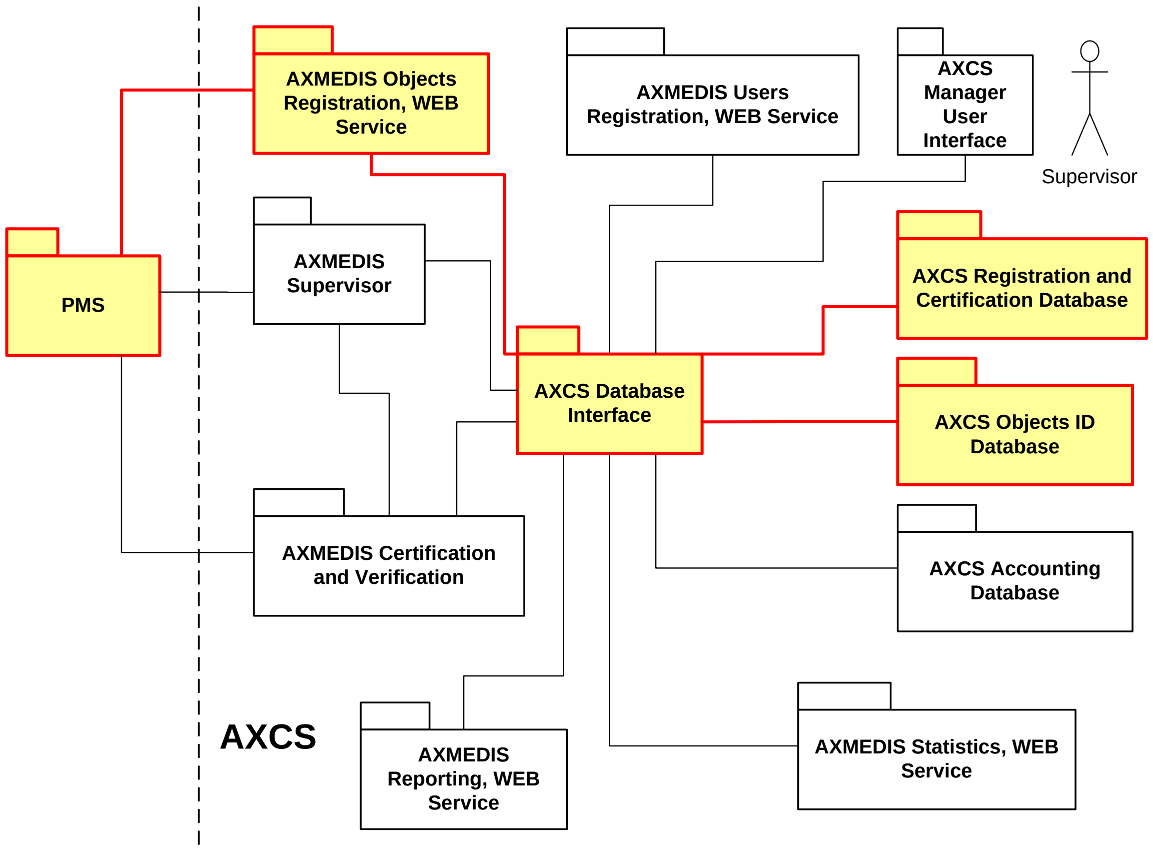
## 7.1 General Description of the Module

All AXMEDIS objects must be registered by an AXCS.
It is important to distinguish between object registration and object insertion in the AXMEDIS database (AXDB). Object registration is about insertion of metadata about the object in the AXCSObjectsID database of an AXCS: no content is inserted in that database, the object is never transferred. Object insertion in the AXDB is about transferring the content (i.e. the object itself) to insert it in the AXDB, which is not part of any AXCS.
An object can be inserted in the AXDB only after it has been registered, i.e. only after AXMEDIS system knows about it. The registration of metadata (in the following named shortly “registration of the object” for sake of simplicity) is a preliminary operation, required to perform the insertion of the content in the AXDB, since the registration assigns a unique identifier to that object and the identifier is needed to do the content insertion.

The registration of an object is a process composed by two different phases: the assignment of an identifier (called AXOID) to the object and the real insertion of metadata in the AXCSObjectsID database (please note that each date is considered in the format yyyy-mm-ddThh:mm:ss, where “T” is used as separator between

date and time and the time is assumed to be referred to GMT+0). Upon insertion, object's hash (i.e. the hash of the block composed by the resource plus the metadata) is signed by the AXCS and returned; in this way it is always guaranteed that the metadata associated to a resource are those effectively used during registration.
These tow phases are completely separated, so that an ID can be assigned to an object some time before it gets registered. This allows content creators to create a new object, obtain an ID for it and then continue to work with the object as long as they desire. This also allows the creation of composed objects; in fact when a composed object is to be registered, all the composing objects must already have unique IDs since their registration is required to happen before the registration of the composed one.

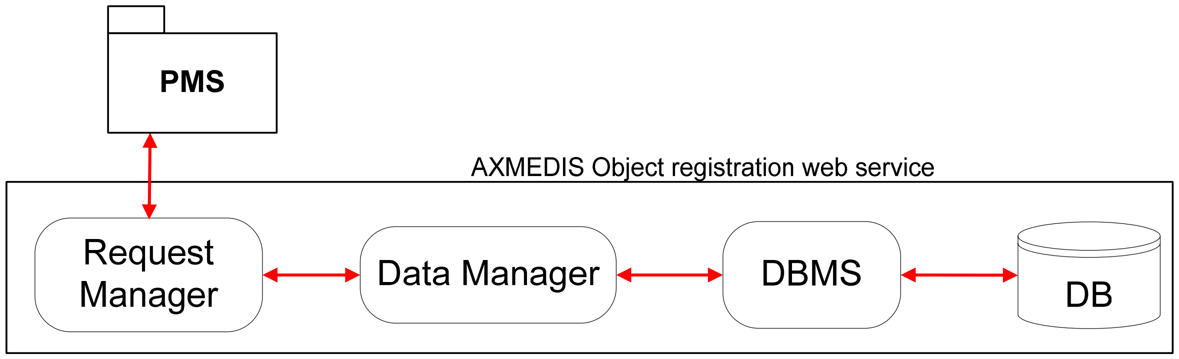
The Object Registration Web Service is the AXCS module that is put in charge for both assigning new AXOIDs and registering object metadata, inserting them in the AXCS Objects ID Database. Finally, the AXCS signs and returns the object hash
The following figure shows which AXCS modules are involved in the object registration process:


AXMEDIS Objects Registration, WEB Service
AXMEDIS Users Registration, WEB Service
AXCS Manager User Interface
Supervisor
PMS
AXMEDIS Supervisor
AXCS Registration and Certification Database
AXCS Database Interface
AXCS Objects ID Database
AXMEDIS Certification and Verification
AXCS Accounting Database
AXCS
AXMEDIS Reporting, WEB Service
AXMEDIS Statistics, WEB Service

Please note that the AXCS Registration and Certification database is involved even if it is not directly related to the object registration task; it is only used to verify the credentials of the user requesting the new registration or the update of data already present in the database.

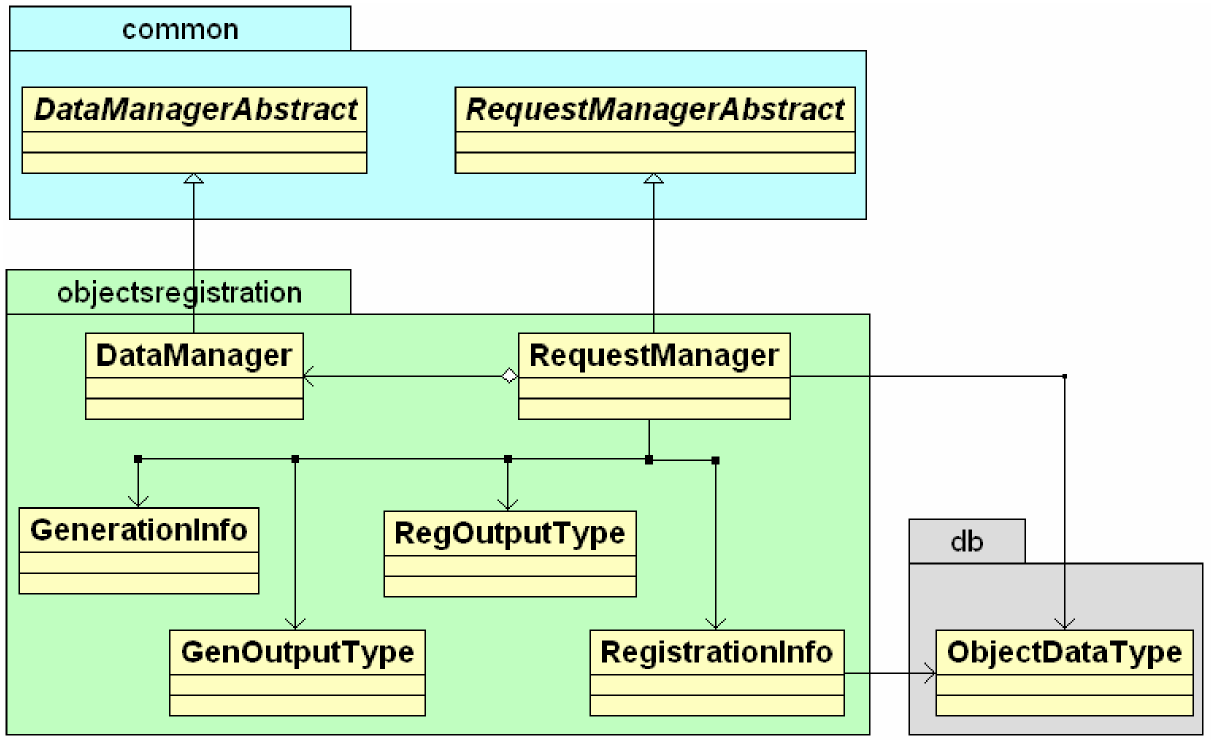
It can be identified the following logical decomposition in the structure of the Object Registration Web Service:

- **Request Manager**: this component receives requests from creators through the PMS and prepares them to be processed by the Data Manager. The communication channel connecting PMS and Request Manager is protected using a secure protocol (for instance SSL). It implements the interface with requestors and manage the whole application.
- **Data Manager**: this component receives data from Request Manager and inserts it in the AXCS Objects ID Database. If necessary elaborates and fits data before insert it in the database. The database management is performed using the related API (AXCS-DB-INTERFACE API).



## 7.2 Module Design in terms of Classes

The logic at the base of the registration mechanism has been subdivided in some classes in order to separate concerns and to make it easier to reuse some functionalities in other modules. The following figure shows the class diagram of the AXCS Object registration web service (the name of the packages have been truncated to keep the figure clear):



## 7.3 User interface description

No user interface is available.

## 7.4 Technical and Installation information

Since the AXCS Users Registration is a web service, it has to be deployed on the chosen platform (ie Tomcat/AXIS, Tomcat/JWSDP, SAS, etc).

| | |
|---|---|
| References to other major | Axcs-db-interface, AXCSRegCert database, AXCSObjectsID database |

| | |
|---|---|
| components needed | |
| Problems not solved | NONE |
| Configuration and execution context | DEPLOY PROCEDURE:<br>1. Use the AdminClient utility that comes with Axis distribution to deploy this web services with the deploy_regs.wsdd file supplied.<br>2. You need also to create a directory called "axcs" (lower case). Place this directory in one of the following location (depending on the OS):<br><br>WINDOWS: C:\axcs<br>*NIX: /axcs<br><br>3. In the axcs directory create a new text file named "axcsdb.ini" (lower case) in which you have to specify location of the AXCS databases and authentication data to access these databases.<br>The file has to be written in the following format.<br><br>AXCSAccounting = jdbc:mysql://<location_of_AXCSAccouning_db>/AXCSAccounting<br>AXCSRegCert = jdbc:mysql://<location_of_AXCSRegCert_db>/AXCSRegCert<br>AXCSObjectsID = jdbc:mysql://<location_of_AXCSObjectsID_db>/AXCSObjectsID<br><br>AXCSDbUser = <user_name><br>AXCSDbPassword = <password><br><br>[dbLog = <log_file_name_with_path>]<br><br>The last line is optional. It lets you specify a logging file in which errors can be written during execution. This file can be placed anywhere you like, but the complete path has to be specified, escaping backslash characters (escape character is a backslash itself). If the last line is not supplied, no log will be created.<br>All data can be specified in any order and white spaces do not matter.<br>Please note that the autentication data have to be the same for all the three databases.<br><br>For instance, if the databases would be hosted on a machine called "MyHost", autentication data would be username: "User" and password: "Pw" and logging file would be "C:\axcs\log_file.txt", the file would appear something similar to:<br><br>AXCSAccounting = jdbc:mysql://MyHost/AXCSAccounting<br>AXCSRegCert = jdbc:mysql://MyHost/AXCSRegCert<br>AXCSObjectsID = jdbc:mysql://MyHost/AXCSObjectsID |

| | |
|---|---|
| | AXCSDbUser = User<br>AXCSDbPassword = Pw<br><br>dbLog = C:\\axcs\\log_file.txt<br><br><br>UNDEPLOY PROCEDURE<br>Use the AdminClient utility that comes with Axis distribution to undeploy this web services with the undeploy_regs.wsdd file supplied.<br><br><br>Platform used in development:Tomcat 5.5 / Axis 1.3, JRE 5.0 |

## 7.5 Draft User Manual

Since the AXCS User Registration is a web service, the public methods it provides should be called using a web service client application. The web service client should call the needed method giving the necessary input parameters and receiving the results. The web service client applications should be implement to use the “Document” style and “Literal“ use.
The requestor calls the “generateAxoid” method, providing his user name and password and a temporary id (used to identify the object inside his factory). If his credentials are correct, he obtains the access to the system and a new AXOID is generated and returned back.
When the creators thinks the object is ready for registration, he calls the “registration” method providing his user name and password, the object metadata he is requesting registration for and the object’s hash. If his credentials are correct, he obtains the access to the system, the sent object metadata are checked and stored in the appropriate database tables and the hash is signed and returned. At the end the object is registered in the system.

## 7.6 Examples of usage

A simple Java client application is provided as example of usage. It is available at https://cvs.axmedis.org/repos/WebServices/axcs-oid-generator/source/client_stubs/original_code/org/axmedis/axcs/ws/objectsmetadatamanager/ClientRegistrator.java

## 7.7 Integration and compilation issues

Since the AXCS Objects Registration is a web service, it should work on every supported platform. It has been developed and tested with Tomcat 5.5 with AXIS 1.3 and Java 5. Interoperability is granted by the respect of the specification included in the related WSDL, which is compliant with the WS-I Basic Profile. This has been achieved realising the web service in “Document” style and “Literal” use.
To deploy the web service it is enough to place Jug and Connector/J libraries in the “lib” directory included in the Axis subtree.

## 7.8 Configuration Parameters

| Config parameter | Possible values |
|---|---|
| NONE | NONE |

## 7.9 Errors reported and that may occur

| Error code | Description and rationales |
|---|---|

|   |   |
|---|---|
| 1 | The required operation failed. More details are provided in the error message returned by the service |

## 7.10 Formal description of algorithm Object Registration

The **Request Manage**r class implements the following functions:

- Accepting authentication data, needful to verify Creators credentials and make them access the system
- Accepting Objects registration data and preparing them to be transferred to Data Manager
- Reply to requestor with the most appropriate message (on the basis of Data Manager responses)

The **Data Manager** class implements the following functions:

- Receiving verification requests and data from the Request Manager
- Accessing the database (using the AXCS-DB-INTERFACE API) to verify Distributors Credential information
- Accessing the database to store received Objects registration data
- Reply to Request Manager according to the performed actions

| **RequestManager** | |
|---|---|
| Method | generateAxoid |
| Description | generates a definitive AXOID to be used in the registration process |
| Input parameters | GenerationInfo genInfo – a data structure containing the nickname and password of the requesting creator and a temporary identifier |
| Output parameters | GenOutputType **-** The result output parameter is a data structure containing three fields:<br>• “resultStatus”: it is set to 0 if the registration is successful otherwise is set to 1.<br>• “axoid”: this field returns the definitive AXOID in case of success, a *null* value otherwise<br>• “errorMessage”: this filed returns the error message in case of failure, a *null* value otherwise |

| **RequestManager** | |
|---|---|
| Method | registration |
| Description | It collects requestor credentials needful to access the system and uses the verifyLogin() (a DataManager method) to verify requestor credentials. It also collects object metadata (regInfo) provided by the requestor and uses the AXCS-DB-INTERFACE API methods provided by AXCS Database Interface to insert received information into database and to provide the result to the requestor. |
| Input parameters | RegistrationInfo regInfo – a data structure containing the nickname and password of the requesting creator and a nested data structure containing the data to be inserted/updated |
| Output parameters | RegOutputType – The result output parameter is a data structure containing three fields:<br>• “resultStatus”: it is set to 0 if the registration is successful otherwise is set to 1.<br>• “signature”: this field returns the object’s hash signed by the AXCS in case of success, a *null* value otherwise<br>• “errorMessage”: this filed returns the error message in case of failure, a *null* value otherwise |

| **RequestManager** | |
|---|---|
| Method | dataPrepare |
| Description | It prepares the Object registration data received from the Creator by the web service to be stored in DB. It uses an UUID generator to get the definitiveID (AXOID) to be inserted in the database and to be sent to requestor |
| Input parameters | ObjectDataType data – the data to be prepared |
| Output | ObjectDataType – data prepared to be stored in the database |

| | |
|---|---|
| parameters | |

| RequestManager | |
|---|---|
| Method | encryptComm |
| Description | It encrypts clear data with the provided asymmetric key (asymKey) |
| Input parameters | String asymKey – the key to be used for encryption<br>String clearData – data to be encrypted |
| Output parameters | String – encrypted data |

| RequestManager | |
|---|---|
| Method | decryptComm |
| Description | It decrypts clear data with the provided asymmetric key (asymKey) |
| Input parameters | String asymKey – the key to be used for encryption<br>String clearData – data to be encrypted |
| Output parameters | String – decrypted data |

| DataManager | |
|---|---|
| Method | verifyLogin |
| Description | It accesses AXCS Registration and Certification Database to verify if the couple NickName and Password is the same provided through the registration process. It also retrieves the user public key stored in the DB.<br>If this method returns *null* it means the user is not trusted (is not present in the database pertinent table).<br>Note: the password is encrypted both the one received as input from the method and the one stored in DB. The equality check is made between encrypted strings |
| Input parameters | String nickName – the nickname of a business user entitled to register new users or to update users data<br>String password – the password of a business user entitled to register new users or to update users data |
| Output parameters | UserLoginData – a data structure containing the AXUID and the public key of the user requesting the registration/update operation in case of success, a *null* value otherwise |

| DataManager | |
|---|---|
| Method | saveObjectData |
| Description | Stores received data in the AXCS Object ID Database |
| Input parameters | Boolean replace – it specifies whether to perform an insert or an update<br>ObjectDataType objData– the password of a business user entitled to register new users or to update users data |
| Output parameters | String – the axoid of the object whose metadata have been inserted/updated in case of success, a *null* values otherwise |

Note that two encrypting/decrypting methods (DecryptComm, EncryptComm) have been introduced to enforce the encrypting robustness. In fact we can suppose to use an encrypted protocol (like SSL), but we can enforce encryption robustness (and therefore security) encrypting ourselves data too using a Public/Private key paradigm. You have to remember that the Distributor public key is stored in AXCS Registration and Certification Database.

In order to generate an object ID that is unique in the whole system, it has been considered to use a standard algorithm: the UUID generator algorithm.
UUID is an identifier that is unique across both space and time, with respect to the space of all UUIDs. A UUID can be used for multiple purposes, from tagging objects with an extremely short lifetime, to reliably identifying very persistent objects across a network. The generation of UUIDs does not require a registration

authority for each single identifier. Instead, it requires a unique value over space for each UUID generator. This spatially unique value is specified as an IEEE 802 address, which is usually already applied to network-connected systems. This 48-bit address can be assigned based on an address block obtained through the IEEE registration authority. This UUID specification assumes the availability of an IEEE 802 address. The UUID consists of a record of 16 octets and must not contain padding between fields. The total size is 128 bits.

More information about UUID generation can be found at the following web address:

http://www.opengroup.org/onlinepubs/9629399/apdxa.htm

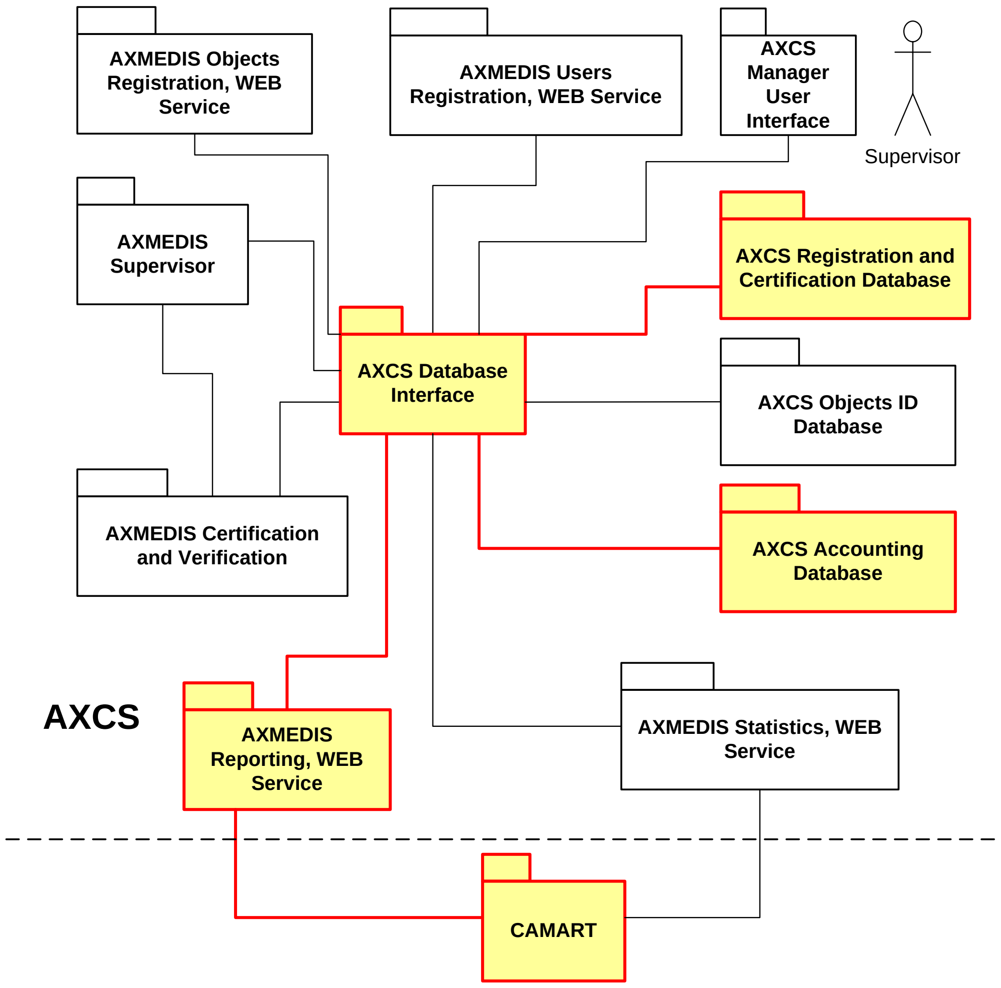
# 8 AXCS Reporting Web Service (DSI)

| Module/Tool Profile | | |
|---|---|---|
| AXCS Reporting Web Service | | |
| Responsible Name | Chellini, Martini | |
| Responsible Partner | DSI | |
| Status (proposed/approved) | Proposed | |
| Implemented/not implemented | Implemented | |
| Status of the implementation | In refinement | |
| Executable or Library/module (Support) | Web service | |
| Single Thread or Multithread | Multithread | |
| Language of Development | Java | |
| Platforms supported | Anyone on which can be installed a servlet container and a JRE 1.5.0 (5.0). It has been tested using Tomcat 5.5, Axis 1.3, JRE 1.5.0_06 | |
| Reference to the AXFW location of the source code demonstrator | https://cvs.axmedis.org/repos/Framework/source/axcs/axcs-framework/src/org/axmedis/axcs/services | |
| Reference to the AXFW location of the demonstrator executable tool for internal download | https://cvs.axmedis.org/repos/WebServices/axcs-reporting | |
| 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://flauto.dsi.unifi.it:8080/axis/services/AXCSReporting | |
| Test cases (present/absent) | present | |
| Test cases location | https://cvs.axmedis.org/repos/WebServices/axcs-reporting/bin/org/axmedis/axcs/ws/reporting/ClientReportingWS.class | |
| 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 | Status of the requestor is not checked, communication is not using a protected channel | |
| | | |
| 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.) |
| acceptRequest | axcs-db-interface | Protected (SSL) |
| | | |
| | | |
| | | |
| Formats Used | Shared with | format name or reference to a section |
| AXMEDIS prefixes | All AXCS Web services | See section 20 |
| AXMEDIS ID | All AXCS Web services | See section 23 |

| | | |
|---|---|---|
| | | |
| Protocol Used | Shared with | Protocol name or reference to a section |
| AXCS reporting protocol | | AXCSReporting (see section 38) |
| | | |
| | | |
| | | |
| Used Database name | | |
| AXCSRegCert | | |
| AXCSAccounting | | |
| | | |
| | | |
| | | |
| 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 |
| MySQL JDBC driver | Connector/J 3.1.10 | GPL, but commercial licenses are available |
| | | |
| | | |
| | | |
| | | |
| | | |

## 8.1 General Description of the Module

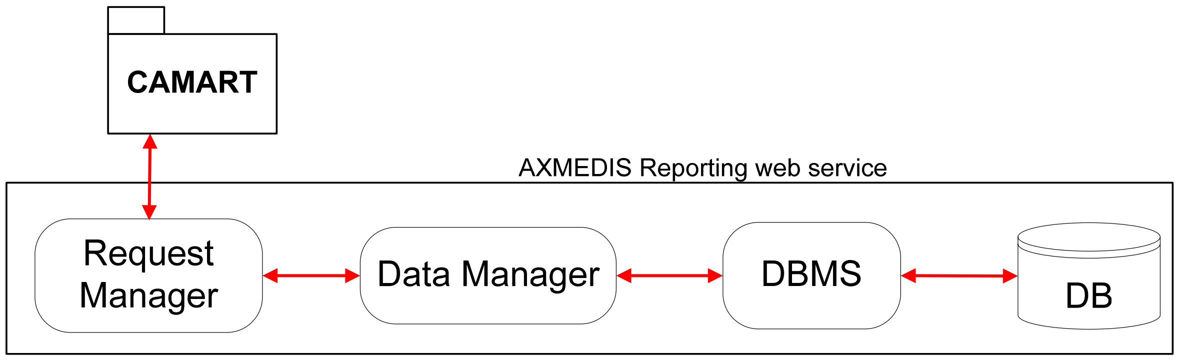
The object usage accounting activity is a fundamental asset in the AXMEDIS system. Distributors, Creators, Integrators, Collecting Societies have to know all the operations performed over their pertinent objects to receive the correctness fee from the object users. This is a vital activity if we want to guarantee the owner rights are respected. All the accounting needful information are stored in AXCS Accounting Database. The AXMEDIS Reporting unit is the high-level interface to this database that provides a sequence of services needful to retrieve information in a correct way, with no errors and with respect to the users privacy and the companies/societies confidential data; it reports only data related to usages of objects pertinent to the requestor (please note that each date is considered in the format yyyy-mm-ddThh:mm:ss, where "T" is used as separator between date and time and the time is assumed to be referred to GMT+0).
The AXMEDIS Reporting Web Services deals with CAMART (Core accounting Manager and Reporting Tool). CAMART is a sort of client application, used by requestors, that queries AXMEDIS Reporting Web Services to retrieve all the needful information. The CAMART can be considered as the client part of the reporting system and the AXMEDIS Reporting Web Service can be considered as the server part.
The following figure shows which AXCS modules are involved in the reporting process:


AXMEDIS Objects Registration, WEB Service
AXMEDIS Users Registration, WEB Service
AXCS Manager User Interface
Supervisor
AXMEDIS Supervisor
AXCS Registration and Certification Database
AXCS Database Interface
AXCS Objects ID Database
AXMEDIS Certification and Verification
AXCS Accounting Database
AXCS
AXMEDIS Reporting, WEB Service
AXMEDIS Statistics, WEB Service
CAMART

Please note that the AXCS Registration and Certification database is involved even if it is not directly related to the object registration task; it is only used to verify the credentials of the user requesting the new registration or the update of data already present in the database.

It can be identified the following logical decomposition in the structure of the Reporting Web Service:

- **Request Manager**: this component receives reporting requests from CAMART and prepares them to be processed by the Data Manager. The communication channel connecting CAMART and Request Manager is protected using a secure protocol (for instance SSL). It implements the interface with requestors and manage the whole application.
- **Data Manager**: this component receives data from Request Manager and retrieves requested data from the AXCS Accounting Database. If necessary elaborates and fits retrieved data before returning them back. The database management is performed using the related API (AXCS-DB-INTERFACE API).



### 8.1.1 Action log received by AXCS

This section describes the data received by the AXCS when an event report is generated; these are also the data returned by the AXCS Reporting Web Service.

The information received by the AXCS could be:

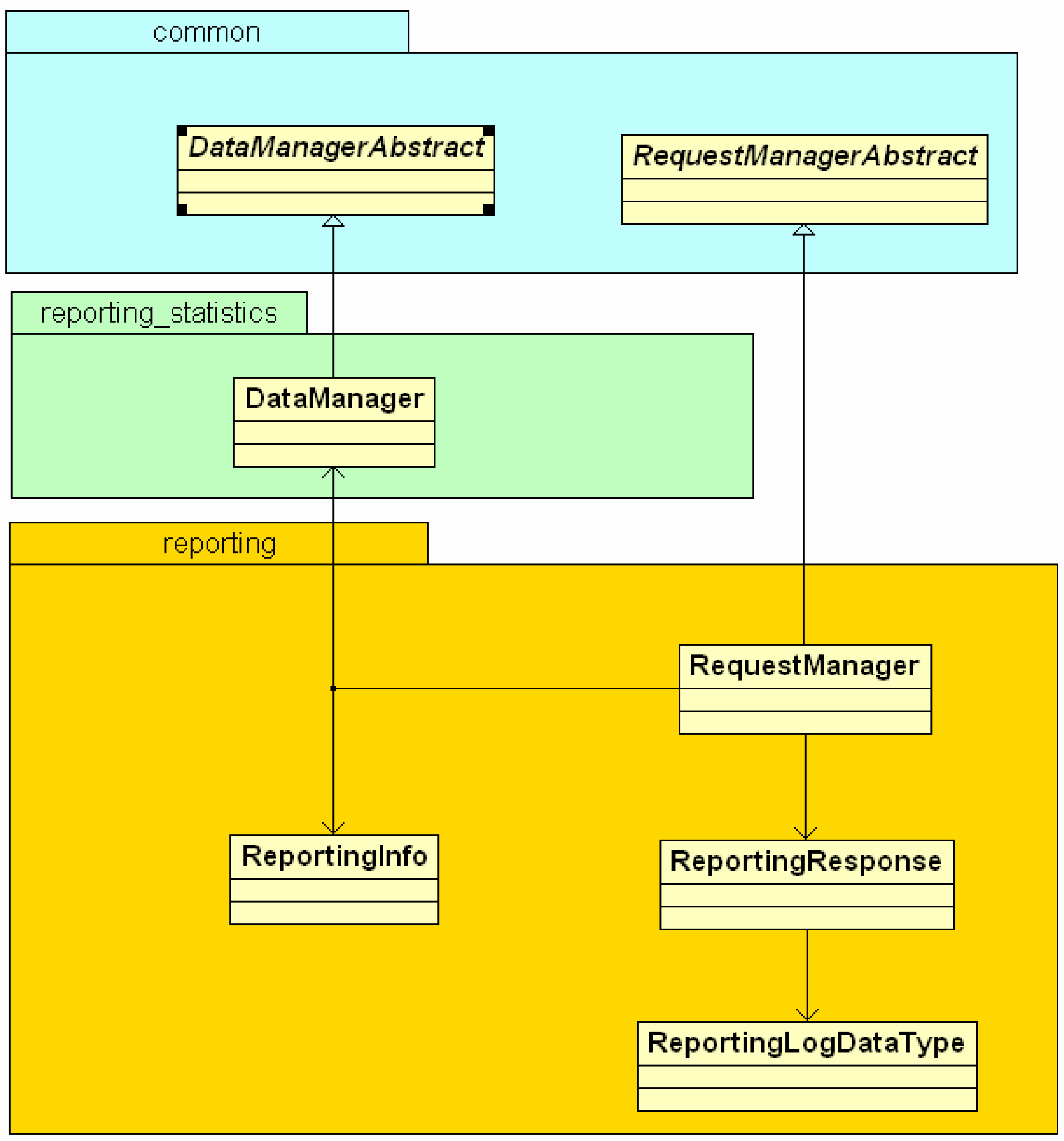
- An ActionLog entity or a list of ActionLog entities
- A SupervisorInputData entity

The ActionLog entity has the following elements:

- LogID : Registration ID in Action-Log Registry. Identifies an ActionLog.
- AXOID: The action log references to a certain Axmedis Object, that it is identified by its ID.
- ObjectVersion: Reference to an Axmedis Object version. The action log references to a certain version of the object related.
- ProtectionStamp: Indicates the way to protect the related object.
- AXWID: Indicates the Work ID. This element is not necessary.
- AXDOM: Indicates the User Domain, if the user has a domain related.
- AXUID: The action log references to a certain User, that it is identified by its ID.
- AXDID: Indicates the pertinent Object Distributor ID.
- AXCID: Indicates the pertinent Object Creator ID.
- OwnerName: Indicates the pertinent Object Owner.
- AXTID: The action log references to a certain certified tool, that it is identified by its ID.
- AXLID: Indicates the pertinent License ID.
- Location: Indicates the nation.
- OperationDetailsIDPk : Reference to an operation details.
- OperationIDPk: Reference to an operaction.
- RegistrationTimestamp: Time of the operation Registration.
- ExecutionTimestamp: Time of the operation Execution.
- InstantLastFPPA: Last Fingerprint of Performed Actions.
- EstimatedHWFingerprint: Indicates the estimated HW Fingerprint of the related terminal.

## 8.2 Module Design in terms of Classes

The logic at the base of the reporting mechanism has been subdivided in some classes in order to separate concerns and to make it easier to reuse some functionalities in other modules. The following figure shows the class diagram of the AXCS Reporting web service (the name of the packages have been truncated to keep the figure clear):



## 8.3 User interface description

No user interface is available.

## 8.4 Technical and Installation information

Since the AXCS Users Registration is a web service, it has to be deployed on the chosen platform (ie Tomcat/AXIS, Tomcat/JWSDP, SAS, etc).

| | |
|---|---|
| References to other major components needed | Axcs-db-interface, AXCSRegCert database, AXCSAccounting database |
| Problems not solved | NONE |
| Configuration and execution context | DEPLOY PROCEDURE:<br>1. Use the AdminClient utility that comes with Axis distribution to deploy this web services with the deploy_regs.wsdd file supplied. |

| | |
|---|---|
| | 2. You need also to create a directory called "axcs" (lower case). Place this directory in one of the following location (depending on the OS):<br><br>WINDOWS: C:\axcs<br>*NIX: /axcs<br><br>3. In the axcs directory create a new text file named "axcsdb.ini" (lower case) in which you have to specify location of the AXCS databases and authentication data to access these databases.<br>The file has to be written in the following format.<br><br>AXCSAccounting = jdbc:mysql://<location_of_AXCSAccouning_db>/AXCSAccounting<br>AXCSRegCert = jdbc:mysql://<location_of_AXCSRegCert_db>/AXCSRegCert<br>AXCSObjectsID = jdbc:mysql://<location_of_AXCSObjectsID_db>/AXCSObjectsID<br><br>AXCSDbUser = <user_name><br>AXCSDbPassword = <password><br><br>[dbLog = <log_file_name_with_path>]<br><br>The last line is optional. It lets you specify a logging file in which errors can be written during execution. This file can be placed anywhere you like, but the complete path has to be specified, escaping backslash characters (escape character is a backslash itself). If the last line is not supplied, no log will be created.<br>All data can be specified in any order and white spaces do not matter.<br>Please note that the autentication data have to be the same for all the three databases.<br><br>For instance, if the databases would be hosted on a machine called "MyHost", autentication data would be username: "User" and password: "Pw" and logging file would be "C:\axcs\log_file.txt", the file would appear something similar to:<br><br>AXCSAccounting = jdbc:mysql://MyHost/AXCSAccounting<br>AXCSRegCert = jdbc:mysql://MyHost/AXCSRegCert<br>AXCSObjectsID = jdbc:mysql://MyHost/AXCSObjectsID<br><br>AXCSDbUser = User<br>AXCSDbPassword = Pw<br><br>dbLog = C:\\axcs\\log_file.txt |

| | |
|---|---|
| | UNDEPLOY PROCEDURE<br>Use the AdminClient utility that comes with Axis distribution to undeploy this web services with the undeploy_regs.wsdd file supplied.<br><br><br>Platform used in development:Tomcat 5.5 / Axis 1.3, JRE 5.0 |

## 8.5 Draft User Manual

Since the AXCS User Registration is a web service, the public methods it provides should be called using a web service client application. The web service client should call the needed method giving the necessary input parameters and receiving the results. The web service client applications should be implement to use the “Document” style and “Literal“ use.
The requestor calls the “acceptRequest” method, providing his user name and password and the criteria to be used to filter all the action logs (only those relative to him), e.g. a couple of dates specifying the period of interest. If his credentials are correct, he obtains the access to the system and the required data are retrieved from the appropriate database tables.
As for the criteria that can be used to filter data, they are to be expressed in standard SQL syntax as a where clause (without the WHERE keyword) using only the following operators:

equal (=), not equal (<>), logical and (AND), logical or (OR)

to filter with respect to these fields: any identifier (*AXUID*, *AXDID*, *AXOID*, ecc.), *Location*, *Operation*, *RegistrationTimeStamp*, *ExecutionTimeStamp*
In case that a filtering with respect to dates is needed (i.e. with respect to *RegistrationTimeStamp* and/or *ExecutionTimeStamp*), the following additional operators are allowed:

greater than (>), greater than or equal (>=), less than (<), less than or equal (<=).
Eventually, an empty caluse can be used if no filtering is desired.

## 8.6 Examples of usage

A simple Java client application is provided as example of usage. It is available at https://cvs.axmedis.org/repos/WebServices/axcs-reporting/source/org/axmedis/axcs/ws/reporting/ClientReportingWS.java

## 8.7 Integration and compilation issues

Since the AXCS Reporting is a web service, it should work on every supported platform. It has been developed and tested with Tomcat 5.5 with AXIS 1.3 and Java 5. Interoperability is granted by the respect of the specification included in the related WSDL, which is compliant with the WS-I Basic Profile. This has been achieved realising the web service in “Document” style and “Literal” use.
To deploy the web service it is enough to place Connector/J library in the “lib” directory included in the Axis subtree.

## 8.8 Configuration Parameters

| **Config parameter** | **Possible values** |
|---|---|
| NONE | NONE |

## 8.9 Errors reported and that may occur

| Error code | Description and rationales |
|---|---|
| 1 | The required operation failed. More details are provided in the error message returned by the service |

## 8.10 Formal description of algorithm Reporting

The **Reporting Request Manager** class implements the following functions:

- Accepting authentication data, needful to verify CAMART credentials and make it access the system
- Accepting reporting requests and preparing them to be transferred to Reporting Data Manager
- Reply to requestor with the most appropriate data or error messages (on the basis of Reporting Data Manager responses)

The **Reporting Data Manager** class implements the following functions:

- Receiving requests and data from the Reporting Request Manager
- Accessing the database (using the AXCS-DB-INTERFACE API) to verify CAMART Users credentials information
- Accessing the database to retrieve data on the basis of CAMART Users requests (received from Reporting Request Manager) and CAMART Users profile (note that requestors can access only pertinent data)
- Reply to Reporting Request Manager according to the performed actions and obtained results

| **RequestManager** | |
|---|---|
| Method | acceptRequest |
| Description | It is the only public method of this web service. It collects CAMART credentials needful to access the system and uses the “verifyLogin” (a DataManager method) to verify requestor credentials. It also collects the CAMART query (in a field called CAMARTQuery). CAMARTQuery contains criteria to be used to filter data returned.<br>This method uses private methods described below to perform its tasks and to answer to requestor. |
| Input parameters | ReportingInfo repInfo – a data structure containing the nickname and password of the requesting distributor and a CAMARTQuery |
| Output parameters | ReportingResponse – The result output parameter is a data structure containing three fields:<br>• “resultStatus”: it is set to 0 if the registration is successful otherwise is set to 1.<br>• “logDataTypes”: this field returns the requested information as a set of structured data in case of success, a *null* value otherwise<br>“errorMessage”: this filed returns the error message in case of failure, a *null* value otherwise |

| **RequestManager** | |
|---|---|
| Method | createQuery |
| Description | It prepares a suitable query (adapting it at the specific database engine used) to retrieve requested data from the database |
| Input parameters | String b2bUserID – AXUID of the user requesting to retrieve data<br>String camartQuery – criteria to be used to filter the requested data |
| Output parameters | String – a query to be performed to retrieve the requested data |

| **RequestManager** | |
|---|---|
| Method | requestElaborate |
| Description | “It elaborates CAMART requests. The CAMARTQuery field is managed (with the “createQuery” method) to extract the needful data to perform the query on database. The query is performed by “queryExecuter”, a DataManager method. It also elaborates query results to be sent to the requesting CAMART |

| | |
|---|---|
| Input parameters | String b2bUserID – AXUID of the user requesting to retrieve data<br>String camartQuery – criteria to be used to filter the requested data |
| Output parameters | ReportingResponse – The result output parameter is a data structure containing three fields:<br>• "resultStatus": it is set to 0 if the registration is successful otherwise is set to 1.<br>• "logDataTypes": this field returns the requested information as a set of structured data in case of success, a *null* value otherwise<br>"errorMessage": this filed returns the error message in case of failure, a *null* value otherwise |

| **RequestManager** | |
|---|---|
| Method | encryptComm |
| Description | It encrypts clear data with the provided asymmetric key (asymKey) |
| Input parameters | String asymKey – the key to be used for encryption<br>String clearData – data to be encrypted |
| Output parameters | String – encrypted data |

| **RequestManager** | |
|---|---|
| Method | decryptComm |
| Description | It decrypts clear data with the provided asymmetric key (asymKey) |
| Input parameters | String asymKey – the key to be used for encryption<br>String clearData – data to be encrypted |
| Output parameters | String – decrypted data |

| **DataManager** | |
|---|---|
| Method | verifyLogin |
| Description | It accesses AXCS Registration and Certification Database to verify if the couple NickName and Password is the same provided through the registration process. It also retrieves the user public key stored in the DB.<br>If this method returns *null* it means the user is not trusted (is not present in the database pertinent table).<br>Note: the password is encrypted both the one received as input from the method and the one stored in DB. The equality check is made between encrypted strings |
| Input parameters | String nickName – the nickname of a business user entitled to register new users or to update users data<br>String password – the password of a business user entitled to register new users or to update users data |
| Output parameters | UserLoginData – a data structure containing the AXUID and the public key of the user requesting the registration/update operation in case of success, a *null* value otherwise |

| **DataManager** | |
|---|---|
| Method | queryExecuter |
| Description | It performs the query on the basis of data elaborated by "acceptRequest", a method of the RequestManager |
| Input parameters | String queryConstraints – the cquery to be used to retrieve the requested data |
| Output parameters | ResultSet – a set containing requested datain case of success, an empty set otherwise |

Note that two encrypting/decrypting methods (DecryptComm, EncryptComm) have been introduced to enforce the encrypting robustness. In fact we can suppose to use an encrypted protocol (like SSL), but we can enforce encryption robustness (and therefore security) encrypting ourselves data too using a Public/Private key paradigm. You have to remember that the Distributor public key is stored in AXCS Registration and Certification Database.

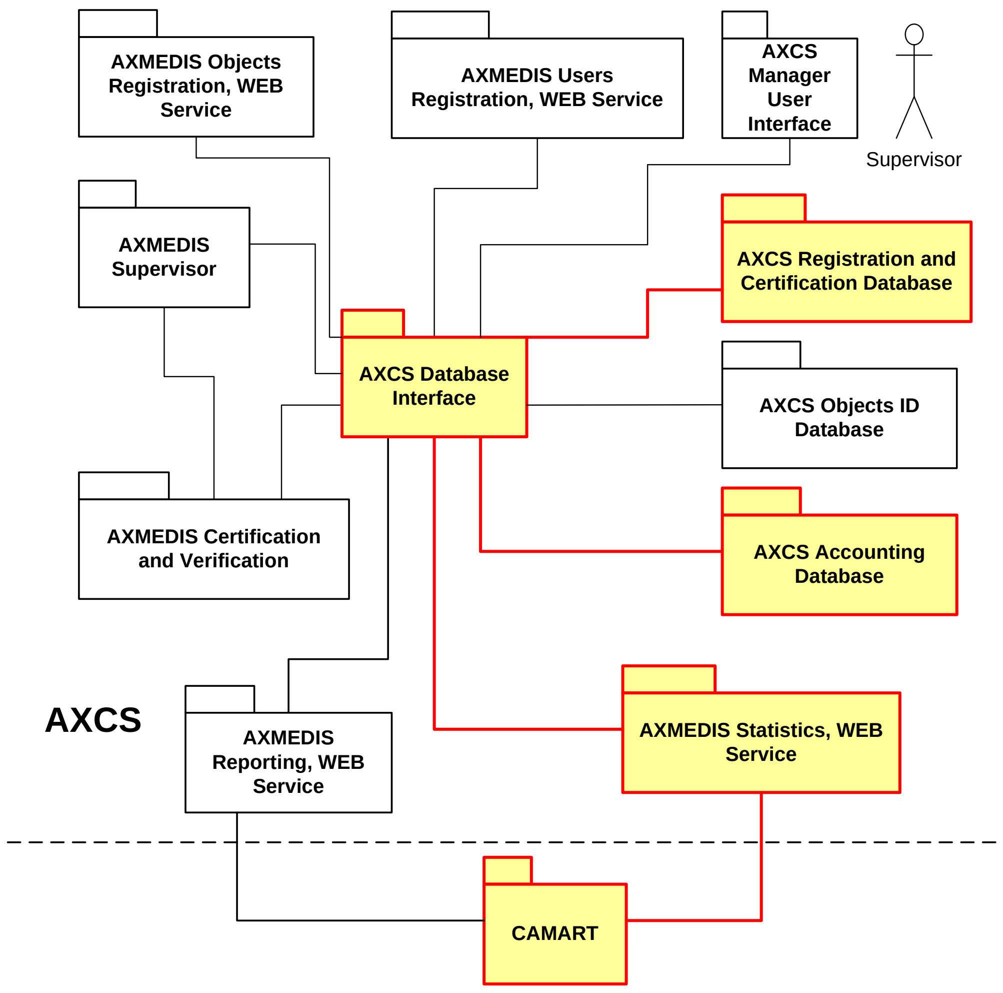
# 9 AXCS Statistics Web Service (DSI)

| Module/Tool Profile | | |
|---|---|---|
| AXCS Statistics Web Service | | |
| Responsible Name | Chellini, Martini | |
| Responsible Partner | DSI | |
| Status (proposed/approved) | Proposed | |
| Implemented/not implemented | Implemented | |
| Status of the implementation | In refinement | |
| Executable or Library/module (Support) | Web service | |
| Single Thread or Multithread | Multithread | |
| Language of Development | Java | |
| Platforms supported | Anyone on which can be installed a servlet container and a JRE 1.5.0 (5.0). It has been tested using Tomcat 5.5, Axis 1.3, JRE 1.5.0_06 | |
| Reference to the AXFW location of the source code demonstrator | https://cvs.axmedis.org/repos/Framework/source/axcs/axcs-framework/src/org/axmedis/axcs/services | |
| Reference to the AXFW location of the demonstrator executable tool for internal download | https://cvs.axmedis.org/repos/WebServices/axcs-statistics | |
| 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://flauto.dsi.unifi.it:8080/axis/services/AXCSStatistics | |
| Test cases (present/absent) | present | |
| Test cases location | https://cvs.axmedis.org/repos/WebServices/axcs-statistics/bin/org/axmedis/axcs/ws/statistics/ClientStatisticsWS.class | |
| 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 | Status of the requestor is not checked, communication is not using a protected channel | |
| | | |
| 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.) |
| acceptRequest | axcs-db-interface | Protected (SSL) |
| | | |
| | | |
| | | |
| Formats Used | Shared with | format name or reference to a section |
| AXMEDIS prefixes | All AXCS Web services | See section 20 |
| AXMEDIS ID | All AXCS Web services | See section 23 |

| | | |
|---|---|---|
| | | |
| | | |
| Protocol Used | Shared with | Protocol name or reference to a section |
| AXCS statistics protocol | | AXCSStatistics (see section 39) |
| | | |
| | | |
| | | |
| Used Database name | | |
| AXCSRegcert | | |
| AXCSAccounting | | |
| | | |
| | | |
| | | |
| 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 |
| MySQL JDBC driver | Connector/J 3.1.10 | GPL, but commercial licenses are available |
| | | |
| | | |
| | | |
| | | |
| | | |

## 9.1 General Description of the Module

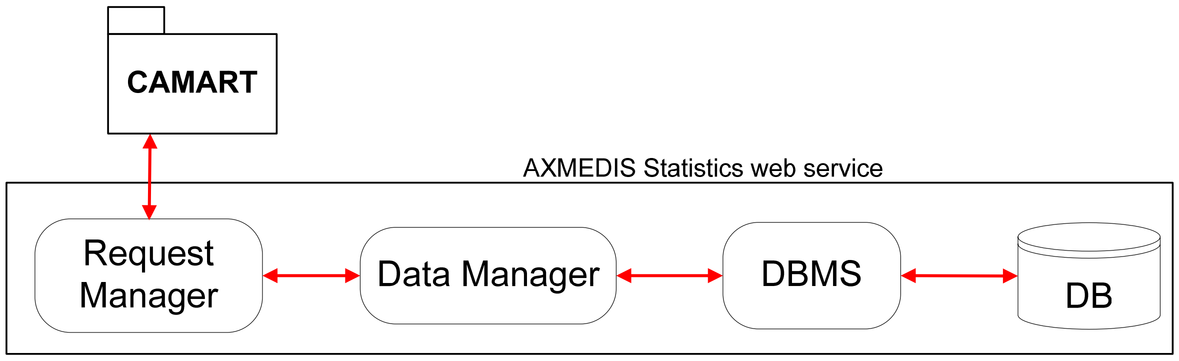
In the AXMEDIS environment, statistics can represent an important asset for the whole system. A Distributor, Creator, Integrator, etc. could be interested in knowing where, when and how an object is used: this interest could be used for commercial and marketing purpose. Knowing usage, distribution and integration statistics could be an important resource for an AXMEDIS subject to improve his business. Therefore has been introduced a statistical tool called AXMEDIS Statistics Web Service which task is retrieving anonymous statistical information from the AXCS Accounting Database. This tool can be queried by all AXMEDIS subjects and produce anonymous statistics concerning objects (please note that each date is considered in the format yyyy-mm-ddThh:mm:ss, where "T" is used as separator between date and time and the time is assumed to be referred to GMT+0).
The AXMEDIS Statistics Analysis Tool deals with CAMART (Core accounting Manager and Reporting Tool). CAMART is a sort of client application, used by requestors, that queries AXMEDIS Statistics Analysis Tool to retrieve all the needful information. The CAMART can be consider the client part of the Statistic Analysis system and the AXMEDIS Statistics Analysis Tool can be consider the server part.
The following figure shows which AXCS modules are involved in the user registration process:


AXMEDIS Objects Registration, WEB Service
AXMEDIS Users Registration, WEB Service
AXCS Manager User Interface
Supervisor
AXMEDIS Supervisor
AXCS Registration and Certification Database
AXCS Database Interface
AXCS Objects ID Database
AXMEDIS Certification and Verification
AXCS Accounting Database
AXCS
AXMEDIS Reporting, WEB Service
AXMEDIS Statistics, WEB Service
CAMART

Please note that the AXCS Registration and Certification database is involved even if it is not directly related to the object registration task; it is only used to verify the credentials of the user requesting the new registration or the update of data already present in the database.

It can be identified the following logical decomposition in the structure of the Statistics Web Service:

- **Statistics Request Manager**: this component receives requests from CAMART, and prepares them to be processed by the Data Manager. The communication channel connecting CAMART and Statistics Request Manager is protected using a secure protocol (for instance SSL). It implements the interface with requestors and manage the whole application.
- **Statistics Data Manager**: this component receives data from Request Manager and retrieves requested data from the AXCS Accounting Database. If necessary elaborates and fits retrieved data before returning them back. The database management is performed using the related API (AXCS-DB-INTERFACE API).



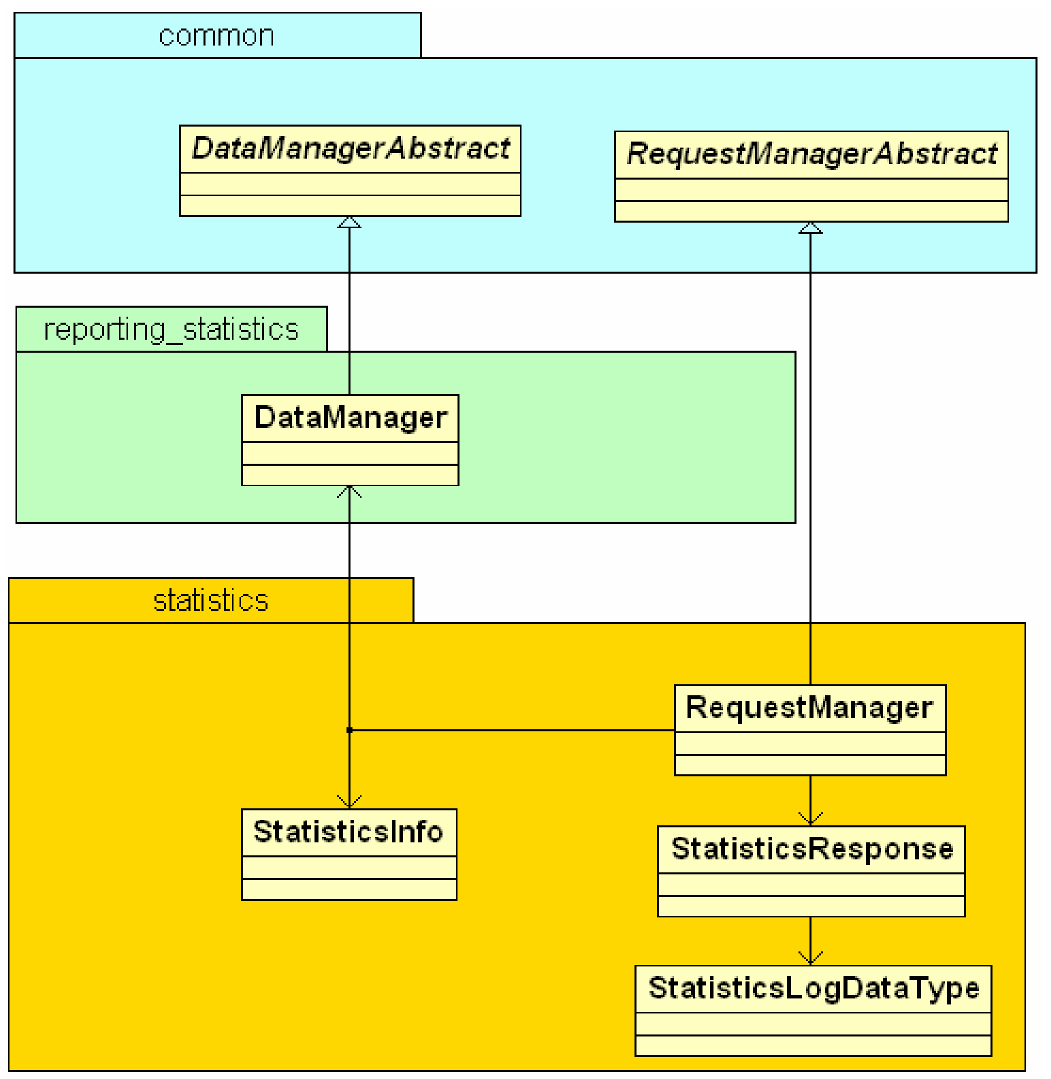
### 9.1.1 Data returned by Statistics Web Service

Similarly to the Reporting web Service, the Statistics Web Service returns data about object usages. Anyway, the difference is that this web service cannot give back only anonymous data. Therefore, data returned by it are a subset of those described in section 8.1.1. Please refer to that section for more details, here is reported only a list of data returned by this web service:

- AXOID
- ObjectVersion
- ProtectionStamp
- AXWID
- AXDOM
- AXDID
- AXCID
- OwnerName
- AXCSID
- Location
- OperationDetailsIDPk
- OperationIDPk
- RegistrationTimestamp
- ExecutionTimestamp

## 9.2 Module Design in terms of Classes

The logic at the base of the statistics mechanism has been subdivided in some classes in order to separate concerns and to make it easier to reuse some functionalities in other modules. The following figure shows the class diagram of the AXCS Reporting web service (the name of the packages have been truncated to keep the figure clear):



## 9.3 User interface description

No user interface is available.

## 9.4 Technical and Installation information

Since the AXCS Users Registration is a web service, it has to be deployed on the chosen platform (e.g. Tomcat/AXIS, Tomcat/JWSDP, SAS, etc).

| | |
|---|---|
| References to other major components needed | Axcs-db-interface, AXCSRegCert database, AXCSAccounting database |
| Problems not solved | NONE |
| Configuration and execution context | DEPLOY PROCEDURE:<br>1. Use the AdminClient utility that comes with Axis distribution to deploy this web services with the deploy_regs.wsdd file supplied.<br>2. You need also to create a directory called "axcs" (lower case). Place this directory in one of the following location (depending on the OS): |

| | |
|---|---|
| | WINDOWS: C:\axcs<br>*NIX: /axcs<br><br>3. In the axcs directory create a new text file named "axcsdb.ini" (lower case) in which you have to specify location of the AXCS databases and authentication data to access these databases.<br>The file has to be written in the following format.<br><br>AXCSAccounting = jdbc:mysql://<location_of_AXCSAccouning_db>/AXCSAccounting<br>AXCSRegCert = jdbc:mysql://<location_of_AXCSRegCert_db>/AXCSRegCert<br>AXCSObjectsID = jdbc:mysql://<location_of_AXCSObjectsID_db>/AXCSObjectsID<br><br>AXCSDbUser = <user_name><br>AXCSDbPassword = <password><br><br>[dbLog = <log_file_name_with_path>]<br><br>The last line is optional. It lets you specify a logging file in which errors can be written during execution. This file can be placed anywhere you like, but the complete path has to be specified, escaping backslash characters (escape character is a backslash itself). If the last line is not supplied, no log will be created.<br>All data can be specified in any order and white spaces do not matter.<br>Please note that the autentication data have to be the same for all the three databases.<br><br>For instance, if the databases would be hosted on a machine called "MyHost", autentication data would be username: "User" and password: "Pw" and logging file would be "C:\axcs\log_file.txt", the file would appear something similar to:<br><br>AXCSAccounting = jdbc:mysql://MyHost/AXCSAccounting<br>AXCSRegCert = jdbc:mysql://MyHost/AXCSRegCert<br>AXCSObjectsID = jdbc:mysql://MyHost/AXCSObjectsID<br><br>AXCSDbUser = User<br>AXCSDbPassword = Pw<br><br>dbLog = C:\\axcs\\log_file.txt<br><br>UNDEPLOY PROCEDURE<br>Use the AdminClient utility that comes with Axis distribution to undeploy this web services with the undeploy_regs.wsdd file supplied. |

| | |
|---|---|
| | Platform used in development:Tomcat 5.5 / Axis 1.3, JRE 5.0 |

## 9.5 Draft User Manual

Since the AXCS User Registration is a web service, the public methods it provides should be called using a web service client application. The web service client should call the needed method giving the necessary input parameters and receiving the results. The web service client applications should be implement to use the “Document” style and “Literal“ use.
The requestor calls the “acceptRequest” method, providing his user name and password and the criteria to be used to filter all the action log, e.g. a couple of dates specifying the period of interest. If his credentials are correct, he obtains the access to the system and the required data are retrieved in an anonymous format from the appropriate database tables.
As for the criteria that can be used to filter data, they are to be expressed in standard SQL syntax as a where clause (without the WHERE keyword) using only the following operators:

equal (=), not equal (<>), logical and (AND), logical or (OR)

to filter with respect to these fields: any identifier (*AXUID*, *AXDID*, *AXOID*, ecc.), *Location*, *Operation*, *RegistrationTimeStamp*, *ExecutionTimeStamp*
In case that a filtering with respect to dates is needed (i.e. with respect to *RegistrationTimeStamp* and/or *ExecutionTimeStamp*), the following additional operators are allowed:

greater than (>), greater than or equal (>=), less than (<), less than or equal (<=).
Eventually, an empty caluse can be used if no filtering is desired.

## 9.6 Examples of usage

A simple Java client application is provided as example of usage. It is available at https://cvs.axmedis.org/repos/WebServices/axcs-statistics/bin/org/axmedis/axcs/ws/statistics/ClientStatisticsWS.java

## 9.7 Integration and compilation issues

Since the AXCS Reporting is a web service, it should work on every supported platform. It has been developed and tested with Tomcat 5.5 with AXIS 1.3 and Java 5. Interoperability is granted by the respect of the specification included in the related WSDL, which is compliant with the WS-I Basic Profile. This has been achieved realising the web service in “Document” style and “Literal” use.
To deploy the web service it is enough to place Connector/J library in the “lib” directory included in the Axis subtree.

## 9.8 Configuration Parameters

| Config parameter | Possible values |
|---|---|
| NONE | NONE |

## 9.9 Errors reported and that may occur

| Error code | Description and rationales |
|---|---|
| 1 | The required operation failed. More details are provided in the error message returned by the service |

## 9.10 Formal description of algorithm Statistics

The **Statistics Request Manager** class implements the following functions:

- Accepting authentication data, needful to verify CAMART credentials and make it access the system
- Accepting requests and preparing them to be transferred to Statistics Data Manager
- Reply to requestor with the most appropriate data or error messages (on the basis of Statistics Data Manager responses)

The **Statistics Data Manager** class implements the following functions:

- Statistics requests and data from the Statistics Request Manager
- Accessing the database (using the AXDB-API) to verify CAMART Users credentials information
- Accessing the database to retrieve statistics data on the basis of CAMART Users requests (received from Statistics Request Manager)
- Reply to Statistics Request Manager according to the performed actions and obtained results

| **RequestManager** | |
|---|---|
| Method | acceptRequest |
| Description | It is the only public method of this web service. It collects CAMART credentials needful to access the system and uses the "verifyLogin" (a DataManager method) to verify requestor credentials. It also collects the CAMART query (in a field called CAMARTQuery). CAMARTQuery contains criteria to be used to filter data returned.<br>This method uses private methods described below to perform its tasks and to answer to requestor. |
| Input parameters | StatisticsInfo statInfo - a data structure containing the nickname and password of the requesting distributor and a CAMARTQuery |
| Output parameters | StatisticsResponse – The result output parameter is a data structure containing three fields:<br>• "resultStatus": it is set to 0 if the registration is successful otherwise is set to 1.<br>• "logDataTypes": this field returns the requested information as a set of structured data in case of success, a *null* value otherwise<br>"errorMessage": this filed returns the error message in case of failure, a *null* value otherwise |

| **RequestManager** | |
|---|---|
| Method | createQuery |
| Description | It prepares a suitable query (adapting it at the specific database engine used) to retrieve requested data from the database |
| Input parameters | String b2bUserID – AXUID of the user requesting to retrieve data<br>String camartQuery – criteria to be used to filter the requested data |
| Output parameters | String – a query to be performed to retrieve the requested data |

| **RequestManager** | |
|---|---|
| Method | requestElaborate |
| Description | "It elaborates CAMART requests. The CAMARTQuery field is managed (with the "createQuery" method) to extract the needful data to perform the query on database. The query is performed by "queryExecuter", a DataManager method. It also elaborates query results to be sent to the requesting CAMART |
| Input parameters | String b2bUserID – AXUID of the user requesting to retrieve data<br>String camartQuery – criteria to be used to filter the requested data |
| Output parameters | ReportingResponse – The result output parameter is a data structure containing three fields:<br>• "resultStatus": it is set to 0 if the registration is successful otherwise is set to 1.<br>• "logDataTypes": this field returns the requested information as a set of structured data in case of success, a *null* value otherwise |

| | |
|---|---|
| | “errorMessage”: this filed returns the error message in case of failure, a *null* value otherwise |

| **RequestManager** | |
|---|---|
| Method | encryptComm |
| Description | It encrypts clear data with the provided asymmetric key (asymKey) |
| Input parameters | String asymKey – the key to be used for encryption<br>String clearData – data to be encrypted |
| Output parameters | String – encrypted data |

| **RequestManager** | |
|---|---|
| Method | decryptComm |
| Description | It decrypts clear data with the provided asymmetric key (asymKey) |
| Input parameters | String asymKey – the key to be used for encryption<br>String clearData – data to be encrypted |
| Output parameters | String – decrypted data |

| **DataManager** | |
|---|---|
| Method | verifyLogin |
| Description | It accesses AXCS Registration and Certification Database to verify if the couple NickName and Password is the same provided through the registration process. It also retrieves the user public key stored in the DB.<br>If this method returns *null* it means the user is not trusted (is not present in the database pertinent table).<br>Note: the password is encrypted both the one received as input from the method and the one stored in DB. The equality check is made between encrypted strings |
| Input parameters | String nickName – the nickname of a business user entitled to register new users or to update users data<br>String password – the password of a business user entitled to register new users or to update users data |
| Output parameters | UserLoginData – a data structure containing the AXUID and the public key of the user requesting the registration/update operation in case of success, a *null* value otherwise |

| **DataManager** | |
|---|---|
| Method | queryExecuter |
| Description | It performs the query on the basis of data elaborated by “acceptRequest”, a method of the RequestManager |
| Input parameters | String queryConstraints – the cquery to be used to retrieve the requested data |
| Output parameters | ResultSet – a set containing requested datain case of success, an empty set otherwise |

Note that two encrypting/decrypting methods (DecryptComm, EncryptComm) have been introduced to enforce the encrypting robustness. In fact we can suppose to use an encrypted protocol (like SSL), but we can enforce encryption robustness (and therefore security) encrypting ourselves data too using a Public/Private key paradigm. You have to remember that the Distributor public key is stored in AXCS Registration and Certification Database.

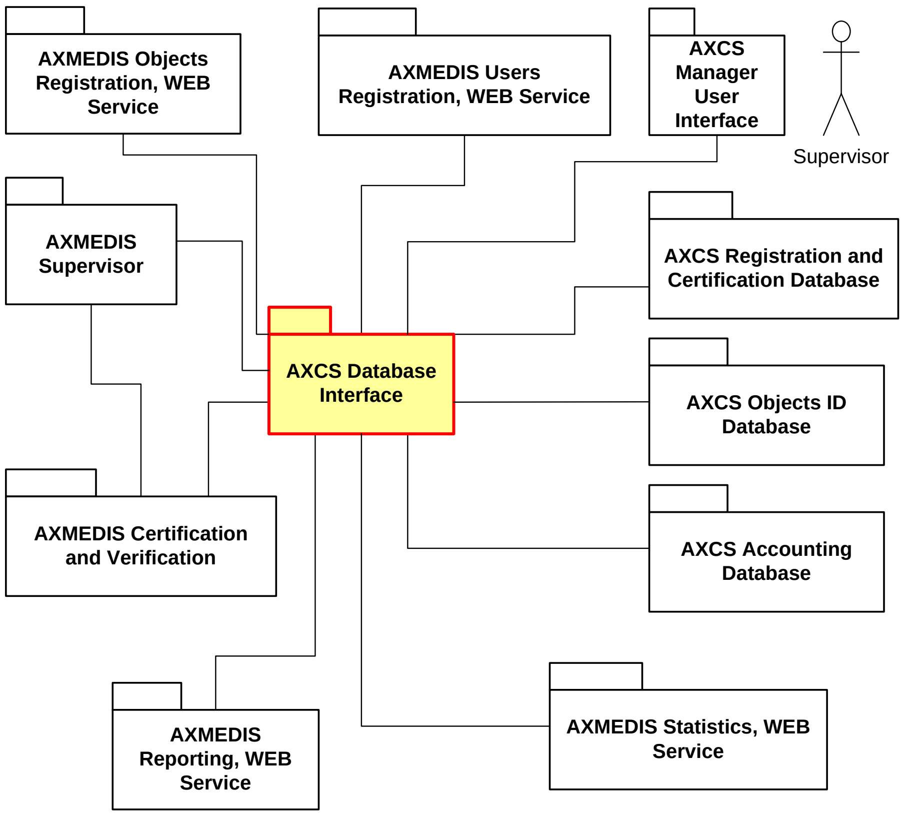
# 10 AXCS Database Interface (DSI)

| Module/Tool Profile | | |
|---|---|---|
| AXCS Database Interface | | |
| Responsible Name | Chellini, Martini | |
| Responsible Partner | DSI | |
| Status (proposed/approved) | Proposed | |
| Implemented/not implemented | Implemented | |
| Status of the implementation | First prototype | |
| Executable or Library/module (Support) | Library (jar file) | |
| Single Thread or Multithread | Multithread | |
| Language of Development | Java | |
| Platforms supported | Anyone on which can be installed a JRE 1.5.0 (5.0) and MySQL | |
| Reference to the AXFW location of the source code demonstrator | https://cvs.axmedis.org/repos/Framework/source/axcs/axcs-framework/src/org/axmedis/axcs/db | |
| Reference to the AXFW location of the demonstrator executable tool for internal download | https://cvs.axmedis.org/repos/Framework/bin/axcs | |
| Reference to the AXFW location of the demonstrator executable tool for public download | | |
| Address for accessing to WebServices if any, add accession information (user aNd Passwd ) if any | | |
| Test cases (present/absent) | absent | |
| Test cases location | | |
| Usage of the AXMEDIS configuration manager (yes/no) | No | |
| Usage of the AXMEDIS Error Manager (yes/no) | No | |
| Major Problems not solved | 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 |
| AXMEDIS prefixes | All AXCS Web services | See section 20 |
| AXMEDIS ID | All AXCS Web services | See section 23 |
| | | |
| | | |
| Protocol Used | Shared with | Protocol name or reference to a |

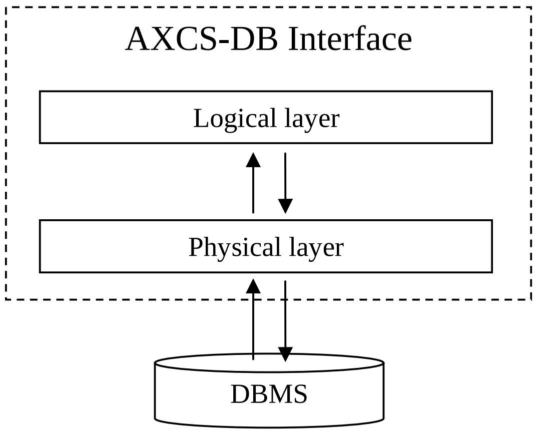
| | | section |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| Used Database name | | |
| AXCSRegCert | | |
| AXCSObjectsID | | |
| AXCSAccounting | | |
| | | |
| | | |
| User Interface | Development model, language, etc. | Library used for the development, platform, etc. |
| | | |
| | | |
| | | |
| | | |
| | | |
| Used Libraries | Name of the library and version | License status: GPL. LGPL. PEK, proprietary, authorized or not |
| UUID generation | Jug (Doomdark) | |
| MySQL JDBC driver | Connector/J 3.1.10 | GPL, but commercial licenses are available |
| | | |
| | | |
| | | |
| | | |

## 10.1 General Description of the Module

The AXCS Database Interface (called AXCS-DB Interface) is the abstraction layer that has the task of giving to the rest of AXCS modules a view of the database that is independent from the database that is really employed.
Thie following figure shows which place this module occupies in the AXCS architecture:

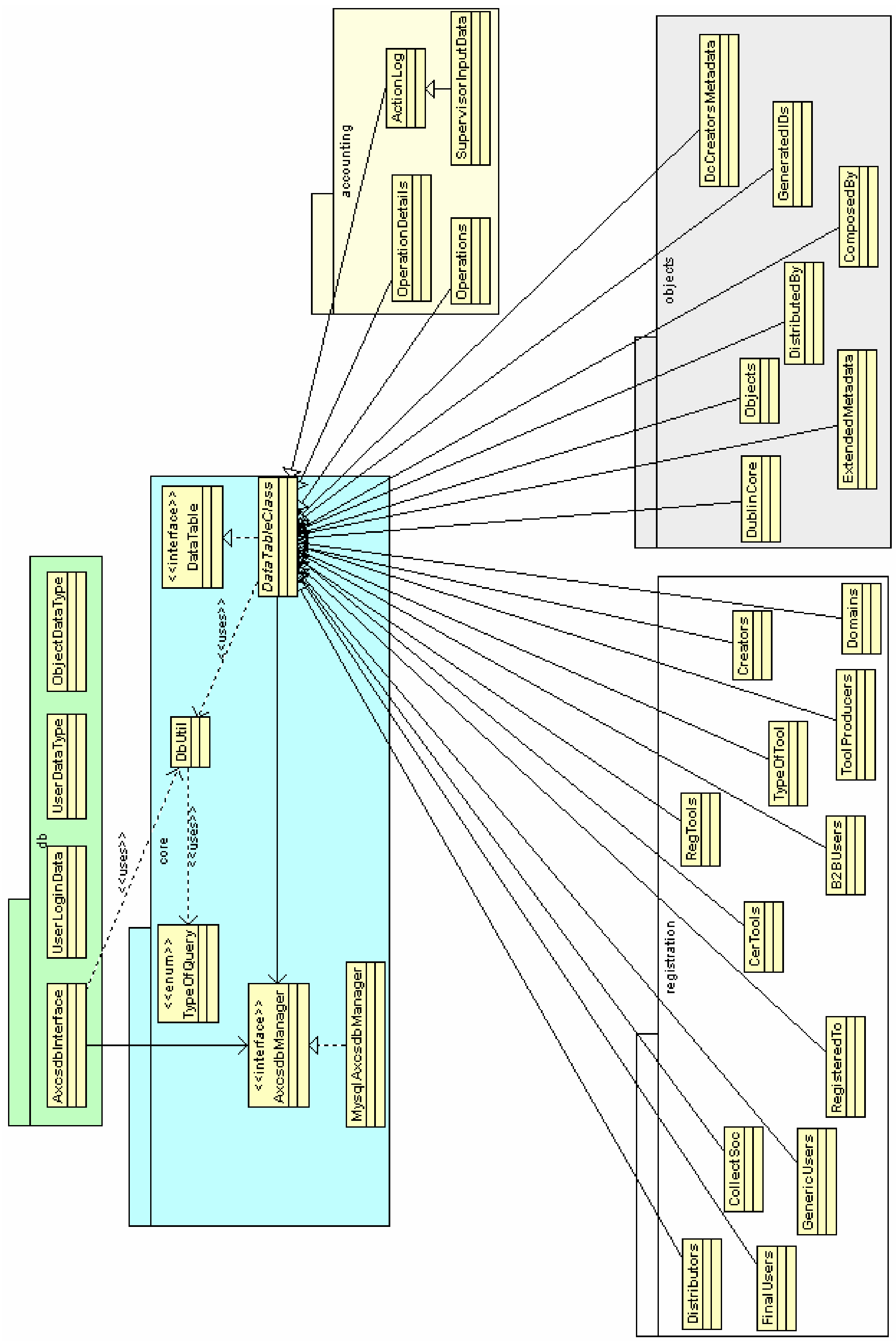

AXMEDIS Objects Registration, WEB Service
AXMEDIS Users Registration, WEB Service
AXCS Manager User Interface
Supervisor
AXMEDIS Supervisor
AXCS Registration and Certification Database
AXCS Database Interface
AXCS Objects ID Database
AXMEDIS Certification and Verification
AXCS Accounting Database
AXMEDIS Reporting, WEB Service
AXMEDIS Statistics, WEB Service

This module can be decomposed in other two layers used to realize the abstraction from the database. The first layer (logical layer) is an object model of the database structure: every table is represented as a class and related managing methods have been implemented. The second layer (physical layer) is realized to decouple database model among the specific DataBase Management System used. In this way, different DBMS such as Mysql, MS-SQL Server, etc. have their own physical layer, and a change in DBMS used does not require a logical layer change.



## 10.2 Module Design in terms of Classes

The logic at the base of this module has been subdivided in some classes in order to separate concerns and to make it easier to reuse some functionalities in other modules. The following figure shows the class diagram of the AXCS database Interface web service (the name of the packages have been truncated to keep the figure clear):



## 10.3 User interface description

No user interface is available.

## 10.4 Technical and Installation information

The axcs-db-interface.jar file has to be placed in a directory included in the classpath environment variable. It also needs the presence of a configuration file named "axcsdb.ini", to be placed in a directory called:

- C:\axcs in Windows environments
- /axcs in *nix environments

See below for more details on the configuration file format.

| | |
|---|---|
| References to other major components needed | Connector/J, Jug |
| Problems not solved | NONE |
| Configuration and execution context | Here are reported some instructions to create the configuration file "axcsdb.ini":<br><br>1. You need also to create a directory called "axcs" (lower case). Place this directory in one of the following location (depending on the OS):<br><br>WINDOWS: C:\axcs<br>*NIX: /axcs<br><br>2. In the axcs directory create a new text file named "axcsdb.ini" (lower case) in which you have to specify location of the AXCS databases and authentication data to access these databases.<br>The file has to be written in the following format.<br><br>AXCSAccounting = jdbc:mysql://<location_of_AXCSAccouning_db>/AXCSAccounting<br>AXCSRegCert = jdbc:mysql://<location_of_AXCSRegCert_db>/AXCSRegCert<br>AXCSObjectsID = jdbc:mysql://<location_of_AXCSObjectsID_db>/AXCSObjectsID<br><br>AXCSDbUser = <user_name><br>AXCSDbPassword = <password><br><br>[dbLog = <log_file_name_with_path>]<br><br>The last line is optional. It lets you specify a logging file in which errors can be written during execution. This file can be placed anywhere you like, but the complete path has to be specified, escaping backslash characters (escape character is a backslash itself). If the last line is not supplied, no log will be created.<br>All data can be specified in any order and white spaces do not matter.<br>Please note that the autentication data have to be the same for all the three databases.<br><br>For instance, if the databases would be hosted on a machine called "MyHost", autentication data would be username: |

| | |
|---|---|
| | "User" and password: "Pw" and logging file would be "C:\axcs\log_file.txt", the file would appear something similar to:<br><br>AXCSAccounting = jdbc:mysql://MyHost/AXCSAccounting<br>AXCSRegCert = jdbc:mysql://MyHost/AXCSRegCert<br>AXCSObjectsID = jdbc:mysql://MyHost/AXCSObjectsID<br><br>AXCSDbUser = User<br>AXCSDbPassword = Pw<br><br>dbLog = C:\\axcs\\log_file.txt |

## 10.5 Draft User Manual

Since the AXCS-DB-INTERFACE is a Java library, please refer to the Javadoc documentation, available at https://cvs.axmedis.org/repos/Framework/doc/code/axcs.

## 10.6 Examples of usage

Suppose we need  to register a new user in the system. Int his case the code has only to create a new instance of the AxcsdbInterface class and use th *setUsrData* method (see section 16 for further details):

```
MysqlAxcsdbManager regDbManager = new MysqlAxcsdbManager(AXCSDbName.AXCSRegCert, 10, 5);
AxcsdbInterface dbInterface = new AxcsdbInterface(regDbManager);

dbInterface.setUsrData(false, new UserDataType(null, "USR_36351030-8315-6070-8280-380689855558",
"546f8c8312323da", " user@fakedomain.it", "italian", "nickname", "password",
"30818902818100f749ec9ad8a22482f9", “2005-01-01”, “2009-31-12”, “U”));
```

Then *setUsrData* will use the generic class method to do some checks and finally store the data:

```
…
GenericUsers user = new GenericUsers(dbManager);
…
user.saveOnDb(isReplace, conn);
…
```

## 10.7 Integration and compilation issues

Since the AXCS-DB-INTERFACE is a Java library, it is enough to place it in a directory included in the classpath environment variable. The only issue is the location of the configuration file “axcsdb.ini”, which is to be placed in C:\axcs in Windows systems or in /axcs in *nix systems.

## 10.8 Configuration Parameters

| Config parameter | Possible values |
|---|---|
| AXCSAccounting | jdbc:mysql://<host_name>/AXCSAccounting<br><br>Es: jdbc:mysql://flauto.dsi.unifi.it/AXCSAccounting |
| AXCSRegCert | jdbc:mysql://<host_name>/AXCSRegCert<br><br>Es: jdbc:mysql://flauto.dsi.unifi.it/AXCSRegCert |

| | |
|---|---|
| AXCSObjectsID | jdbc:mysql://<host_name>/AXCSObjectsID<br><br>Es: jdbc:mysql://flauto.dsi.unifi.it/AXCSObjectsID |
| dbLog | dbLog = <log_file_name_with_path><br><br>Es: dbLog = C:\\axcs\\log_file.txt |

## 10.9 Errors reported and that may occur

Error codes depend on the specific method invoked, please refer to the Javadoc documentation available at https://cvs.axmedis.org/repos/Framework/doc/code/axcs.

## 10.10 Formal description of algorithm AXCS Database Interface

Each layer (either physical or logical) is composed by a proper interface and a group of classes which realize it.
The logical layer interface defines the minimal set of methods needed to manage data in database tables; each class realizes these methods and an other set of more specific ones related to the pertinent tables.
Since this layer is used to access all the different AXCS databases, it is composed of some specific parts. It can be identified the following modules performing their pertinent tasks:

- Registration: It provides a way to access AXCS Registration and Certification Database; it is implemented in the package *org.axmedis.axcs.db.registration*.
- Objects: It provides a way to access AXCS Objects ID Database; it is implemented in the package *org.axmedis.axcs.db.objects*.
- Accounting: It provides a way to access AXCS Accounting Database; it is implemented in the package *org.axmedis.axcs.db.accounting*.

Each module has the same structure as the others. They are composed by the following four methods defined by the interface used in this layer, and each class realizes a 1 to 1 mapping with the related database table.
Concerning to the following methods, please note that the referential integrity cosntraint checks are performed by the DBMS when these constraints are defined in the database. However, it is not possible to define constraints between tables contained in different databases. So, the following methods are defined to perform by default the additional referential integrity constraint checks as needed; this can be avoided if the methods are called setting to *false* the related optional parameter (named *constarintsCheck*). These constraints are defined in the related database sections.

| **Generic class methods (common for each class)** | |
|---|---|
| Method | saveOnDb |
| Description | It saves the object on the DB. If the input parameter is set to false no replace is performed but only an insert that can fail, otherwise it performs an update. |
| Input parameters | boolean replace – it specifies if the operation required is an insert or an update<br>(optional) boolean constarintsCheck – it specifies if the method has to perform the additional referential integrity constraint checks over the related tables and databases. If it is set to *false* these checks are not performed |
| Output parameters | int errorCode – It spcifies the operation result: 0 means ok; see the code documentation for other values. |

| **Generic class methods (common for each class)** | |
|---|---|
| Method | deleteObject |
| Description | This methods eliminates from the current data table of the DB the object having the primary key alredy set |
| Input parameters | (optional) boolean constarintsCheck – it specifies if the method has to perform the additional referential integrity constraint checks over the related tables and databases. If it is set to *false* these checks are not performed |

| | |
|---|---|
| Output parameters | int errorCode – It spcifies the operation result: 0 means ok; see the code documentation for other values |

| **Generic class methods (common for each class)** | |
|---|---|
| Method | loadFromDb |
| Description | This method loads the object with the parameters related to PK that is alredy present in the instance of the object. Returns true if the object has been found and loaded correctly, false otherwise. If a false value is returned there is no guarantee the values in the fields have some sense. |
| Input parameters | None |
| Output parameters | boolean resultStatus – True if the object whose axoid is given as a parameter has been loaded. |

| **Generic class methods (common for each class)** | |
|---|---|
| Method | loadFromDbMulti |
| Description | This method returns the records with the data related to the constraint specified. Returns a ResultSet if the query has been performed correctly, null otherwise. |
| Input parameters | string constraint – It specifies the criteria to select records from table (it is the condition after the SQL 'WHERE' clause). |
| Output parameters | ResultSet result – A ResultSet typed object (eventually empty) if the query could be performed; a *null* value otherwise. |

The physical layer interface defines the minimal set of methods needed to take into account the usage of different DBMS. This interface defines methods used to mange the connection, the transactions and the execution of queries at a lower level.

| **Physichal layer interface methods** | |
|---|---|
| Method | beginTransaction |
| Description | Disables autocommit and allows to perform a set of trasactional queries. |
| Input parameters | None |
| Output parameters | java.sql.Connection connection – Connection to be used for executing sql statements. |

| **Physichal layer interface methods** | |
|---|---|
| Method | commitTransaction |
| Description | Commits a transaction and returns the connection in the pool of available conncetions. |
| Input parameters | java.sql.Connection connection – Connection to be used |
| Output parameters | None |

| **Physichal layer interface methods** | |
|---|---|
| Method | executeSql |
| Description | Executes a generic SQL statement. |
| Input parameters | java.sql.Connection connection – Connection to be used.<br>string sqlStatement – Sql statement to be executed. |
| Output parameters | boolean isResulSet – true if the first result is a ResultSet object; false if it is an update count or there are no results. |

| **Physichal layer interface methods** |
|---|

| | |
|---|---|
| Method | executeSqlUpdate |
| Description | Executes an Update, Insert or Delete SQL Statement. |
| Input parameters | java.sql.Connection connection – Connection to be used.<br>string sqlUpdateInsertDeleteStatement – Sql statement to be executed: must be an INSERT, UPDATE, DELETE statement. |
| Output parameters | int count – Either the row count for INSERT, UPDATE or DELETE statements, or 0 for SQL statements that return nothing. |

| **Physichal layer interface methods** | |
|---|---|
| Method | getConnection |
| Description | This method gets an available and working connection from the internal pool of connections. |
| Input parameters | None |
| Output parameters | java.sql.Connection connection – a connection in the pool of internal connections. |

| **Physichal layer interface methods** | |
|---|---|
| Method | returnConnection |
| Description | Returns a connection in the pool of the available connections. |
| Input parameters | java.sql.Connection connection – The connection to be put again in the pool. |
| Output parameters | None |

| **Physichal layer interface methods** | |
|---|---|
| Method | rollbackTransaction |
| Description | Undoes all changes made in the current transaction and releases any database locks currently held by the connection passed as parameter. |
| Input parameters | java.sql.Connection connection – Connection to be used. |
| Output parameters | None |

| **Physichal layer interface methods** | |
|---|---|
| Method | rollbackTransaction |
| Description | Undoes all changes made after the given Savepoint object was set. |
| Input parameters | java.sql.Connection connection – Connection to be used.<br>java.sql.Savepoint savepoint – Savepoint to be used for rolling back. |
| Output parameters | None |

| **Physichal layer interface methods** | |
|---|---|
| Method | setTransactionSavePoint |
| Description | Sets a savepoint in the transaction for a future rollback. |
| Input parameters | java.sql.Connection connection – Connection to be used. |
| Output parameters | java.sql.Savepoint savepoint – a Savepoint object to be used for rollback. |

| **Physichal layer interface methods** | |
|---|---|
| Method | sqlQueryClose |
| Description | Closes and invalid the results of a previously issued Select. |
| Input | java.sql.Statement statement – Statement to be used for getting results. |

| | |
|---|---|
| parameters | |
| Output parameters | None |

| **Physichal layer interface methods** | |
|---|---|
| Method | sqlQueryIssue |
| Description | Issues a Select sql statement and gets results |
| Input parameters | java.sql.Connection connection – Connection to be used.<br>string sqlStatement – Sql statement to be executed. |
| Output parameters | java.sql.ResulSet result – a ResulSet object that is managed to get query results. |

Physical connections to the DBMS used are managed by means of a connection pool manager. It creates a predefined but configurable set of connections (the pool) when it is used for the first time. These connections can be used at request and when they are no more used they are released but not destroyed: they return into the pool to be available again. In this way the overhead due to the creation of connections can be avoided at the time a connection is needed. If the pool of available connections becomes empty it gets incremented by a fixed (but configurable) amount of new connections. The management of the pool is completely transparent and it does not require any operation from the requestors.

Endling there is also an higher layer used to make available some particular methods intented to give a different representation of the database to make easier the management. At the moment it includes some methods representing the complex user hierarchy as a single object. This layer is called AXCSdbInterface and it is described in section 16.

A detailed description of each method is provided in the Javadoc documentation available at https://cvs.axmedis.org/repos/Framework/doc/code/axcs.

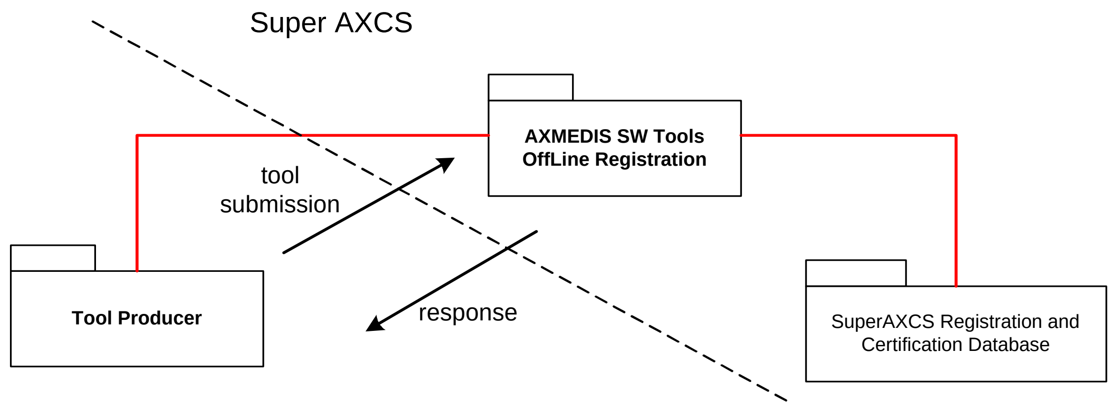
# 11 AXCS Software Tool Off Line Registration (DSI)

| Module/Tool Profile | | |
|---|---|---|
| AXCS Software Tool Off Line Registration | | |
| Responsible Name | Chellini, Martini | |
| Responsible Partner | DSI | |
| Status (proposed/approved) | proposed | |
| Implemented/not implemented | Not yet implemented | |
| Status of the implementation | | |
| Executable or Library/module (Support) | Web Application | |
| Single Thread or Multithread | Multithread | |
| Language of Development | | |
| Platforms supported | | |
| Reference to the AXFW location of the source code demonstrator | | |
| Reference to the AXFW location of the demonstrator executable tool for internal download | | |
| Reference to the AXFW location of the demonstrator executable tool for public download | | |
| Address for accessing to WebServices if any, add accession information (user aNd Passwd ) if any | | |
| Test cases (present/absent) | absent | |
| Test cases location | | |
| Usage of the AXMEDIS configuration manager (yes/no) | | |
| Usage of the AXMEDIS Error Manager (yes/no) | | |
| Major Problems not solved | | |
| Major pending requirements | | |
| | | |
| Interfaces API with other tools, named as | Name of the communicating tools References to other major components needed | Communication model and format (protected or not, etc.) |
| Tool Producer Area | | Protected |
| SuperAXCS:Registration and Certifiaction Database | | |
| | | |
| | | |
| 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. |
| Web Interface | | |
| | | |
| | | |
| | | |
| | | |
| Used Libraries | Name of the library and version | License status: GPL. LGPL. PEK, proprietary, authorized or not |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

## 11.1 General Description of the Module

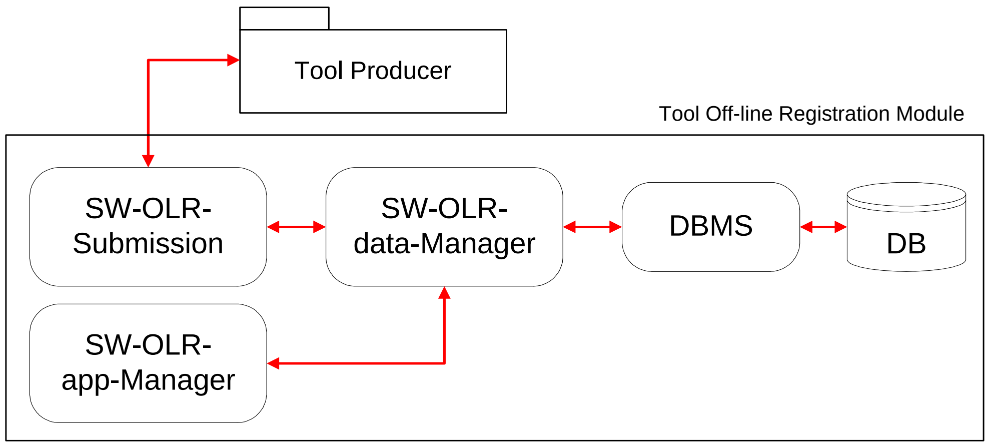
The Software Tool Off-line Registration is a web application oriented to AXMEDIS Tool Producers. Its role is to allow tool producers the submission of their candidate software to become certified AXMEDIS Tools. Tool checking activity is not part of this module: it is made in a second time, after receiving the aspirant tool. It has to remember that a candidate tool has to be checked and tested to verify if it accomplishes AXMEDIS guidelines. The Software Tool Off-line Registration module has to supply also an interface intended to collect data about tool and to insert it in AXCS object database.


Super AXCS
AXMEDIS SW Tools OffLine Registration
tool submission
response
Tool Producer
SuperAXCS Registration and Certification Database

**Tool off-line registration application scenario**

This module has to provide the following functions:

- Receive the software tool candidate from tool producers
- Provide a web interface to SuperAXCS Manager in order to manage the software reception, and to complete data about the received software
- Access SuperAXCS database to manage (register and retrieve data) about candidate tools.



To perform its own task it can be found the following logical decomposition:

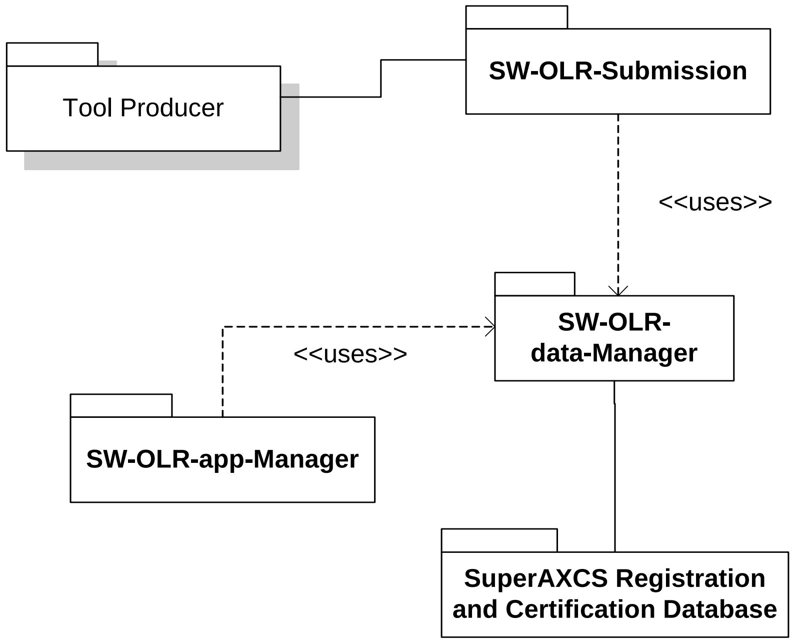
**Sw-OLR-submission**: it realizes the web interface needed to accept and receive the software tool candidate. The received software is temporarily stored on the web server to be retrieved for the necessarily tests. It also place a web interface used by software tool producer to give some more information about the provided candidate.

**Sw-OLR-app-manager**: It is only a web interface needed by the Super AXCS manager to complete data about the received software, accept or discharge it and manage the whole Software Tool Off-line Registration process.

**Sw-OLR-data-manager**: It is a module intended to provide the above modules some method to access database (retrieve and store data).

## 11.2 Module Design in terms of Classes

The logic at the base of the Tool Off-line Registration mechanism has been subdivided in some classes in order to separate concerns and to make it easier to reuse some functionalities in other modules. The following figure shows the class diagram of the AXCS Tool Off line Registration :



## 11.3 User interface description

The user interface of this module will consist of a web page at which users have to provide some identification data (such as Factory name, address telephone number, etc.) and which will provide a mechanism to send the candidate software to become an AXMEDIS compliant tool.

## 11.4 Technical and Installation information

Not yet available

## 11.5 Draft User Manual

Not yet available

## 11.6 Examples of usage

Not yet available

## 11.7 Integration and compilation issues

None

## 11.8 Configuration Parameters

| Config parameter | Possible values |
|---|---|
| None | None |

## 11.9 Errors reported and that may occur

| Error code | Description and rationales |
|---|---|
| None | None |

## 11.10 Formal description of algorithm Tool Off-line Registration

| Method name and parameters | Output | Description |
|---|---|---|
| **scope: public** | | |
| SW-OLR-Submission::ReceiveTool (string NickName, string Password, toolinfoType ToolInfoData) | String ResultStatus | It is a public method. It collects requestor credentials needful to access the system and uses the VerifyLogin() (a SW-OLR-DataManager method) to verify requestor credentials. It also collects tool relate data (in a field called ToolInfoData typed toolinfoType), according to TypeofTool table in Registration and certification database. It also receives the candidate software tool and store it in a folder on the web server.<br>This method uses private methods described below to perform its own tasks. The ResultStatus output parameter is set to 0 if the task is successful and is set to 1 otherwise. |
| SW-OLR-app-Manager::Management (string NickName, string Password) | String ResultStatus | It is a public method. In implements the application management. It collects module manager credentials needful to access the Application Manager and uses the VerifyLogin() (a SW-OLR-DataManager method) to verify user credentials.<br>This module provides also a web interface (a panel) allowing SuperAXCS manager to perform the following tasks:<br>a. candidate tool related data management (edit and complete)<br>b. acception or rejection of candidate software tool<br>c. sending a message to tool producer about the outcome of candidate tool analysis (using the SW-OLR-app-Manager::Response method)<br>d. type of tool fingerprint calculation (using the SW-OLR-app-Manager::FingerprintCalc method) |
| **scope: private** | | |
| SW-OLR-app-Manager::Response (String ToolProducer-email, string message) | String ResultStatus | It sends a message to software tool candidate producer related to the result of software candidate analysis. The ResultStatus output parameter is set to 0 if the task is successful and is set to 1 otherwise. |
| SW-OLR-app-Manager::FingerprintCalc | String ResultStatus | It calculates the software tool |

| | | |
|---|---|---|
| (String ToolPath) | String ToolFinerprint | fingerprint and return it in the ToolFingerprint field to be used by the calling method. The ResultStatus output parameter is set to 0 if the task is successful and is set to 1 otherwise. |
| CollectorTransferManager::EncryptComm (string ClearData, string asimKey) | String EncodedData | Encrypt ClearData with provided asymmetric key (asymKey) |
| CollectorTransferManager::DecryptComm (string String EncodedData, string asimKey) | String ClearData | Decrypt EncodedData with provided asymmetric key (asymKey) |
| SW-OLR-DataManager::Data-Retriever(string Query)<br>(the needful input data can be stored in the class attributes or sent as method parameters) | String ResultQuery | This method retrieve data from database according to its input values. The ResultQuery field returns query response data |
| SW-OLR-DataManager::Data-Inserter()<br>(the needful input data can be stored in the class attributes or sent as method parameters) | String ResultQuery | This method inserts data into database according to its input values. The ResultQuery field returns a value to indicate if operation was successful |
| SW-OLR-DataManager::Data-Modifier()<br>(the needful input data can be stored in the class attributes or sent as method parameters) | String ResultQuery | This method modifies data into database according to its input values. The ResultQuery field returns a value to indicate if operation was successful |
| SW-OLR-DataManager::VerifyLogin (string NickName, string Password) | boolean IsTrusted | It accesses Registration and Certification Database to verify if the couple NickName and Password is the same provided through the registration process. |

## 12 AXCS Manager User Interface (DSI)

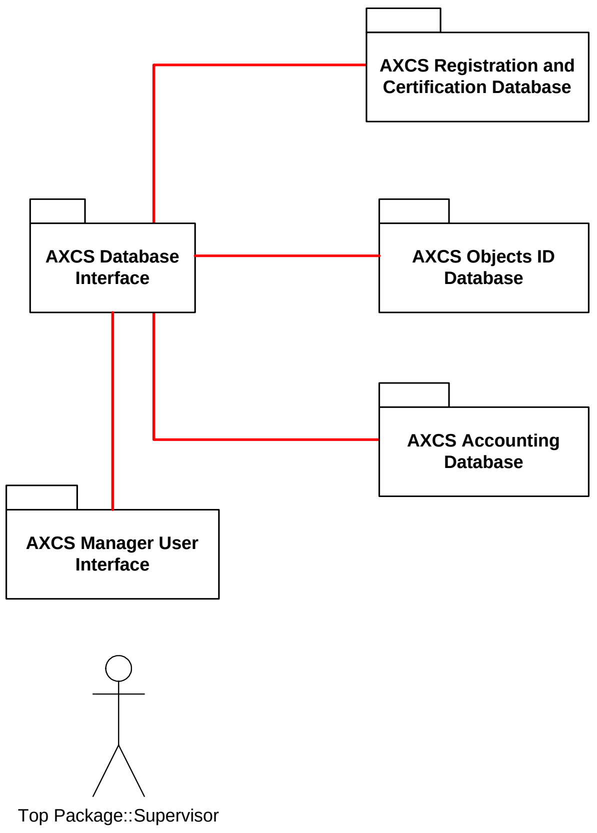
| Module/Tool Profile | | |
|---|---|---|
| AXCS Manager User Interface | | |
| Responsible Name | Chellini, Martini | |
| Responsible Partner | DSI | |
| Status (proposed/approved) | proposed | |
| Implemented/not implemented | Not implemented | |
| Status of the implementation | | |
| Executable or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

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

## 12.1 General Description of the Module

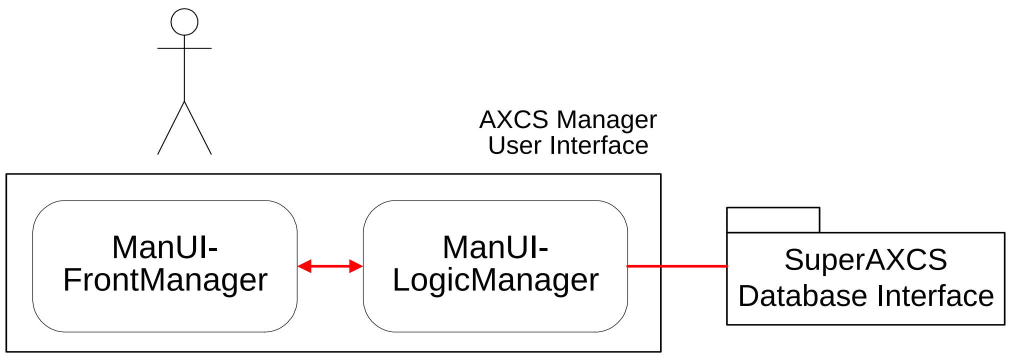
This module has been provided to perform some administrative task needed by AXCS manager. Using this module an AXCS manager can execute many jobs related to database mantainance and check.

## 12.2 Module Design in terms of Classes

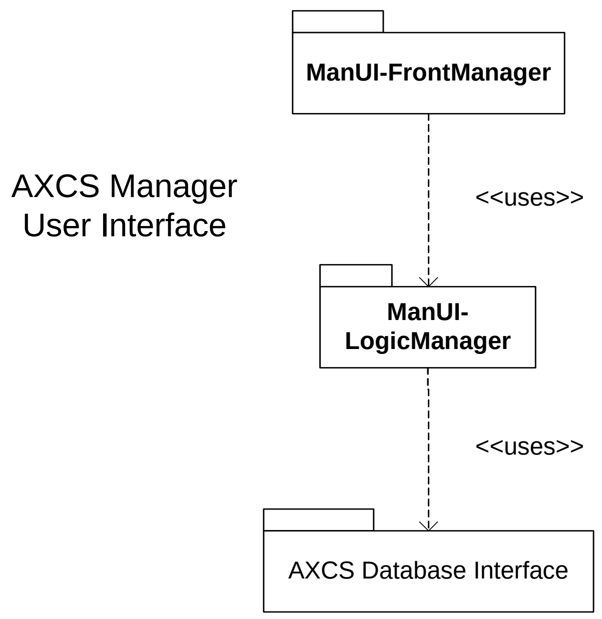


The AXCS Manager User Interface is a module implementing a web interface intented to provide AXCS manager a simple but powerful interface to perform his administrative tasks. This module has to provide the following functions:

- databases maintenance functions such as insertion, editing, deletion and selection of database fields
- tool, objects, users blocking/unblocking
- log viewing in order to detect forcing system attempts



In order to perform its own task it can be found the following logical decomposition:
**ManUI-FrontManager**: it is the web pages manager. It generates all the web pages needful to data collection and data presentation.
**ManUI-LogicManager**: it realizes the whole application logic and uses AXCS Database Interface (and the related AXDB-API) to access database.



Web pages should be organizes so as giving the possibility to user to perform the functions above. Data access views should be organized also by function and by database, in order to aggregate all tables belonging to the same database and to evidence the possible operation that could be performed on.

## 12.3 User interface description

Not yet available

## 12.4 Technical and Installation information

Not yet available

## 12.5 Draft User Manual

Not yet available

## 12.6 Examples of usage

Not yet available

## 12.7 Integration and compilation issues

None

## 12.8 Configuration Parameters

| Config parameter | Possible values |
|---|---|
| None | None |

## 12.9 Errors reported and that may occur

| Error code | Description and rationales |
|---|---|

| | |
|---|---|
| None | None |

## 12.10 Formal description of algorithm
Not yet available

# 13 AXCSs/PMSs: Data Request and Diffusion (DSI)

| Module/Tool Profile | | |
|---|---|---|
| AXCSs/PMSs: Data Request and Diffusion | | |
| Responsible Name | Chellini, Martini | |
| Responsible Partner | DSI | |
| Status (proposed/approved) | Proposed | |
| Implemented/not implemented | Not implemented | |
| Status of the implementation | | |
| Executable or Library/module (Support) | | |
| Single Thread or Multithread | | |
| Language of Development | | |
| Platforms supported | | |
| Reference to the AXFW location of the source code demonstrator | | |
| Reference to the AXFW location of the demonstrator executable tool for internal download | | |
| Reference to the AXFW location of the demonstrator executable tool for public download | | |
| Address for accessing to WebServices if any, add accession information (user aNd Passwd ) if any | | |
| Test cases (present/absent) | | |
| Test cases location | | |
| Usage of the AXMEDIS configuration manager (yes/no) | | |
| Usage of the AXMEDIS Error Manager (yes/no) | | |
| Major Problems not solved | None | |
| Major pending requirements | --<br>-- | |
| | | |
| Interfaces API with other tools, named as | Name of the communicating tools References to other major components needed | Communication model and format (protected or not, etc.) |
| | | |
| | | |
| | | |
| | | |
| Formats Used | Shared with | format name or reference to a section |
| | | |
| | | |
| | | |
| | | |

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

## 13.1 General Description of the Module

As stated above in the present document, AXCS and PMS are not unique entities in the system but there can be found many AXCSs and many PMSs organized as a logical network. Each of them is put in charge of manage at best their pertinent data. Moreover it is needed the system to be fault tolerant, and any data managed by any AXCS or PMS has to be available even if the owner entity is off-line or not reachable for any reason. Thus a mechanism for data migration among AXCSs and PMSs have to be planned and implemented. It has to be noticed that data has to be transferred fron an AXCS to another and from a PMS to another and not from AXCS to PMS. This transfer mechanism has to be able to manage all the problem related to data replication including data sinchronzation and data gathering (collection).
The couple AXCS Synchronizer and AXCS Collector has been defined in order to supply an efficient way to transfer data among AXCSs and PMSs. Moreover a specialization of AXCS, called SuperAXCS has been theorized, in order to excecute coordination and supervising tasks over AXCSs activity.
Several architectures can be thinked:

1. Hierarchical Network. In this architecture SuperAXCS is located at an higher level than AXCSs. Here has to be present a client part and a server part. SuperAXCS Collector could be the client part that periodically requests to AXCSs Synchronizer all the data needed and store it in its database. This is the simplest way but probably it isn't the most efficient. A simple variant could be the "on-line" synchronization where AXCSs Synchronizer tries to transfer data at the same time they perform their tasks. Here are reported two simple cases to better explain these concepts.

| On-line synchronization | |
|---|---|
| 1 | End user uses an AXMEDIS tool to operate on AXMEDIS Protected Objects that are on different |

| | |
|---|---|
| | distribution channels |
| 2 | Protection Manager Support allows only authorized operations on the objects |
| 3 | Objects are accessed on different channels and each AXCS stores its Action-Logs |
| 4 | Via AXCS synchronizer general information on Objects or information that allow Super AXCS to recover Action-Logs from the different AXCSs are transferred to Super AXCS Collector |
| 4a | If connection between AXCS and Super AXCS is not active, AXCS synchronizer stores the information to be transferred in a queue called AXCS Synchronizer Queue. Information stored in that queue are transferred to SuperAXCS Collector when the connection returns active. |
| 5 | Super AXCS collects and sores the received information |

| Off-line synchronization | |
|---|---|
| 1 | Super AXCS Collector retrieve the list of all the AXCS registered in the system (performing a query to Active AXCS List Database) |
| 2 | Super AXCS Collector slides every entry in that list and, for each AXCS, sends a Queue Pull Request, i.e. a request for data present in the AXCS Synchronizer Queue. |
| 3 | The contacted AXCS Synchronizer respond providing the requested data (if present) and empty the AXCS Synchronizer Queue. |

2. Peer Network. In this architecture SuperAXCS is only a coordinator element. It stores only the list of registered AXCSs and other few information. Every AXCS manage its own data. When information stored in a specific AXCS is needed, SuperAXCS sent it to some AXCS that probably retains the needed data (chosen with some criteria). Every AXCS is a peer and queries are propagated over the network in order to retrieve the needed information. Every AXCS stores its own data and some other data coming from the adjacent AXCSs to be able to propagate the query in a tricky way. In this architecture the couple Synchronizer/Collector is not necessary. It can be supposed also a mix of Peer and Hierarchical network, a kind of a network structured on two or more levels, where the highest level peers have major functions respect to lower level peers.

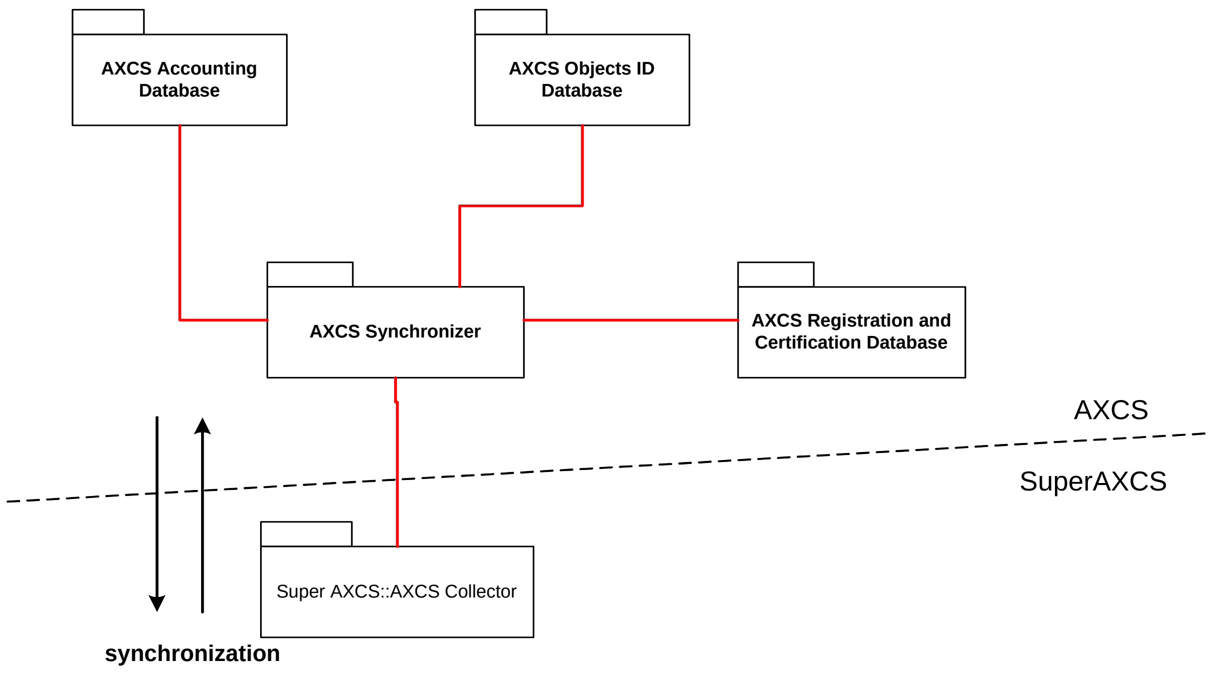
3. Blend Network. In this architecture SuperAXCS is more than a coordinator element. It stores only data needful to retrieve information about which is the correct AXCS to query (the AXCS that really stores the needed data). In this architecture data transferred by Synchronizer to Collector is not so excessive. The synchronization methods could be the same as the Hierarchical Network.

This different network architectures could be implemented and tested in a second time, in order to test them and be able to make the better choice.

## 13.2 Module Design in terms of Classes

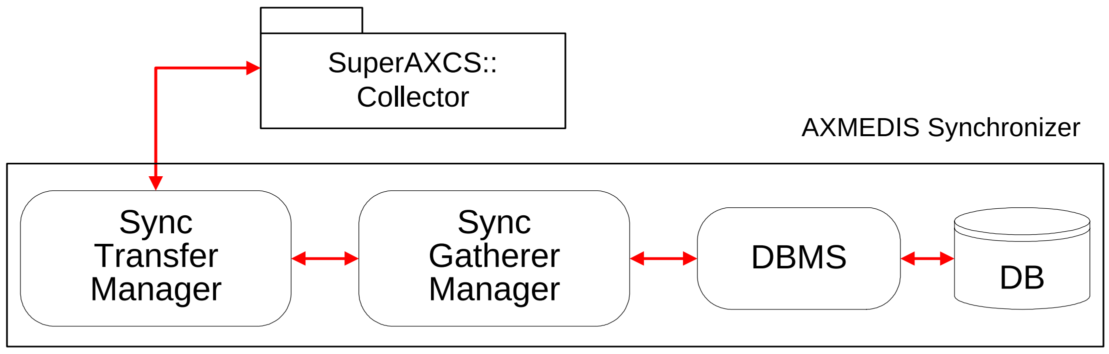
### 13.2.1 AXCS Synchronizer

AXCS Synchronizer is an AXCS module used to synchronize AXCSs databases. Every AXCS database entry has to be retrievable by other AXCSs in order to be able to provide to requestors all the information contained in each AXCS database.



Synchronizer should be composed by some modules performing the following tasks:

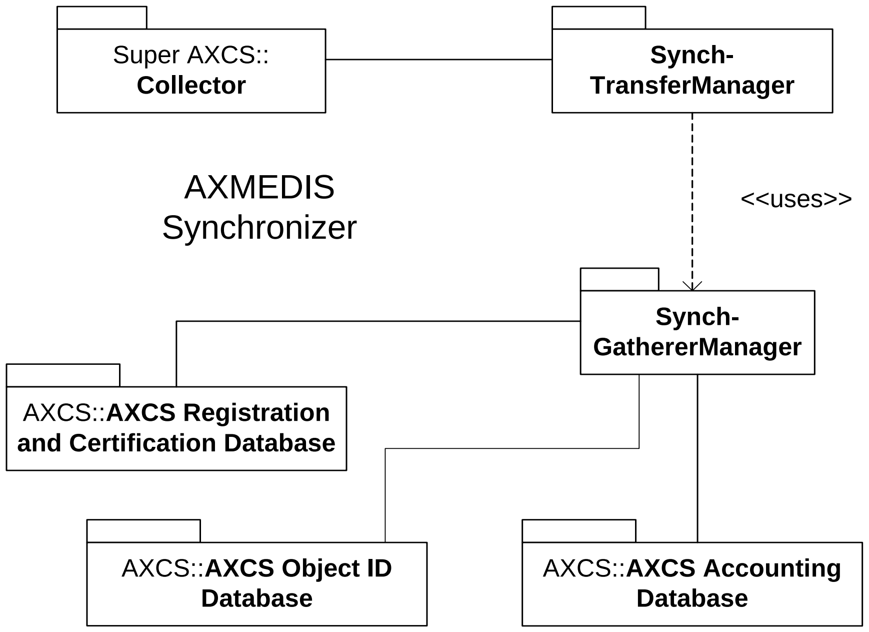
- gather the AXCS database data that has to be transferred to SuperAXCS Collector
- transfer it to SuperAXCS Collector



To perform these tasks can be considered two different modules:

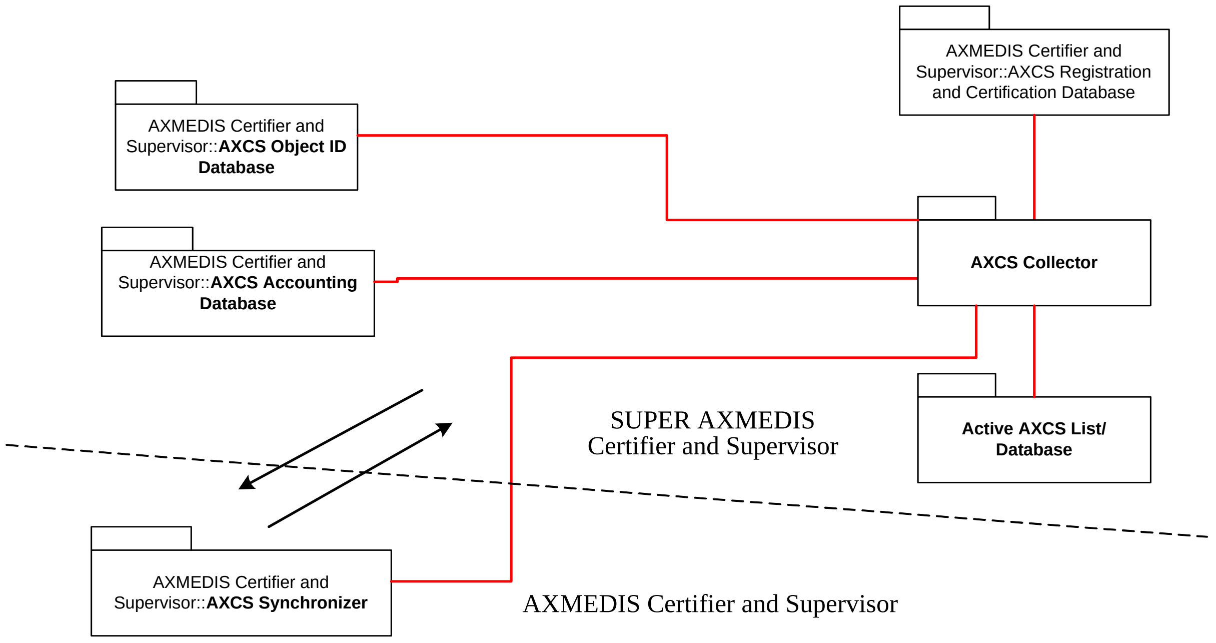
**SyncGathererManager**: it gathers all data that has to be transfer to Collector and elaborates it before is treated by SyncTransferManager. The database management is performed using the related API (AXDB-API). To support synchronization a new structure has been introduced: the AXCS Synchronizer Queue. When the connection betrwwen Sychronizer (in AXCS) and Collector (in SuperAXCS) is not active, AXCS synchronizer stores the information to be transferred in the AXCS Synchronizer Queue. Information stored in that queue are transferred to SuperAXCS Collector when the connection returns active.

**SyncTransferManager**: it manages the connection and deals with SuperAXCS Collector in order to transfer the pertinent data. The communication channel connecting SuperAXCS Collector and AXCS Synchronizer is protected using a secure protocol (for instance SSL). The communication can happen with Synchronizer as client or server (and the Collector as a respective counterpart). To perform both paradigms have been considered a public method both in Synchronizer and in Collector.


Super AXCS:: Collector
Synch-TransferManager
AXMEDIS Synchronizer
<<uses>>
Synch-GathererManager
AXCS::AXCS Registration and Certification Database
AXCS::AXCS Object ID Database
AXCS::AXCS Accounting Database

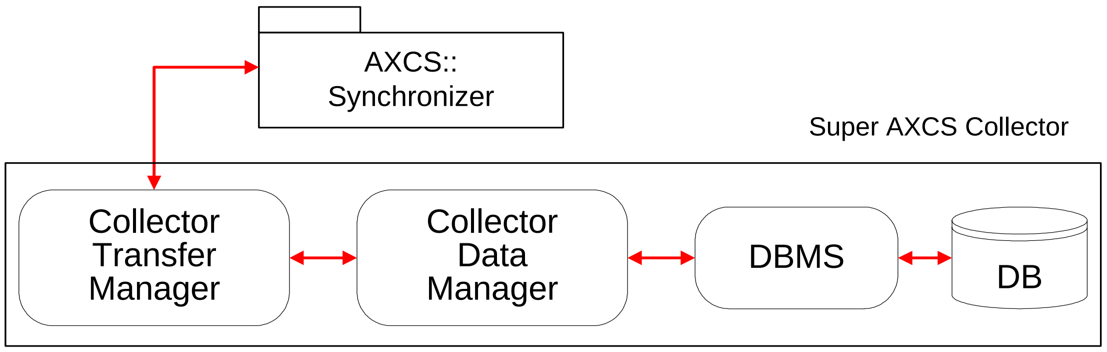
### 13.2.2 AXCS Collector

AXCS Collector is th counterpart of AXCS synchronizer. The interaction and collaboration between these two components is very important. To better unserstand the AXCS Collector role and logic, please take a look to the AXCS Synchronizer section.


AXMEDIS Certifier and Supervisor::AXCS Registration and Certification Database
AXMEDIS Certifier and Supervisor::AXCS Object ID Database
AXCS Collector
AXMEDIS Certifier and Supervisor::AXCS Accounting Database
SUPER AXMEDIS Certifier and Supervisor
Active AXCS List/ Database
AXMEDIS Certifier and Supervisor::AXCS Synchronizer
AXMEDIS Certifier and Supervisor

The SuperAXCS Collector has to perform the following tasks:

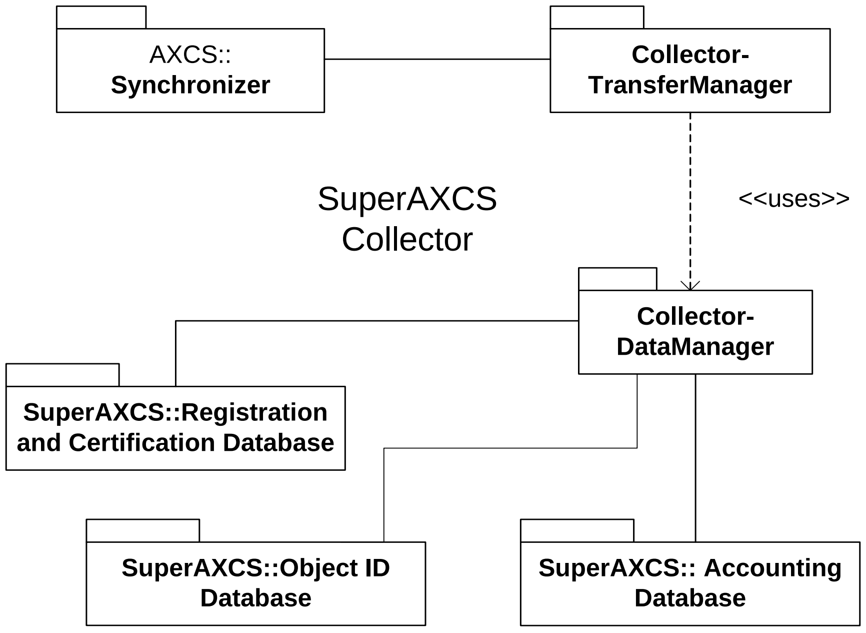
- receive data from AXCSs Synchronizer (requesting it or registering their posts)
- insert the received (or requested data) into database



In order to perform its own task AXCS Collector can be decomposed in the following modules:

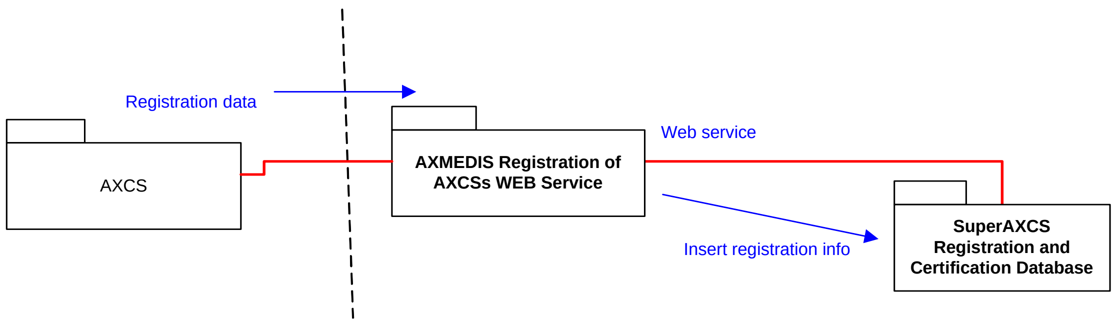
**CollectorTransferManager**: it manages the connection and deals with AXCS Synchronizer in order to receive the pertinent data. The communication channel connecting SuperAXCS Collector and AXCS Synchronizer is protected using a secure protocol (for instance SSL). The communication can happen with Synchronizer as client or server (and the Collector as a respective counterpart). To perform both paradigms have been considered a public method both in Synchronizer and in Collector.

**CollectorDataManager**: it collects all data received from AXCS Synchronizer and inserts it in database. It could be thought as a composition of AXCS components that perform single tasks in order to maximize the ideas and code reuse. For instance we can think to reuse the AXMEDIS Registration WEB Service to collect data about AXMEDIS Users, AXCS OID Generator to collect data about AXMEDIS Objects, and so on.



### 13.2.3 Axmedis Registration of AXCSs

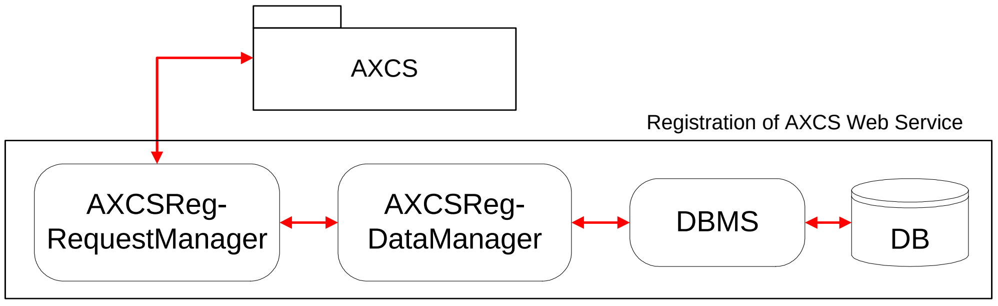
All AXMEDIS AXCS must be registered by other AXCSs or SuperAXCS. The AXMEDIS Registration Web Services is the AXCS module that receives data about AXCS and store it in the AXCS Registration and Certification Database. Once the AXCS is registered in the system and the related data is stored in the pertinent database, other modules can access this database to retrieve some information needful to perform their work.


Registration data
AXCS
AXMEDIS Registration of AXCSs WEB Service
Web service
Insert registration info
SuperAXCS Registration and Certification Database

**AXCSs registration application scenario**

The AXMEDIS Registration of AXCS Web Service is a web application running on a web server, implemented as a set of web scripts. We can identify the following logical decomposition:

- **AXCSReg-RequestManager**: this component receives registration requests from AXCS, and prepares them to be processed by the AXCSReg-DataManager. The communication channel connecting AXCS and AXCSReg-RequestManager is protected using a secure protocol (for instance SSL). It implements the interface with requestors and manages the whole application.
- **AXCSReg-DataManager**: this component receives data from AXCSReg-RequestManager and inserts it in the SuperAXCS Registration and Certification Database. If necessary elaborates and fits data before insert it in the database. The database management is performed using the related API (AXDB-API).



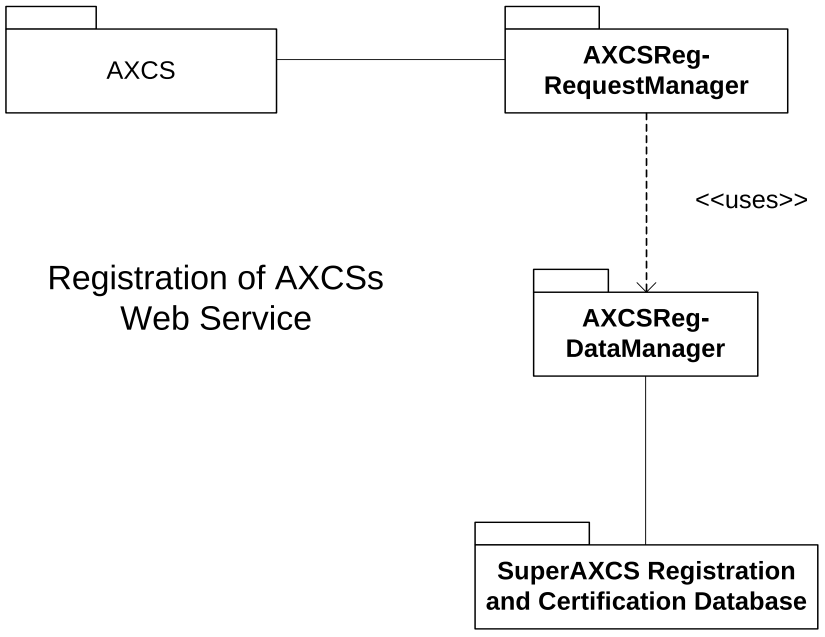
AXMEDIS Registration of AXCSs Web Service architecture.

The **AXCSReg-RequestManager** should be composed by some modules implementing the following functions:

- Accepting authentication data, needful to verify reuqestor credentials and make it access the system
- Accepting AXCS registration data and preparing it to be transferred to AXCSReg-DataManager
- Reply to requestor with the most appropriate message (on the basis of AXCSReg-DataManager responses)

The **AXCSReg-DataManager** should be composed by some modules implementing the following functions:

- Receiving verification requests and data from the AXCSReg-RequestManager
- Accessing the database (using the AXDB-API) to verify requestors credential information
- Accessing the database to store received AXCSs registration data
- Reply to AXCSReg-RequestManager according to the performed actions


AXCS
AXCSReg-RequestManager
<<uses>>
Registration of AXCSs Web Service
AXCSReg-DataManager
SuperAXCS Registration and Certification Database

## 13.3 User interface description

Not yet available

## 13.4 Technical and Installation information

Not yet available

## 13.5 Draft User Manual

Not yet available

## 13.6 Examples of usage

Not yet available

## 13.7 Integration and compilation issues

None

## 13.8 Configuration Parameters

| Config parameter | Possible values |
|---|---|
| None | None |

## 13.9 Errors reported and that may occur

| Error code | Description and rationales |
|---|---|
| None | None |

## 13.10 Formal description of algorithm

The following table describes methods thought to be used in AXCS Synchronizer.

| **Method name and parameters** | **Output** | **Description** |
|---|---|---|
| **scope: public** | | |
| SyncTransferManager::Synchronize(string NickName, string Password, string TimeStampQuery) | String ResultStatus syncDataType Response | It is the only public method of this module. It collects Super AXCS credentials needful to access the system and uses the VerifyLogin() (a GathererManager method) to verify requestor credentials. This method uses private methods described below to perform its own tasks and to answer to requestor. The TimeStampQuery contains the time (in timestamp form) after which the collector needs to be updated; this field is optional and depends on the type of architecture/synchronization mode will be chosen. The ResultStatus output parameter is set to 0 if the task is successful and is set to 1 otherwise. This method returns also the data to be transferred to the requestor using the Response field (typed syncDataType). |
| **scope: private** | | |
| SyncTransferManager::EncryptComm (string ClearData, string asimKey) | String EncodedData | Encrypt ClearData with provided asymmetric key (asymKey) |
| SyncTransferManager::DecryptComm (string String EncodedData, string asimKey) | String ClearData | Decrypt EncodedData with provided asymmetric key (asymKey) |
| SyncGathererManager::DataGatherer() (the needful input data can be stored in the class attributes or received as method parameters) | String QueryResult | It gathers the needed data from database. In order to support its own task it uses the AXCS Synchronizer Queue (see above) .Query results are stored in the QueryResult field |
| SyncGathererManager::VerifyLogin (string NickName, string Password) | boolean IsTrusted | It accesses AXCS Registration and Certification Database to verify if the couple NickName and Password is the same provided through the registration process Note: the password is encrypted both the one received as input from the method and the one stored in DB. The equality check is made between encrypted strings. |

Note that it has been introduced two encrypting/decrypting methods (DecryptComm, EncryptComm) to enforce the encrypting robustness. In fact we can suppose to use an encrypted protocol (like SSL), but we can enforce encryption robustness (and therefore security) encrypting our self data too using a Public/Private key paradigm. You have to remember that public keys are stored in AXCS Registration and Certification Database.

The following table describes methods thought to be used in Super AXCS Collector.

| **Method name and parameters** | **Output** | **Description** |
|---|---|---|

| | | |
|---|---|---|
| **scope: public** | | |
| CollectorTransferManager::Synchronize(string NickName, string Password, syncDataType TransferData) | String ResultStatus | It is the only public method of this module. It collects AXCSs credentials needful to access the system and uses the VerifyLogin() (a CollectorDataManager method) to verify requestor credentials.<br>This method uses private methods described below to perform its own tasks and to answer to requestor.<br>The ResultStatus output parameter is set to 0 if the task is successful and is set to 1 otherwise. This method receives (as input parameter) also data to be transferred from the requestor in the TransferData field (typed syncDataType). |
| **scope: private** | | |
| CollectorTransferManager::EncryptComm (string ClearData, string asimKey) | String EncodedData | Encrypt ClearData with provided asymmetric key (asymKey) |
| CollectorTransferManager::DecryptComm (string String EncodedData, string asimKey) | String ClearData | Decrypt EncodedData with provided asymmetric key (asymKey) |
| CollectorDataManager::DataCollect(syncDataType TransferData)<br>(the needful input data can otherwise be stored in the class attributes or received as method parameters) | String ResultStauts | It collects the received data (using the TranferData field) and inserts it in database.<br>The ResultStatus output parameter is set to 0 if the task is successful and is set to 1 otherwise. |
| CollectorDataManager::VerifyLogin (string NickName, string Password) | boolean IsTrusted | It accesses SuperAXCS Registration and Certification Database to verify if the couple NickName and Password is the same provided through the registration process<br>Note: the password is encrypted both the one received as input from the method and the one stored in DB. The equality check is made between encrypted strings. |

The following table describes methods thought to be used in AXMEDIS Registration Web Service.

| Method name and parameters | Output | Description |
|---|---|---|
| **scope: public** | | |
| AXCSReg-RequestManager::Registration(string NickName, string Password, AXCSDatatype regdata) | string result<br>string definitive-AXCSID | It is the only public method of this web service. It collects requestor credentials needful to access the system and uses the VerifyLogin() (an AXCSReg-DataManager method) to verify requestor credentials. It collects also registration data (regdata) provided by the requestor and uses the other methods (described below) to insert them in database and to provide the result to the requestor. The result output parameter is set to 0 if the registration is successful otherwise is set to 1. This method returns also the definitive AXCSID in definitive-AXCSID |
| **scope: private** | | |
| AXCSReg-RequestManager::DataPrepare() | All user registration fields. See below for details. | It prepares the AXCS registration data received by the web service to be stored in DB. Is also uses |

| | | |
|---|---|---|
| | | IDGenerator() to get the definitive AXCSID to be inserted in the database and to be sent to requestor |
| AXCSReg-AXCSReg-RequestManager::IDGenerator(string TempUID) | String DefinitiveAXCSID | It provides the definitive AXCSID. |
| AXCSReg-RequestManager::EncryptComm(string ClearData, string asimKey) | String EncodedData | Encrypt ClearData with provided asymmetric key (asymKey) |
| AXCSReg-RequestManager::DecryptComm(string String EncodedData, string asimKey) | String ClearData | Decrypt EncodedData with provided asymmetric key (asymKey) |
| AXCSReg-DataManager::VerifyLogin(string NickName, string Password) | boolean IsTrusted<br>string PublicKey | It accesses Registration and Certification Database to verify if the couple NickName and Password is the same provided through the registration process. It also retrieves the requestor public key stored in the DB.<br>Note: the password is encrypted both the one received as input from the method and the one stored in DB. The equality check is made between encrypted strings. |
| AXCSReg-DataManager::StoreData()<br>(the needful input data can be stored in the class attributes or sent as method parameters) | | Store received data in the Registration and Certification Database |

Note that it has been introduced two encrypting/decrypting methods (DecryptComm, EncryptComm) to enforce the encrypting robustness. In fact we can suppose to use an encrypted protocol (like SSL), but we can enforce encryption robustness (and therefore security) encrypting our self data too using a Public/Private key paradigm. You have to remember that public keys are stored in Registration and Certification Database.

### 13.10.1 AXMEDIS Registration of AXCSs WEB Service interface formalization

In the present paragraph is explained the Registration of AXCS Web Service interface using the WSDL formalism.

WSDL

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:rm="http://new.webservice.namespace"
targetNamespace="http://new.webservice.namespace">
    <types>
        <xs:schema targetNamespace="http://new.webservice.namespace" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
            <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
            <complexType name="usrDatatype">
                <sequence>
                    <!-- …… data according to database structure … -->
                </sequence>
            </complexType>
        </xs:schema>
    </types>
    <message name="RegistrationRequest">
        <part name="NickName" type="xs:string"/>
        <part name="Password" type="xs:string"/>
        <part name="Regdata" type="rm:usrDatatype"/>
    </message>
```

```
	<message name="RegistrationResponse">
		<part name="Result" type="xs:string"/>
		<part name="definitive-AXCSUID" element="" type="xs:string"/>
	</message>
	<portType name="Registration_PortType">
		<operation name="Registration">
			<input message="rm:RegistrationRequest"/>
			<output message="rm:RegistrationResponse"/>
		</operation>
	</portType>
	<binding name="Registration" type="rm:Registration_PortType">
		<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
		<operation name="Registration">
			<soap:operation soapAction="urn:#Registration"/>
			<input>
				<soap:body use="literal"/>
			</input>
			<output>
				<soap:body use="literal"/>
			</output>
			<soap:operation soapAction="urn:#Registration"/>
		</operation>
	</binding>
	<service name="RegistrationWebService">
		<port name="Registration" binding="rm:Registration">
			<soap:address location="No Target Adress"/>
		</port>
	</service>
</definitions>
```

Request Sample Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
	<SOAP-ENV:Body>
		<m:Registration xmlns:m="http://new.webservice.namespace">
			<NickName>…distributor user…</NickName>
			<Password>…password…</Password>
			<Regdata>
				<!-- …… data according to database structure … -->
			</Regdata>
		</m:Registration>
	</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response Sample Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
	<SOAP-ENV:Body>
		<m:Registration xmlns:m="http://new.webservice.namespace">
			<Result>0</Result>
			<definitive-AXCSID>0A2Z4X678B0124C56X89W123452CV478</definitive-AXCSID>
		</m:Registration>
	</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# 14 AXMEDIS User Registration Portal (DSI)

| Module/Tool Profile | | |
|---|---|---|
| AXMEDIS User Registration Portal | | |
| Responsible Name | Chellini, Martini | |
| Responsible Partner | DSI | |
| Status (proposed/approved) | Proposed | |
| Implemented/not implemented | Implemented | |
| Status of the implementation | In refinement | |
| Executable or Library/module (Support) | Web site | |
| Single Thread or Multithread | Multithred | |
| Language of Development | PHP | |
| Platforms supported | Anyone on which can be installed a web server capable of PHP processing. It has been tested using Apache v.2 | |
| Reference to the AXFW location of the source code demonstrator | | |
| Reference to the AXFW location of the demonstrator executable tool for internal download | | |
| Reference to the AXFW location of the demonstrator executable tool for public download | | |
| Address for accessing to WebServices if any, add accession information (user aNd Passwd ) if any | http://flauto.dsi.unifi.it/general_registration/registration1.php | |
| Test cases (present/absent) | Absent | |
| Test cases location | | |
| Usage of the AXMEDIS configuration manager (yes/no) | | |
| Usage of the AXMEDIS Error Manager (yes/no) | | |
| Major Problems not solved | None | |
| Major pending requirements | --<br>-- | |
| | | |
| 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.) |
| | AXCSUserRegistration Web Service | http, smtp |
| | Smtp service | |
| | | |
| | | |
| Formats Used | Shared with | format name or reference to a section |
| | | |
| | | |

| | | |
|---|---|---|
| | | |
| Protocol Used | Shared with | Protocol name or reference to a section |
| AXCS User Registration | | AXCSUserRegistration (see section 36) |
| | | |
| | | |
| | | |
| Used Database name | | |
| User Registration database for user registration portal | | General_registration (see section 20) |
| | | |
| | | |
| | | |
| | | |
| User Interface | Development model, language, etc. | Library used for the development, platform, etc. |
| Web interface (HTML) | PHP, HTML | |
| | | |
| | | |
| | | |
| | | |
| Used Libraries | Name of the library and version | License status: GPL. LGPL. PEK, proprietary, authorized or not |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

## 14.1 General Description of the Module

The AXMEDIS User Registration portal is a web site to let users register in AXMEDIS system.
It is provided in order to support distributors who do not want to manage user registrations in AXMEDIS, to not acquire the needed interface toward the AXCS in order to minimize the impact of the AXMEDIS system in their business structures. Therefore, this web site provide the users with the necessary data (an AXUID and a user certificate) needed in successive transactions with distributors. Once a user has registered him/herself, the certificate obtained can be used to establish secure (SSL) connections to distributor web sites.

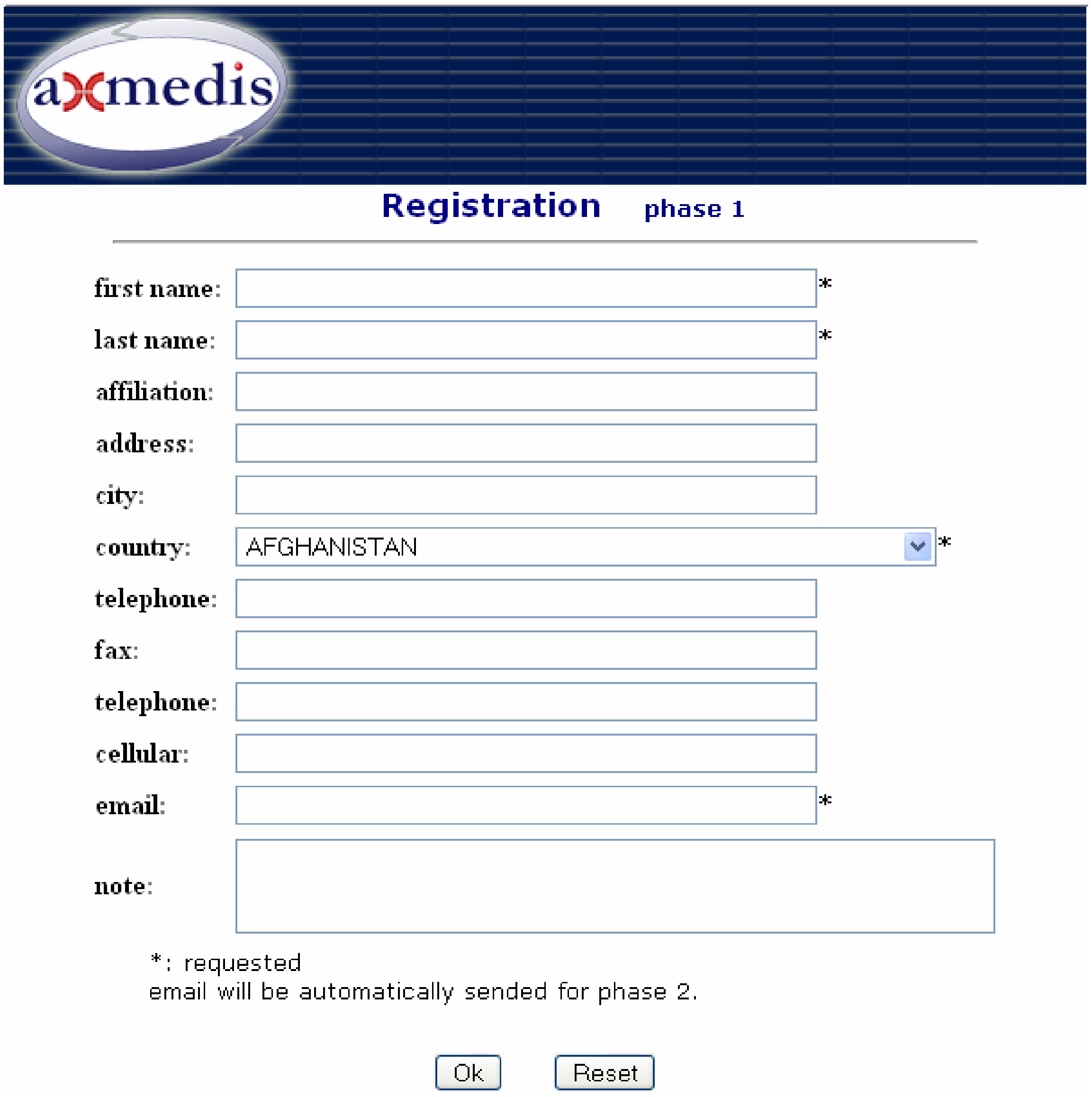
## 14.2 Module Design in terms of Classes

This module has been developed using PHP and HTML, therefore no object oriented approach was used. Rather, the module has the typical structure of web sites, i.e. single pages presenting to the user a graphic interface useful to collect needed data.

## 14.3 User interface description

The registration process goes through three successive phases:

- **Phase 1:** the user provides his/her data (such as name, affiliation, address, email, etc.) and the portal sends an email to the user.

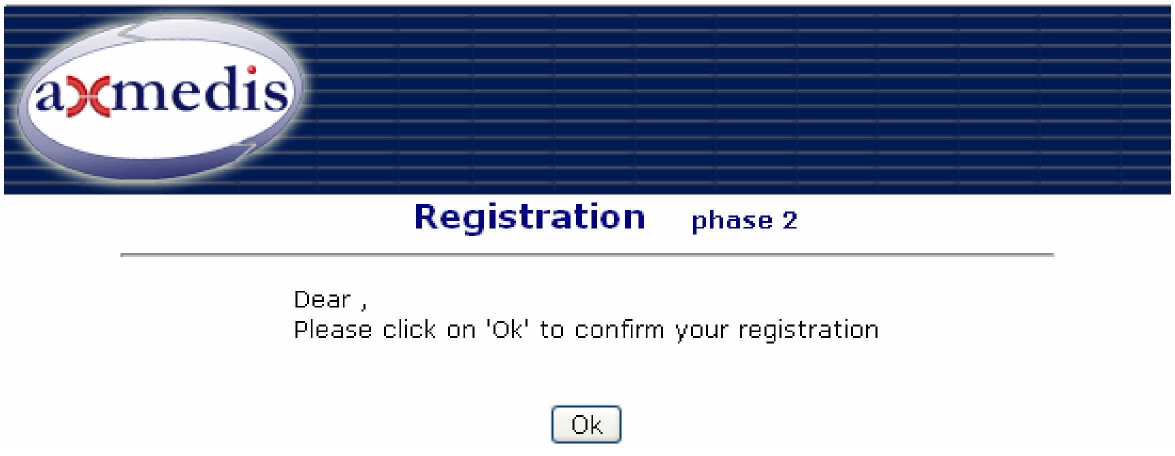


Registration phase 1

| | | |
|---|---|---|
| first name: | | * |
| last name: | | * |
| affiliation: | | |
| address: | | |
| city: | | |
| country: | AFGHANISTAN | * |
| telephone: | | |
| fax: | | |
| telephone: | | |
| cellular: | | |
| email: | | * |
| note: | | |

*: requested
email will be automatically sended for phase 2.

- **Phase 2:** the user registration willing has to be confirmed by the user him/her-self. To this end, the email sent to the user contains the address of a confirmation page. So, all that is needed is the user opening the correspondent web page in a browser and clicking the confirmation button..



Dear ,
Please click on 'Ok' to confirm your registration

- **Phase 3:** At this time the portal uses the AXCS User Registration Web Service to register the user in the AXMEDIS system. In this way an AXUID is assigned to the user and a pertinent certificate is produced and sent to the user

## 14.4 Technical and Installation information

Since the User Registration Portal is a web site, it has to be hosted on a web server capable of PHP processing on the chosen platform (e.g. Apache, MS IIS, iPlanet, etc.)

| | |
|---|---|
| References to other major components needed | User Registration database for User Registration Portal, AXCS UserRegistration Web service endpoint |
| Problems not solved | None |
| Configuration and execution context | Platform used in development: Apache 2.0, PHP 4.1 |

## 14.5 Draft User Manual

The usage of the User Registration Portal is very easy and intuitive. For a step-by-step procedure see “User interface description” section

## 14.6 Examples of usage

See “User interface description” section

## 14.7 Integration and compilation issues

Since the User Registration Portal is developed in PHP and HTML, it should work on every web server capable of PHP processing. It has been developed and tested on Apache 2.0 with PHP 4.1 module

## 14.8 Configuration Parameters

| Config parameter | Possible values |
|---|---|
| None | None |

## 14.9 Errors reported and that may occur

| Error code | Description and rationales |
|---|---|

| | |
|---|---|
| None | None |

## 14.10 Formal description of algorithm Self User Registration

The registration process goes through three successive phases:

- **Phase 1:** the user provides his/her data (such as name, affiliation, address, email, etc.) and the portal inserts him/her in its User Registration database (different from the AXCS User Registration database) marking the record as "to be confirmed". In the meanwhile an email is sent to the user.
- **Phase 2:** the user registration willing has to be confirmed by the user him/her-self (in order to protect the system from automatic registration processes). To this end, the email sent to the user contains the address of a confirmation page specific for the user (identified by means of an appropriate code assigned by the portal). So, all that is needed is the user opening the correspondent web page in a browser and click the confirmation button. In this way the pertinent record in the database is marked as “confirmed” and the process advances to phase 3.
- **Phase 3:** At this time the portal uses the AXCS User Registration Web Service to register the user in the AXMEDIS system. In this way an AXUID is assigned to the user and a pertinent certificate is produced and returned to the portal. The record in the portal database is marked as “accepted” and the certificate is sent to the user

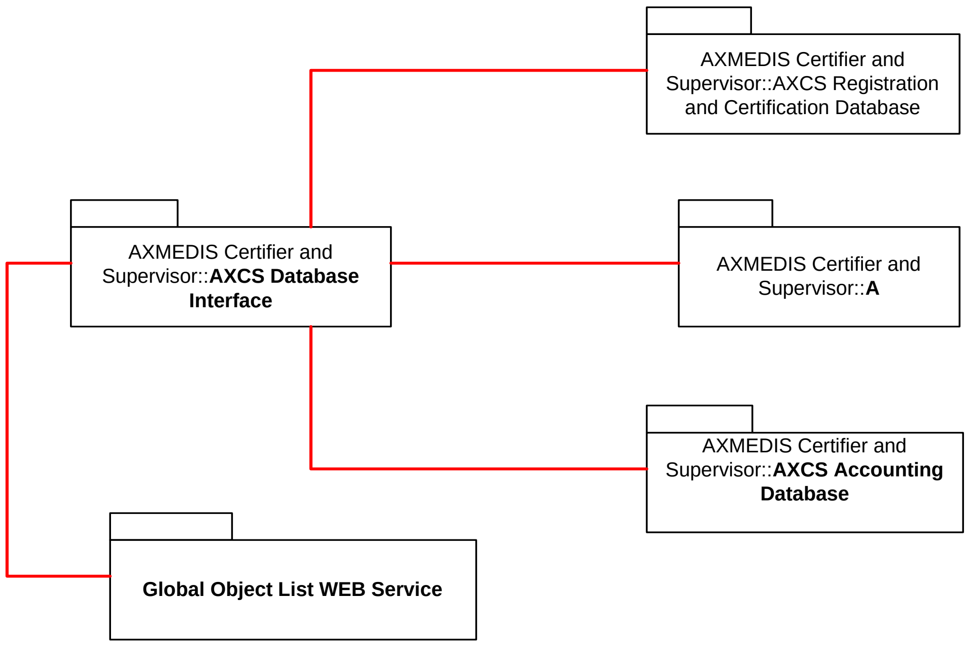
## 15 AXCS Global Object List Web Service (DSI)

| Module/Tool Profile | | |
|---|---|---|
| AXCS Global Object List Web Service | | |
| Responsible Name | Chellini, Martini | |
| Responsible Partner | DSI | |
| Status (proposed/approved) | Proposed | |
| Implemented/not implemented | Not implemented | |
| Status of the implementation | | |
| Executable or Library/module (Support) | Web service | |
| Single Thread or Multithread | Multithread | |
| Language of Development | Java | |
| Platforms supported | Anyone on which can be installed a servlet container and a JRE 1.5.0 (5.0). It has been tested using Tomcat 5.5, Axis 1.3, JRE 1.5.0_06 | |
| Reference to the AXFW location of the source code demonstrator | | |
| Reference to the AXFW location of the demonstrator executable tool for internal download | | |
| Reference to the AXFW location of the demonstrator executable tool for public download | | |
| Address for accessing to WebServices if any, add accession information (user aNd Passwd ) if any | | |
| Test cases (present/absent) | Absent | |
| Test cases location | | |
| Usage of the AXMEDIS configuration manager (yes/no) | | |
| Usage of the AXMEDIS Error Manager (yes/no) | | |
| Major Problems not solved | --<br>-- | |
| Major pending requirements | --<br>-- | |
| | | |
| 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.) |
| SuperAXCS:Database interface | | Protected (SSL) |
| Any authorized requestor (Distributors, Integrators, Creators and so on) | | |
| | | |
| | | |
| Formats Used | Shared with | format name or reference to a section |

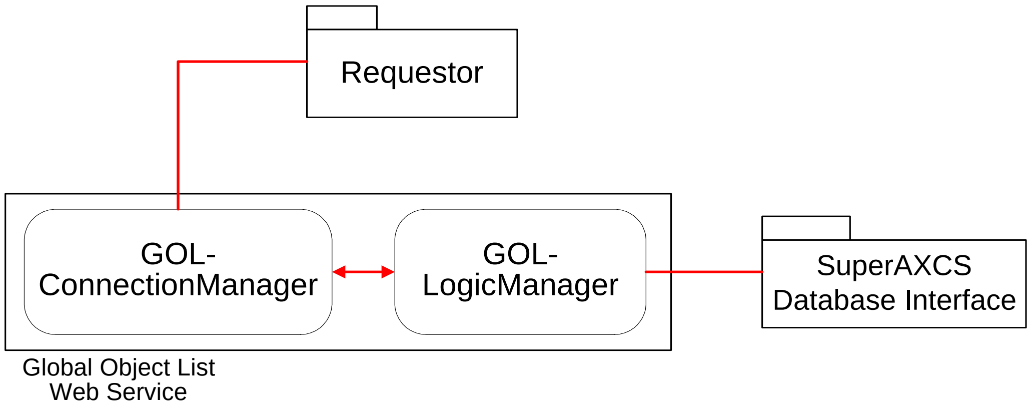
| | | |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| Protocol Used | Shared with | Protocol name or reference to a section |
| | | |
| | | |
| | | |
| | | |
| Used Database name | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| User Interface | Development model, language, etc. | Library used for the 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 Global Object List web service has been introduced with the aim of providing information about all the objects present in the AXMEDIS system. It can be queried by Distributors, Creators, Integrators, and so on (in general B2B Users). This web service could be also used as background logic for web applications implementing searching and browsing services of AXMEDIS Objects (such as searching engines about AXMEDIS Objects).


AXMEDIS Certifier and Supervisor::AXCS Registration and Certification Database
AXMEDIS Certifier and Supervisor::**AXCS Database Interface**
AXMEDIS Certifier and Supervisor::**A**
AXMEDIS Certifier and Supervisor::**AXCS Accounting Database**
**Global Object List WEB Service**

The Global Object List Web Service is a web application running on a web server implemented as a set of scripts. It has to provide to the system the possibility of retrieving information about all the objects present in the system. Given some data (tipically a set of metadata) a requestor has to be able to recover all references about objects that are compliant with the given data.

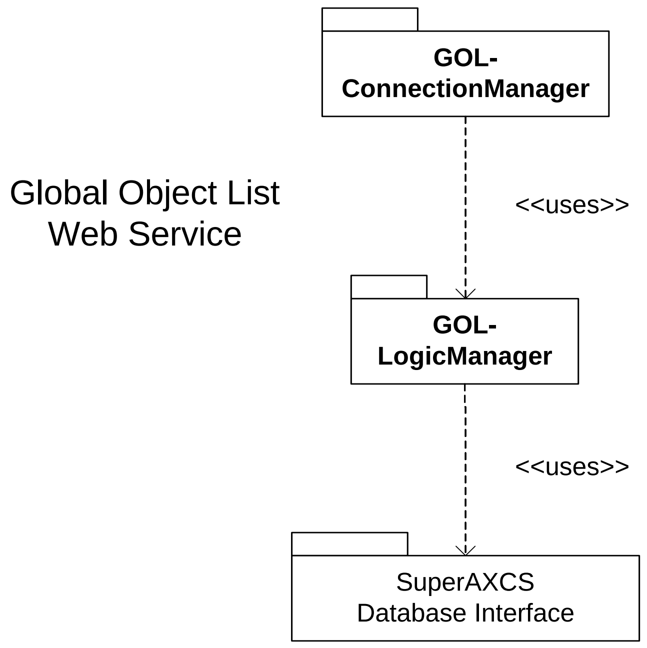


It can be identified the following logical decomposition:

**GOL-ConnectionManager**: this component manages the requests received by web service, collects data requests and response to requestors providing information compliant with requests.
**GOL-LogicManager**: this component realizes the whole application logic and uses AXCS Database Interface (and the related AXDB-API) to access database.

## 15.2 Module Design in terms of Classes

The logic at the base of the Global Object List Web Service mechanism has been subdivided in some classes in order to separate concerns and to make it easier to reuse some functionalities in other modules. The following figure shows the class diagram of the AXCS Tool Off line Registration :



## 15.3 User interface description

Not yet available

## 15.4 Technical and Installation information

Not yet available

## 15.5 Draft User Manual

Not yet available

## 15.6 Examples of usage

Not yet available

## 15.7 Integration and compilation issues

None

## 15.8 Configuration Parameters

| Config parameter | Possible values |
|---|---|
| None | None |

## 15.9 Errors reported and that may occur

| Error code | Description and rationales |
|---|---|
| None | None |

## 15.10 Formal description of algorithm Global Object List

The following table describes methods thought to be used in Global Object List Web Service:

| Method name and parameters | Output | Description |
|---|---|---|
| **scope: public** | | |
| GOL-ConnectionManager::AcceptRequest (string NickName, string Password, string RequestQuery) | String ResultStatus<br>String ResponseData | It is a public method. It collects requestor credentials needful to access the system and uses the VerifyLogin() (a GOL-LogicManager method) to verify requestor credentials. It also collects queries data (in a field called RequestQuery) .<br>This method uses private methods described below to perform its own tasks. The ResultStatus output parameter is set to 0 if the task is successful and is set to 1 otherwise. It returns query results (compliant with data request) in a field called ResponseData. String data has been chosen to have more flexibility in input and output parameters. |
| **scope: private** | | |
| GOL-ConnectionManager::EncryptComm (string ClearData, string asimKey) | String EncodedData | Encrypt ClearData with provided asymmetric key (asymKey) |
| GOL-ConnectionManager::DecryptComm (string String EncodedData, string asimKey) | String ClearData | Decrypt EncodedData with provided asymmetric key (asymKey) |
| GOL-LogicManager::Data-Retriever(string Query)<br>(the needful input data can be stored in the class attributes or sent as method parameters) | String ResultQuery | This method retrieves data from database (using SuperAXCS Database Interface API) according to its input values. The ResultQuery field returns query response data. |
| GOL-LogicManager::VerifyLogin (string NickName, string Password) | boolean IsTrusted | It accesses Registration and Certification Database to verify if the couple NickName and Password is the same provided through the registration process. |

WSDL

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:rm="http://new.webservice.namespace"
targetNamespace="http://new.webservice.namespace">
    <types>
        <xs:schema targetNamespace="http://new.webservice.namespace" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
            <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
        </xs:schema>
    </types>
    <message name="GOL-Request">
        <part name="NickName" type="xs:string"/>
        <part name="Password" type="xs:string"/>
        <part name="RequestQuery" type="xs:string"/>
    </message>
    <message name="GOL-Response">
        <part name="ResultStatus" type="xs:string"/>
        <part name="ResponseData" type="xs:string"/>
    </message>
    <portType name="GOL-PortType">
        <operation name="GOL">
            <input message="rm:GOL-Request"/>
            <output message="rm:GOL-Response"/>
        </operation>
    </portType>
    <binding name="GOList" type="rm:GOL-PortType">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GOL">
            <soap:operation soapAction="urn:# GOL"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
            <soap:operation soapAction="urn:# GOL"/>
        </operation>
    </binding>
    <service name="GOLWebService">
        <port name="GOL-port" binding="rm:GOList">
            <soap:address location="No Target Adress"/>
        </port>
    </service>
</definitions>
```

Request Sample Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAP-ENV:Body>
        <m:GOL xmlns:m="http://new.webservice.namespace">
            <NickName>… user name …</NickName>
            <Password>… relatd password …</Password>
            <RequestQuery>DCCCreatorsMetadata.CreatorValue="Mozart" and DublinCore.Language=IT
</RequestQuery>
        </m:GOL>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response Sample Message

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAP-ENV:Body>
        <m:GOL xmlns:m="http://new.webservice.namespace">
            <ResultStatus>0</ ResultStatus >
            <ResponseData>… response string …</ ResponseData >
        </m:GOL>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

# 16 Provided API named "AXCS-DB-Interface" (DSI)

Section 10 described the structure of the AXCS-DB-Interface module, illustrating how it is composed of two layers: a logical layer on top of a physical layer.
This section describes more in deep the logical layer and the methods at disposal of other AXCS modules.

The logical layer provides access to the AXCS databases in terms of object oriented representation regardless of the underlying DBMS used. Therefore, it provides abstract representations for:

- a database manager, by means of an interfacecalled *AxcsdbManager*
- objects representing database tables, by means of an interface called *DataTable*
- common complex operations on database data, grouping all the needed sub-operations together and providing high-level methods via a unique "access point", called *AxcsdbInterface*
- logically related data grouping them in single classes, such as data needed for user (*UserDataType*) or object (*ObjectDataType*) registrations and user identifying data (*UserLoginData*)

Modules using this API simply use either the *AxcsdbInterface* methods to perform their own tasks or use directly objects representing tables to realize specific complex operations. Since the most common complex operations are directly provided by *AxcsdbInterface*, tipically other modules use those methods, having to deal with the other entities only in order to properly initialize the *AxcsdbInterface* object (e.g. instantiating a suitable database manager implementing the *AxcsdbManager* interface) or to get/send data from/to requestors. So, generally, these modules do not directly manipulate objects representing tables, since this is done by the high-level methods of *AxcsdbInterface*.

| **AxcsdbInterface** | |
|---|---|
| Method | setUsrData |
| Description | Used to update user data or insert data about new users |
| Input parameters | boolean isReplace – indicates if the operation requested is an insertion or an update<br>UserDataType userData – an object containing data to be inserted/updated. Only the information corresponding to the fields not set to null are used for the operation.<br><br>Note: the AXUID field can not be updated and in case of updates it must be an existing AXUID; for insertions it is treated as a temporary ID, so it can be any string |
| Output parameters | boolean – true if the operation succeeds, false otherwise |
| Request Sample Message | setUsrData(false, new UserDataType(null, "USR_36351030-8315-6070-8280-380689855558", "546f8c8312323da", " user@fakedomain.it", "italian", "nickname", "password", "30818902818100f749ec9ad8a22482f9", "2005-01-01", "2009-31-12", "U")); |
| Response Sample Message | True |

| **AxcsdbInterface** | |
|---|---|
| Method | getUsrData |
| Description | Tries to find user data according to the input parameter |
| Input parameters | string axuid – The identifier of the user (AXUID) for whom data are to be retrieved |
| Output parameters | UserDataType – an object containing all related data about the user in case of success, null otherwise |
| Request Sample Message | getUsrData("USR_36351030-8315-6070-8280-380689855558"); |
| Response Sample Message | |

| **AxcsdbInterface** | |
|---|---|
| Method | delUsrData |
| Description | Used to delete a user according to the input parameters |
| Input parameters | string axuid – The identifier of the user (AXUID) for whom data are to be deleted |
| Output parameters | boolean – true if the operation was successful, false otherwise |
| Request Sample Message | delUsrData("USR_36351030-8315-6070-8280-380689855558"); |
| Response Sample Message | True |

| **AxcsdbInterface** | |
|---|---|
| Method | login |
| Description | Tries to perform a login to the system using the couple username and password |
| Input parameters | string name – it is the login user name (nickname)<br>string passwd – it is the login password |
| Output parameters | UserLoginData – an object containing data related to the user (AXUID and public key), or null if the login was unsuccessful |
| Request Sample Message | login("nickname", "password"); |
| Response Sample Message | |

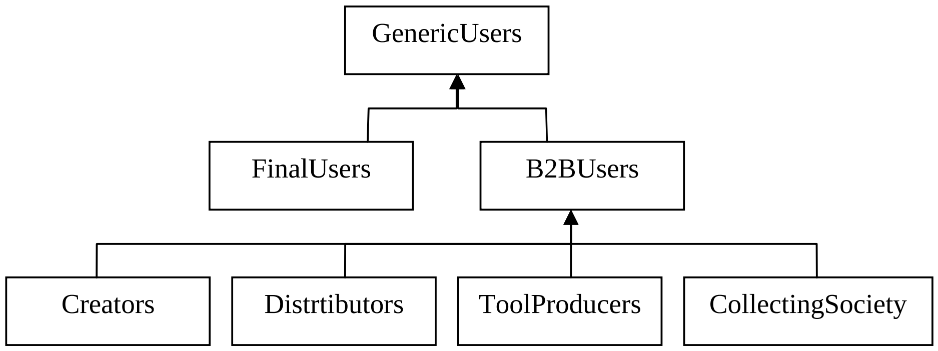
Methods defined in *DataTable* interface (therefore available in each class mapping database tables) have been already described in subsection 10.10 (where they are named as "Generica class methods"), therefore they are not reported here.
Methods defined in *AxcsdbManager* (therefore available in each class representing a specific dabase manager, e.g. *MysqlAxcsdbManager*) are those implemented in the physical layer of the AXCS Database Interface module and have been already described in section 10.10 (where they are named as "Physical layer interface methods"). Therefore, they are not reported here.

A detailed description of each method is provided in the Javadoc documentation available at https://cvs.axmedis.org/repos/Framework/doc/code/axcs.

# 17 Table description for AXCS Registration and certification database (DSI)

## 17.1 Entity-Relationship description

In order to define the AXCS Registration and Certification Database schema, first we have to identify the entities and related relations.
Since some different categories of users have to be represented, an ISA hierarchy have been conceived to appropriately reflect all the different kinds of users. This hierarchy is reported below:



Please note that this hierarchy is intended in terms of ISA relations among database tables, therefore it is enforced through referential integrity constraints.
Here is reported the list of the identified entities and the related meanings.

1. **GenericUsers**: this entity contains a part of data about AXMEDIS users common to all the specific kinds of users (see below) and requested by AXCS to perform its work.
2. **FinalUsers**: this entity represents end users of the AXMEDIS objects and is a specialization of the GenericUsers entity. It contains more details on the status and the registration date of an end user.
3. **B2BUsers**: this entity contains general data about B2B users like Creators, Distributors, Collecting Society, Tool Producers and so on. It does not contains information on the registration date and the status of a B2BUser because these data are related to the specific kind of user (Creator, Distributor, etc.).
4. **Domains**: this entity contains information about AXMEDIS Domains. A Domain can be referred to a FinalUser (if the PMS is for private use at home) or to a B2BUser (if the PMS is located in an organization like a company or a school). It is linked to the parent entity FinalUser or B2BUsers according to the prefix of the AXUID field which identifies if the AXUID is relative to a FinalUser or a to a B2BUser.
5. **Creators**: this entity contains specific data about Object Creators, Integrators and Producers. It is linked to the parent entity B2BUsers.
6. **Distributors**: this entity contains specific data about Object Distributors. It is linked to the parent entity B2BUsers.
7. **CollectSoc**: this entity contains data about Collecting Societies. It is linked to the parent entity B2BUsers.
8. **ToolProducers**: this entity contains data about Tool Producers. It is linked to the parent entity B2BUsers.
9. **RegTools**: this entity contains data about Registered Tools. The “registration” term refers to Tool Off-line Registration scenario. A registered tool is a software product. An instance of a Registered Tool running on a terminal becomes a Certified Tool. A registered tool is identified by an ID called AXRTID.
10. **TypeOfTool**: this entity contains all the types of tool that can be used in the AXMEDIS system. A set of possible tool types is:

    Composition Engine

Formatting Engine
Editor PC/MAC
Viewer PC/MAC
Viewer/player: PDA
Viewer/player: mobile
Protection Tool Editor
AXEPTool
Programme & Publication Engine
Publication Tool
Generator from CMS
AXCS
Super AXCS
AXMEDIS OID Generator
AXMEDIS PMS Client
AXMEDIS PMS Home
AXMEDIS PMS Server
PLUGIN xxxxx
PLUGIN yyyyy

11. **CerTools**: this entity contains data about Certified Tools. The "certification" term refers to Certification of a Tool/User scenario. A certified tool is an instance of a tool running on a terminal. A certified tool is identified by an ID called AXTID field. It is linked to the parent entity User or B2BUsers according to the value of TypeOdID which identifies if the relative AXID is a AXUID or a B2BUserID.

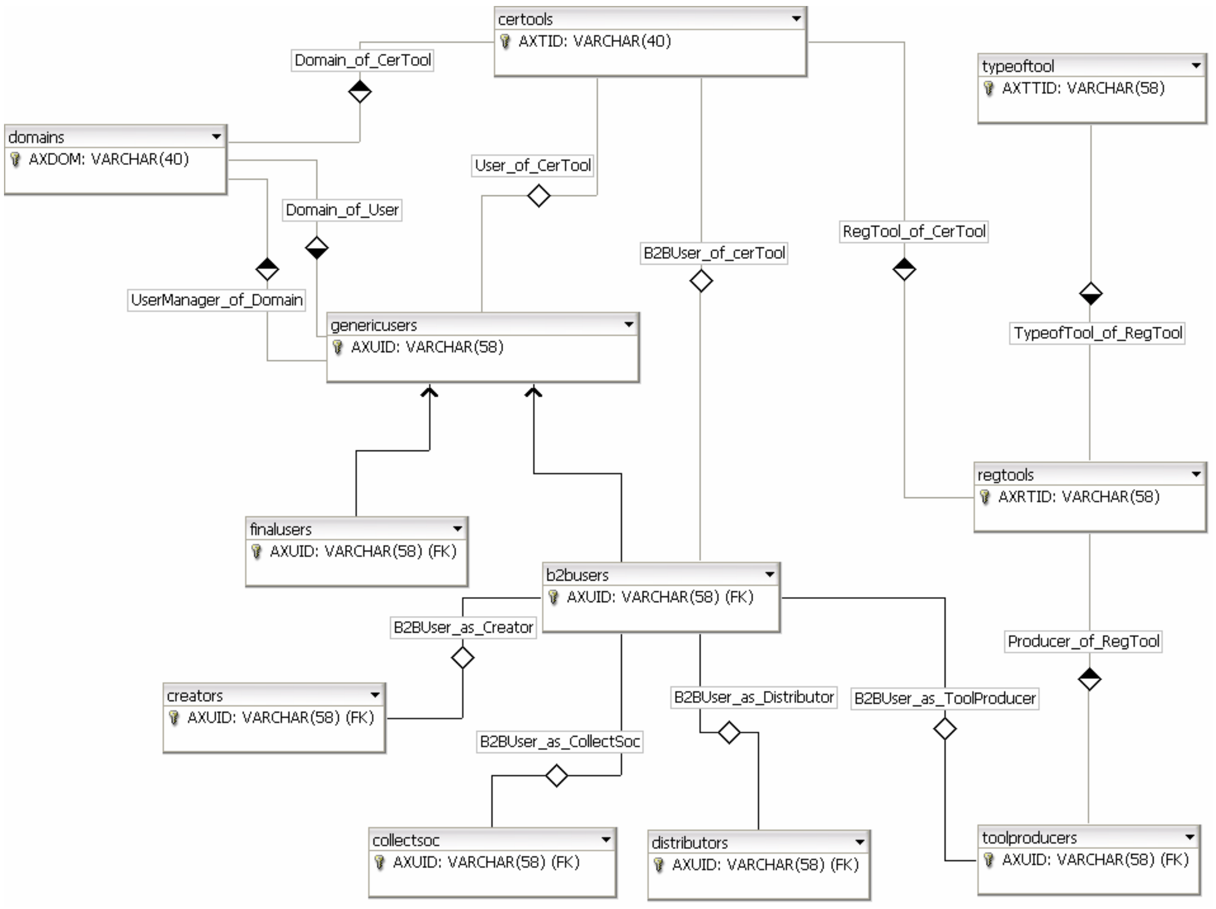
Every entity is assigned an identifier, which is named accordingly to the name of the related entity.
For the GenericUsers entity this identifier is called AXUID; this is because the GenericUsers entity represents the concept of user without deeper distinctions. In this table B2BUsers can be distinguished from FinalUsers on the basis of the prefix of the AXUID field: for the former its value is "BUS", for the latter its value is "USR".
The B2BUsers groups all the data common to the different business users. For this reason the identifier used in this entity is also called AXUID and it takes the same value of the corresponding record in the GenericUsers table. The specification of the role of this business user is then stored in the related entity (Creators, Distributors, CollectingSociety or ToolProducers), which reports also the same identifier used in the B2BUser table. A B2BUser can have more than one role in AXMEDIS: for example, he can be either a Creator, a Distributor and a ToolProducer
To have an example, consider a B2BUser who is either a Creator and a Distributor of objects. In this case, in the GenericUsers, B2BUsers, Creators and Distribitors tables we would have the following identifier:

AXUID = BUS_1b4e28ba-2fa1-11d2-883f-b9a761bde3fb

In the case of a final users, instead, in the GenericUsers and FinalUsers tables we would have the following identifier:

AXUID = USR_1b4e28ba-2fa1-11d2-883f-b9a761bde3fb

Please note that a user cannot be at the same time a Final User and a B2BUser. In this way it is not possible to have a USR prefixed AXUID and a BUS prefixed AXUID with the same UUID. For example the following situation will be not allowed:
USR_1b5e28ba-2fa1-11d2-883f-b9a761bde3fb
BUS_1b5e28ba-2fa1-11d2-883f-b9a761bde3fb



## 17.2 Relational database schema extended description

Here is reported the list of identified tables came from entities and relations previously stated.

### GenericUsers

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXUID** | **PK** | **I** | VARCHAR(58) | Not allowed | |
| **AXDOM** | **FK** | **I** | VARCHAR(58) | Allowed | |
| **Email** | | **I** | VARCHAR(255) | Not allowed | |
| **NickName** | | **I** | VARCHAR(255) | Not allowed | |
| **Password** | | | VARCHAR(255) | Not allowed | |
| **Nationality** | | **I** | VARCHAR(255) | Not allowed | |
| **PubKey** | | | LONGTEXT | Not allowed | |
| **CertificateSerialNumber** | | **I** | VARCHAR(255) | Not allowed | |
| **Status** | | | VARCHAR(1) | Not allowed | B/U (Blocked/Unblocked) |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXDOM** | AXDOM | Domains.AXDOM |

| Column details | |
|---|---|
| **1. AXUID** (PK) | |
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the user |

**2. AXDOM** (FK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Allowed |
| **Notes:** | AXMEDIS Current Domain of the user (if any) |

**3. Email**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | email of the user |

**4. NickName**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | NickName of the user |

**5. Password**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | MD5 or other encryption of user password |

**6. Nationality**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Nationality of the user |

**7. PubKey**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | public key of the user |

**8. CertificateSerialNumber**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | user certificate serial number |

**9. Status**

| | |
|---|---|
| **Physical data type:** | VARCHAR(1) |
| **Allow NULLs:** | Not Allowed |
| **Notes:** | Status of the user: B/U (Blocked/Unblocked) |

Concerning the status of B2BUsers it is necessary to make a distinction. On one side it has to be considered the right to use AXMEDIS services (e.g. object registration, user registration, reporting, etc.). On the other side it has to be considered the right to perform actions over objects. In this case, if an improper behaviour is detected, the user has to be blocked regardless of the role he's currently playing. Therefore a higher level of user blocking has been introduced by means of the "status" field in the genericusers table which is independent from the specific business role status fields. In this way, if an user is blocked at this higher level, he is not allowed to perform any action over objects. Anyway, he can continue to use AXMEDIS services (e.g. reporting or statistics) if he is not blocked as a specific business user role (such as Creator, Distributor, Collecting Society and Tool Producer). To block a B2BUser as a specific role a status field in every specific role tables has been introduced.
In the same way, if an user is blocked as a specific business role, he can continue to perform actions over objects but he is not allowed anymore to exploit some AXMEDIS services.

## FinalUsers

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXUID** | **PK, FK** | **I** | VARCHAR(58) | Not allowed | |
| **RegDate** | | | DATETIME | Not allowed | |
| **RegDeadline** | | | DATETIME | Not allowed | |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXUID** | AXUID | GenericUsers.AXUID |

**Column details**

**1. AXUID** (PK, FK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the related final user |

**2. RegDate**

| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of user registration, referred to GMT+0 |

**3. RegDeadline**

| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of user registration end, referred to GMT+0 |

## B2BUsers

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXUID** | **PK, FK** | **I** | VARCHAR(58) | Not allowed | |
| **TypeOfUser** | | **I** | VARCHAR(4) | Not allowed | |
| **Website** | | | LONGTEXT | Allowed | |
| **RefName** | | **I** | VARCHAR(255) | Allowed | |
| **Phone** | | | VARCHAR(255) | Allowed | |
| **Company** | | **I** | VARCHAR(255) | Not allowed | |
| **CmpAddress** | | | VARCHAR(255) | Allowed | |
| **CmpPhone1** | | | VARCHAR(255) | Allowed | |
| **CmpPhone2** | | | VARCHAR(255) | Allowed | |
| **CmpFax** | | | VARCHAR(255) | Allowed | |
| **Location** | | | VARCHAR(255) | Allowed | |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXUID** | AXUID | GenericUsers.AXUID |
| **AXUID** | ToolProducers. AXUID | AXUID |
| **AXUID** | Creators. AXUID | AXUID |
| **AXUID** | Distributors. AXUID | AXUID |

**Column details**

**1. AXUID** (PK, FK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the related B2B user |

**2. TypeOfUser**

| | |
|---|---|
| **Physical data type:** | VARCHAR(4) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Encode of the specific type of B2B User. It is a positional encode, composed by four chars each representing a specific specialization. For example, xxx1 means the user is a creator and xxx0 means the user isn't a creator. Please note the user can have more than one specialization, so the code can have more than one character set to '1'. The meaning of the positions is below. |

**3. Website**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Web site of the B2B user |

**4. RefName**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Reference name in the B2B company |

**5. Phone**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Allowed |

| | |
|---|---|
| **Notes:** | Reference telephone number in the B2B company |

**6. Company**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Name of the B2B company |

**7. CmpAddress**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Address of the B2B company |

**8. CmpPhone1**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Phone number of the B2B company |

**9. CmpPhone2**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Phone number of the B2B company |

**10. CmpFax**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Fax number of the B2B company |

**11. Location**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Location of the B2B user |

Here is the encoding of the possible B2B user specializations:

| TypeOfUser | Encode |
|---|---|
| Creator | xxx1 |
| Distributor | xx1x |
| Collecting Society | x1xx |
| Tool Producer | 1xxx |

Examples: 0011 means the user is a Creator and a Distributor. 1001 means the user is a Creator and a Tool Producer.

## Domains

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXDOM** | **PK** | **I** | VARCHAR(58) | Not allowed | |
| **AXUID** | **FK** | **I** | VARCHAR(58) | Not allowed | |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXUID** | AXUID | GenericUsers.AXUID |
| **AXDOM** | CerTools.AXDOM | AXDOM |
| **AXDOM** | Users.AXDOM | AXDOM |

**Column details**

**1. AXDOM** (PK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the AXMEDIS Domain |

**2. AXUID** (FK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |

|  |  |
|---|---|
| **Allow NULLs:** | Not allowed |
| **Notes:** | It’s the Domain Manager ID: it can be an ID of a B2BUser or an End User according to the value of TypeOfID |

## Creators

| Columns |  | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXUID** | **PK, FK** | **I** | VARCHAR(58) | Not allowed |  |
| **RegDate** |  |  | DATETIME | Not allowed |  |
| **RegDeadline** |  |  | DATETIME | Not allowed |  |
| **Status** |  |  | VARCHAR(1) | Not allowed | B/U (Blocked/Unblocked) |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXUID** | AXUID | B2BUsers.AXUID |

**Column details**

**1. AXUID** (PK, FK)

|  |  |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the related B2B user |

**2. RegDate**

|  |  |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of creator registration, referred to GMT+0 |

**3. RegDeadline**

|  |  |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of creator registration end, referred to GMT+0 |

**4. Status**

|  |  |
|---|---|
| **Physical data type:** | VARCHAR(1) |
| **Allow NULLs:** | Not Allowed |
| **Notes:** | Status of the creator: B/U (Blocked/Unblocked) |

## Distributors

| Columns |  | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXUID** | **PK, FK** | **I** | VARCHAR(58) | Not allowed |  |
| **RegDate** |  |  | DATETIME | Not allowed |  |
| **RegDeadline** |  |  | DATETIME | Not allowed |  |
| **Status** |  |  | VARCHAR(1) | Not allowed | B/U (Blocked/Unblocked) |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXUID** | AXUID | B2BUsers.AXUID |

**Column details**

**1. AXUID** (PK, FK)

|  |  |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the related B2B user |

**2. RegDate**

|  |  |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of distributor registration, referred to GMT+0 |

**3. RegDeadline**

|  |  |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |

| | |
|---|---|
| **Notes:** | Timestamp of distributor registration end, referred to GMT+0 |

**4. Status**

| | |
|---|---|
| **Physical data type:** | VARCHAR(1) |
| **Allow NULLs:** | Not Allowed |
| **Notes:** | Status of the distributor: B/U (Blocked/Unblocked) |

## CollectSoc

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXUID** | **PK, FK** | **I** | VARCHAR(58) | Not allowed | |
| **NationDomain** | | **I** | VARCHAR(255) | Not allowed | |
| **RegDate** | | | DATETIME | Not allowed | |
| **RegDeadline** | | | DATETIME | Not allowed | |
| **Status** | | | VARCHAR(1) | Not allowed | B/U (Blocked/Unblocked) |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXUID** | AXUID | B2BUsers.AXUID |

**Column details**

**1. AXUID** (PK, FK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the related B2B user |

**2. NationDomain**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Nation related to the collecting society. |

**3. RegDate**

| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of collecting society registration, referred to GMT+0 |

**4. RegDeadline**

| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of collecting society registration end, referred to GMT+0 |

**5. Status**

| | |
|---|---|
| **Physical data type:** | VARCHAR(1) |
| **Allow NULLs:** | Not Allowed |
| **Notes:** | Status of the collecting society: B/U (Blocked/Unblocked) |

## ToolProducers

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXUID** | **PK, FK** | **I** | VARCHAR(58) | Not Allowed | |
| **RegDate** | | | DATETIME | Not allowed | |
| **RegDeadline** | | | DATETIME | Not allowed | |
| **Status** | | | VARCHAR(1) | Not allowed | B/U (Blocked/Unblocked) |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXUID** | AXUID | B2BUsers.AXUID |
| **AXTPID** | RegTools.AXTPID | AXTPID |

**Column details**

**1. AXUID** (PK, FK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |

|  |  |
|---|---|
| **Notes:** | ID of the related B2B user |

**<u>2. RegDate</u>**

|  |  |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of tool producer registration, referred to GMT+0 |

**<u>3. RegDeadline</u>**

|  |  |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of tool producer registration end, referred to GMT+0 |

**<u>5. Status</u>**

|  |  |
|---|---|
| **Physical data type:** | VARCHAR(1) |
| **Allow NULLs:** | Not Allowed |
| **Notes:** | Status of tool producer: B/U (Blocked/Unblocked) |

## RegTools

*(This table refers to Tool Off-line Registration)*

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXRTID (PK)** | **PK** | **I** | VARCHAR(58) | Not allowed | |
| **AXTTID (FK)** | **FK** | **I** | VARCHAR(58) | Not allowed | |
| **AXTPID (FK)** | **FK** | **I** | VARCHAR(58) | Not allowed | |
| **Version** | | | VARCHAR(20) | Not allowed | |
| **Year** | | | VARCHAR(4) | Not allowed | |
| **Language** | | **I** | VARCHAR(255) | Not allowed | |
| **OS** | | **I** | VARCHAR(255) | Not allowed | |
| **SWFingerprint** | | | LONGTEXT | Not allowed | |
| **RegDate** | | | DATETIME | Not allowed | |
| **RegDeadline** | | | DATETIME | Not allowed | |
| **Status** | | | VARCHAR(1) | Not allowed | B/U (Blocked/Unblocked) |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXTTID** | AXTTID | TypeOfTool.AXTTID |
| **AXTPID** | AXTPID | ToolProducers.AXTPID |
| **AXRTID** | CerTools.AXRTID | AXRTID |

**Column details**

**<u>1. AXRTID</u>** (PK)

|  |  |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the registered tool (ID of the the class of registered tool given by SuperAXCS) |

**<u>2. AXTTID</u>** (FK)

|  |  |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the type of tool |

**<u>3. AXTPID</u>** (FK)

|  |  |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the tool producer |

**<u>4. Version</u>**

|  |  |
|---|---|
| **Physical data type:** | VARCHAR(20) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | version of the registered tool |

**<u>5. Year</u>**

|  |  |
|---|---|
| **Physical data type:** | VARCHAR(4) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | year of the registered tool |

| | |
|---|---|
| **6. Language** | |
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | language of the registered tool |

| | |
|---|---|
| **7. OS** | |
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Operative System of the registered tool |

| | |
|---|---|
| **8. SWFingerprint** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Software fingerprint of the registered tool |

| | |
|---|---|
| **9. RegDate** | |
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of registered tool registration, referred to GMT+0 |

| | |
|---|---|
| **10. RegDeadline** | |
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of registered tool registration end, referred to GMT+0 |

| | |
|---|---|
| **11. Status** | |
| **Physical data type:** | VARCHAR(1) |
| **Allow NULLs:** | Not Allowed |
| **Notes:** | Status of tool producer: B/U (Blocked/Unblocked) |

## TypeOfTool

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXTTID (PK)** | **PK** | **I** | VARCHAR(58) | Not allowed | |
| **Description** | | **I** | VARCHAR(255) | Not allowed | |
| **Notes** | | | LONGTEXT | Allowed | |

**Column details**

| | |
|---|---|
| **1. AXTTID** (PK) | |
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the type of tool |

| | |
|---|---|
| **2. Description** | |
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Description of the type of tool. |

| | |
|---|---|
| **3. Notes** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | additional information about the type of tool |

## CerTools

*(This table refers to the "Certification of a Tool" scenario)*

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXTID** | **PK** | **I** | VARCHAR(58) | Not allowed | |
| **AXRTID** | **FK** | **I** | VARCHAR(58) | Not allowed | |
| **AXUID** | **FK** | **I** | VARCHAR(58) | Not allowed | |
| **AXDOM** | **FK** | **I** | VARCHAR(58) | Allowed | |
| **HWFingerprint** | | | LONGTEXT | Not allowed | |
| **HWFingerprintDigest** | | | LONGTEXT | Not allowed | |

| | | | | | |
|---|---|---|---|---|---|
| **EnablingCode** | | | LONGTEXT | Not allowed | |
| **LastFPPA** | | | LONGTEXT | Not allowed | |
| **PubKey** | | | LONGTEXT | Not allowed | |
| **CertificateSerialNumber** | | | VARCHAR(255) | Not allowed | |
| **RegDate** | | | DATETIME | Not allowed | |
| **RegDeadline** | | | DATETIME | Not allowed | |
| **Status** | | | VARCHAR(1) | Not allowed | B/U (Blocked/Unblocked) |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXRTID** | AXRTID | RegTools.AXRTID |
| **AXDOM** | AXDOM | Domains.AXDOM |
| **AXUID** | AXUID | GenericUsers.AXUID |

**Column details**

**1. AXTID** (PK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the certified tool (the single instance of the installed tool) |

**2. AXRTID** (FK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the related registered tool (ID of the the class of registered tool given by SuperAXCS) |

**3. AXUID** (FK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | related user ID: it can be an ID of a B2BUser or an End User according to the value of TypeOfID |

**4. AXDOM** (FK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Allowed |
| **Notes:** | AXMEDIS Domain of certified tool (if any) |

**5. HWFingerprint**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | HW fingerprint of the related terminal (PC or anything else) the software is running on |

**6. HWFingerprintDigest**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Digest of HW fingerprint of the related terminal (PC or anything else) the software is running on |

**7. EnablingCode**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | An activation code provided by AXCS to Certified Tool during the initialization phase. |

**8. LastFPPA**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | LastFPPA: Last Fingerprint of Performed Actions |

**9. PubKey**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | public key of the certified tool |

**10. CertificateSerialNumber**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | tool certificate serial number |

**11. RegDate**

| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of certified tool registration, referred to GMT+0 |

**12. RegDeadline**

| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of certified tool registration end, referred to GMT+0 |

**13. Status**

| | |
|---|---|
| **Physical data type:** | VARCHAR(1) |
| **Allow NULLs:** | Not Allowed |
| **Notes:** | Status of certified tool: B/U (Blocked/Unblocked) |

# 18 Table description for AXCS ObjectsID database (DSI)

## 18.1 Entity-Relationship description

In order to define the AXCS ObjectsID database schema, first we have to identify the entities and related relations. Here is reported the list of the identified entities and the related meaning.

**Objects**: this entity contains a part of data about objects. It is linked to Creators entity (located in the AXCS Registration and Certification Database) with an 1-N relation and to the Distributors entity (also located in the AXCS Registration and Certification Database) with an M-N relation implemented as a table called **DistributedBy**. It means that an Object can be distributed by more than one Distributor. It is also linked to itself with an N-M relation implemented as a table called **ComposedBy** to state that an Object can be a Complex Object composed by other Objects.

**GeneratedIDs**. This entity contains the AXOIDs generated by the GernerateAxoid method included in Object Registration Web Service. It is needed to be sure that the AXOID it is going to be delivered to a requestor has not been already generated.

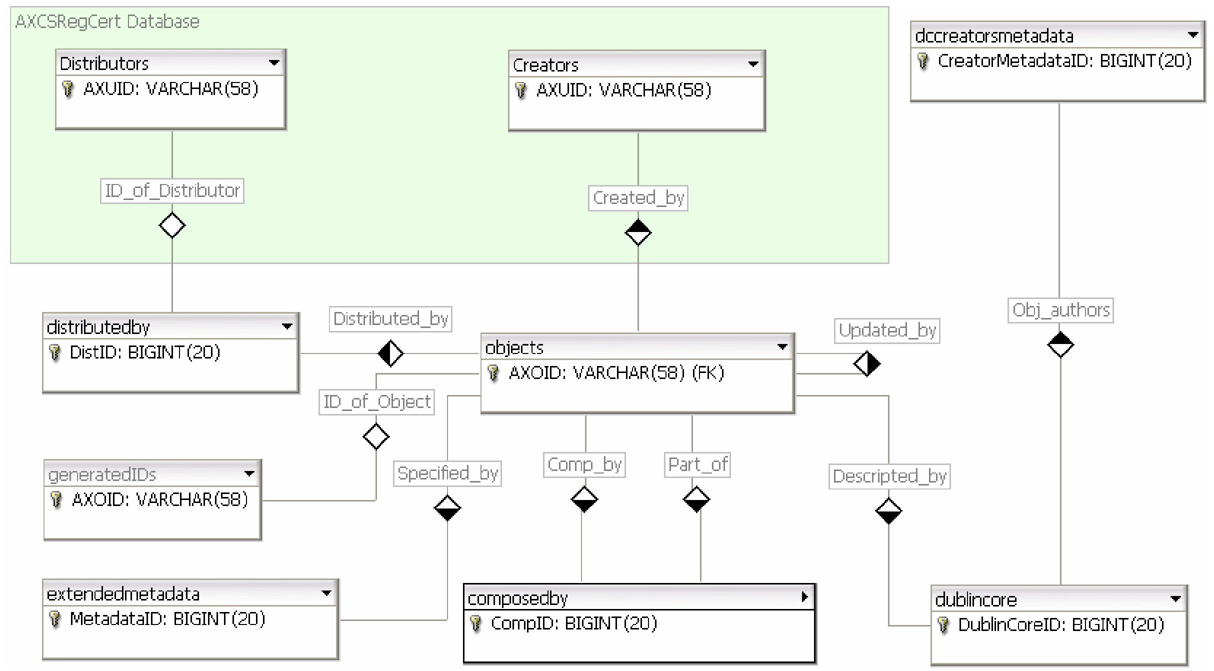
**DublinCore**: this entity contains Dublin Core metadata related to an Object. Every raw in the pertinent relational schema table is a Dublin Core set of metadata description related to an Object expressed in a specific language.

**DCCreatorsMetadata**: this entity contains data about Authors, pertinent to a specified set of Dublin Core metadata related to a language. It has been introduced to take into account the more than one Author multiplicity. It is linked to the **DublinCore** entity with an 1-N relation.

**ExtendedMetadata**: this entity contains optional metadata about Object not included in Dublin Core. Every row in the pertinent relational schema table is a single metadata value. This is a way to have a variable number of metadata fields related to every object.

## 18.2 Relational database schema extended description

Here is reported the list of identified tables came from entities and relations previously stated.



Please note that in ObjectsID database, the axmedis identifiers assigned to users are called with various names to distinguish the role played by an user in this case. These names are: AXCID (AXmedis Creator ID), AXDID (AXmedis Distributors ID), AXCSID (AXmedis Collecting Society ID).

## Objects

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXOID** | **PK** | **I** | VARCHAR(58) | Not allowed | |
| **ObjectVersion** | | **I** | VARCHAR(20) | Allowed | |
| **ProtectionStamp** | | **I** | VARCHAR(255) | Allowed | |
| **ObjectNewAxoid** | **FK** | **I** | VARCHAR(58) | Allowed | |
| **ObjectStatus** | | | VARCHAR(1) | Not allowed | B/U (Blocked/ Unblocked) |
| **AXWID** | **FK** | **I** | VARCHAR(58) | Not allowed | |
| **AXCID** | **FK** | **I** | VARCHAR(58) | Not allowed | |
| **FingerprintInfo** | | **I** | VARCHAR(255) | Not allowed | |
| **ProtectionInfo** | | | LONGTEXT | Not allowed | |
| **RegDate** | | | DATETIME | Not allowed | |
| **RegDeadline** | | | DATETIME | Allowed | |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXOID** | AXOID | GeneratedIDs.AXOID |
| **ObjectNewAxoid** | ObjectNewAxoid | AXOID |
| **AXOID** | Dublincore.AXOID | AXOID |
| **AXOID** | DistributedBy.AXOID | AXOID |
| **AXOID** | ExtendedMetadata.AXOID | AXOID |

| Column details | |
|---|---|
| **1. AXOID** (PK, FK) | |
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the Object |
| **2. ObjectVersion** | |
| **Physical data type:** | VARCHAR(20) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Current version of the object |
| **3. ProtectionStamp** | |
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Indicates the way to protect the related object. |
| **4 ObjectNewAXOID** (FK) | |
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Eventually new version of the object (if any) |
| **5. ObjectStatus** | |
| **Physical data type:** | VARCHAR(1) |
| **Allow NULLs:** | Not Allowed |
| **Notes:** | Status of object: B/V (Blocked/Valid) |
| **6. AXWID** (FK) | |
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the Work (the intellectual work) requested by Object Creator |
| **7. AXCID** (FK) | |
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the Object Creator |

**8. FingerprintInfo**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Fingerprint information related to the object |

**9. ProtectionInfo**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Protection information related to the object |

**10. RegDate**

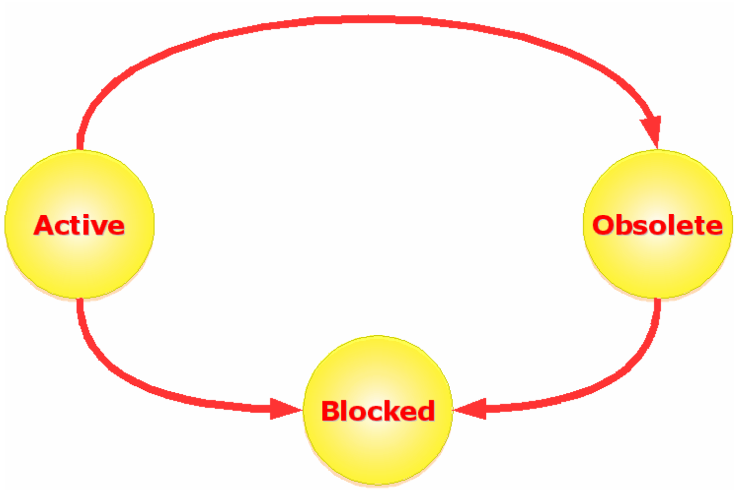
| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of the object registration (insertion in the AXMEDIS system), referred to GMT+0 |

**11. RegDeadline**

| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Allowed |
| **Notes:** | Timestamp of the object lifecycle end (if any), referred to GMT+0 |

Here it has to be specified how the ObjectStatus works and how it can be used in conjunction with the ObjectNewAXOID field. When an object is submitted for the distribution in the AXMEDIS system the object is “Active”. This means that the object is ready to be used by all the users who got the right to use it. An object can be set to “Blocked” when the system manager or the object owner discovery some problems about it and decide to block the object usage. This block can be temporary or definitive. . An object can also become obsolete if a new version of that specific object is issued. It has to be underlined that new version of an object are shipped with a new AXOID and are considered as a different object. In this case the object become “obsolete”, the ObjectNewAXOID field related to the old object is filled with the AXOID of the new object released. When an object become “obsolete” can be blocked or not, depending on the will of object owner and user license. An user can automatically have the rights to use the new version of the object or must acquire a new right, depending on the license he owns. The system has to flag to the user the availability of a new version of the object. In term of database field values, the status of the object can be summarized by the following table where is showed the status of the object depending on the database field values.

| | **Active** | **Obsolete Usable** | **Obsolete Unusable** | **Blocked** |
|---|---|---|---|---|
| **ObjectStatus** | U | U | B | B |
| **ObjectNewAXOID** | NULL | any value not null | any value not null | any value |



ObjectStatus state diagram

## GeneratedIDs

| **Columns** | | **idx** | **Data type** | **Allow NULLs** | **Value/Range** |
|---|---|---|---|---|---|
| **AXOID** | **PK** | **I** | VARCHAR(58) | Not allowed | |

| Column details | |
|---|---|
| **1. AXOID** (PK) | |
| **Physical data type:** | VARCHAR(40) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the Object |

## ComposedBy

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **CompID** | **PK** | **I** | BIGINT(20) | Not allowed | unsigned |
| **MainObjAXOID** | **FK** | **I** | VARCHAR(58) | Not allowed | |
| **IncludedObjAXOID** | **FK** | **I** | VARCHAR(58) | Not allowed | |

| Foreign keys | Child | Parent |
|---|---|---|
| **MainObjAXOID** | MainObjAXOID | Objects.AXOID |
| **IncludedObjAXOID** | IncludedObjAXOID | Objects.AXOID |

| Column details | |
|---|---|
| **1. CompID** (PK) | |
| **Physical data type:** | BIGINT(20) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Table primary key. Unsigned. |
| | |
| **2. MainObjAXOID** (FK) | |
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Main Object ID (ID of the complex Object) |
| | |
| **3. IncludedObjAXOID** (FK) | |
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the Object included in the related Main Object ID (MainAXOID) |

## DistributedBy

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **DistID** | **PK** | **I** | BIGINT(20) | Not allowed | unsigned |
| **AXOID** | **FK** | **I** | VARCHAR(58) | Not allowed | |
| **AXDID** | **FK** | **I** | VARCHAR(58) | Not allowed | |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXOID** | AXOID | Objects.AXOID |

| Column details | |
|---|---|
| **1. DistID** (PK) | |
| **Physical data type:** | BIGINT(20) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the row. It can be an auto-incremental ID and is used only by DB Manager and API DB. Unsigned |
| | |
| **2. AXOID** (FK) | |
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the Object |
| | |
| **3. AXDID** | |
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the Distributor. |

## DublinCore

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **DubliCoreID** | **PK** | **I** | BIGINT(20) | Not allowed | unsigned |
| **AXOID** | **FK** | **I** | VARCHAR(58) | Not allowed | |
| **Contributor** | | | LONGTEXT | Allowed | |
| **Coverage** | | | LONGTEXT | Allowed | |
| **Date** | | | DATETIME | Allowed | |
| **Description** | | | LONGTEXT | Allowed | |
| **Format** | | | LONGTEXT | Allowed | |
| **Identifier** | | | LONGTEXT | Allowed | |
| **Language** | | | VARCHAR(3) | Allowed | See ISO 639-2 standard |
| **Publisher** | | | LONGTEXT | Allowed | |
| **Relation** | | | LONGTEXT | Allowed | |
| **Rights** | | | LONGTEXT | Allowed | |
| **Source** | | | LONGTEXT | Allowed | |
| **Subject** | | | LONGTEXT | Allowed | |
| **Title** | | | LONGTEXT | Allowed | |
| **Type** | | | LONGTEXT | Allowed | |
| **XMLRefFile** | | | LONGTEXT | Allowed | |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXOID** | AXOID | Objects.AXOID |
| **DublinCoreID** | DCCreatorsMetadata.DublinCoreID | DublincoreID |

**Column details**

**1. DublinCoreID**

| | |
|---|---|
| **Physical data type:** | BIGINT(20) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Dublin Core metadata ID. Table primary-key.Unsigned. |

**2. AXOID** (FK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the related Object |

**3. Contributor**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field |

**4. Coverage**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field |

**5. Date**

| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field, referred to GMT+0 |

**6. Description**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field |

**7. Format**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field |

**8. Identifier**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field |

| | |
|---|---|
| **9. Language** | |
| **Physical data type:** | VARCHAR(3) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field. Its value is a language code according to ISO 639-2 standard. |
| **10. Publisher** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field |
| **11. Relation** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field |
| **12. Rights** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field |
| **13. Source** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field |
| **14. Subject** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field |
| **15. Title** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field |
| **16. Type** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Dublin Core metadata field |
| **17. XMLRefFile** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Link to an XML File containing Dublin Core Metadata |

## DCCreatorsMetadata

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **CreatorMetadataID** | **PK** | **I** | BIGINT(20) | Not allowed | Unsigned |
| **DublinCoreID** | **FK** | **I** | BIGINT(20) | Not allowed | unsinged |
| **CreatorValue** | | | LONGTEXT | Not allowed | |

| Foreign keys | Child | Parent |
|---|---|---|
| **DublinCoreID** | DublinCoreID | DublinCore.DublinCoreID |

| Column details | |
|---|---|
| **1. CreatorMetadataID** | |
| **Physical data type:** | BIGINT(20) |
| **Allow NULLs:** | Not Allowed |
| **Notes:** | CreatorMetadata ID. Table primary-key. Unsigned. |
| **2. DublinCoreID** | |
| **Physical data type:** | BIGINT(20) |

| | |
|---|---|
| **Allow NULLs:** | Not allowed |
| **Notes:** | Related Dublin Core metadata ID. Unsigned. |

**3 CreatorValue**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Creator metadata field value |

## ExtendedMetadata

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **MetadataID** | **PK** | **I** | BIGINT(20) | Not allowed | unsigned |
| **AXOID** | **FK** | **I** | VARCHAR(58) | Not allowed | |
| **MetadataFieldName** | | | LONGTEXT | Not allowed | |
| **MetadataFieldValue** | | | LONGTEXT | Not allowed | |
| **MetadataLang** | | | VARCHAR(3) | Not allowed | See ISO 639-2 standard |

| Foreign keys | Child | Parent |
|---|---|---|
| **AXOID** | AXOID | Objects.AXOID |

**Column details**

**1. MetadataID** (FK)

| | |
|---|---|
| **Physical data type:** | BIGINT(20) |
| **Allow NULLs:** | Not Allowed |
| **Notes:** | Metadata ID. Table primary-key. Unsigned |

**2. AXOID** (FK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the related Object |

**3. MetadataFieldName**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Metadata field name |

**4. MetadataFieldValue**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Metadata field value |

**5. MetadataLang**

| | |
|---|---|
| **Physical data type:** | VARCHAR(3) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Metadata field language code according to ISO 639-2 standard. |

# 19 Table description for AXCS Accounting database (DSI, FUPF)

## 19.1 Entity-Relationship description

The Accounting database is structured to take care of Action-Logs and once it will be standardized will have to implement all the necessary parts of MPEG21 Event Reporting. In order to define the AXCS Accounting Database schema, first we have to identify the entities and related relations. Here is reported the list of the identified entities and the related meaning.

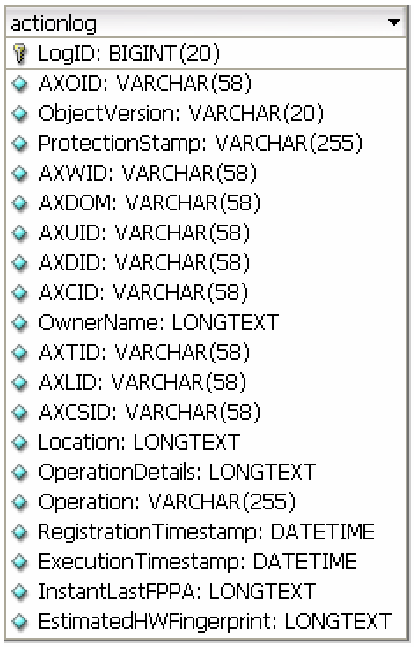
**ActionLog**: this entity stores the Action-Log as it is with some fixed information such as the AXOID on which the operation is performed, the AXUID of the user that performed the operation, the registration timestamp and execution timestamp (that can be different because of off-line operations performed on the objects). This entity is linked to the **OperationDetails** entity described below.

**SupervisorInputData**: The SupervisorInputData entity contains the same elements of an ActionLog entity plus a new element called AdditionalData. The role of this field is the storing of internal actions or actions between modules. It is used to check that some internal operations, like license requests, key requests, etc. have been done. A SupervisorInputData is generated in some situations when an Action Log is not, and that enable to track different issues, as for example:

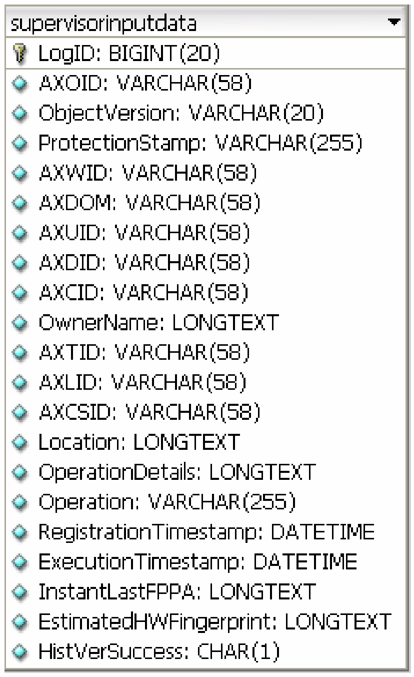
- AXCV stores a SupervisorInputData when a user is blocked during certification or verification to explain the reasons why. In this way, it is possible to track why the user was blocked.
- PMS Server stores a SupervisorInputData:
    - 1) when an AXMEDIS user requests a license to consume an AXMEDIS object (on-line or off-line), which enables to track license requests
    - 2) when an end user requests permission to perform an action on an AXMEDIS Object and PMS Server does not positively authorize the user, which enables to track attempts of incorrect usage of contents

Both **ActionLog** and **SupervisorInputData** report details about operations performed by users. These operations are based on the RDD terms of MPEG-21 standard, which have the concept of operations associated to an Event Report (Action Log for us). We have based our proposal of structure on this. We consider only operations made over an AXMEDIS Object. These operations generate an Action Log.
An example of these operations is:

- Modify : To edit an object in order to change it, or to protect it adding rules or metadata.
- Aggregate : To obtain an AXMEDIS Object as a composition of AXMEDIS Objects
- Render : To use or view an Object
- Play : Render as Performance
- Print : Render as Fixation
- Originate : To create a new AXMEDIS Object
- Enlarge : To Add something to an AXMEDIS Object already created
- Reduce : To modify an AXMEDIS Object by taking away from it
- Diminish : To create a new AXMEDIS Object from another one. The Object created is smaller than the source.
- Adapt : To Copy. To edit an AXMEDIS Object creating a new one which has the changes.
- Embed : To put an element or AXMEDIS Object into another AXMEDIS Object
- Delete : To destroy an AXMEDIS Object
- Identify : To nominate an AXMEDIS Object uniquely



actionlog
- LogID: BIGINT(20)
- AXOID: VARCHAR(58)
- ObjectVersion: VARCHAR(20)
- ProtectionStamp: VARCHAR(255)
- AXWID: VARCHAR(58)
- AXDOM: VARCHAR(58)
- AXUID: VARCHAR(58)
- AXDID: VARCHAR(58)
- AXCID: VARCHAR(58)
- OwnerName: LONGTEXT
- AXTID: VARCHAR(58)
- AXLID: VARCHAR(58)
- AXCSID: VARCHAR(58)
- Location: LONGTEXT
- OperationDetails: LONGTEXT
- Operation: VARCHAR(255)
- RegistrationTimestamp: DATETIME
- ExecutionTimestamp: DATETIME
- InstantLastFPPA: LONGTEXT
- EstimatedHWFingerprint: LONGTEXT



supervisorinputdata
- LogID: BIGINT(20)
- AXOID: VARCHAR(58)
- ObjectVersion: VARCHAR(20)
- ProtectionStamp: VARCHAR(255)
- AXWID: VARCHAR(58)
- AXDOM: VARCHAR(58)
- AXUID: VARCHAR(58)
- AXDID: VARCHAR(58)
- AXCID: VARCHAR(58)
- OwnerName: LONGTEXT
- AXTID: VARCHAR(58)
- AXLID: VARCHAR(58)
- AXCSID: VARCHAR(58)
- Location: LONGTEXT
- OperationDetails: LONGTEXT
- Operation: VARCHAR(255)
- RegistrationTimestamp: DATETIME
- ExecutionTimestamp: DATETIME
- InstantLastFPPA: LONGTEXT
- EstimatedHWFingerprint: LONGTEXT
- HistVerSuccess: CHAR(1)

Please note that in Action Logs and Supervisor Input Data the axmedis identifiers assigned to users are called with various names to distinguish the role played by an user in this case. These names are: AXCID (AXmedis Creator ID), AXDID (AXmedis Distributors ID), AXCSID (AXmedis Collecting Society ID).

## 19.2 Relational database schema extended description

Here is reported the list of identified tables came from entities and relations previously stated.

### ActionLog

| Columns | | Idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **LogID** | **PK** | **I** | BIGINT(20) | Not allowed | unsigned |
| **AXOID** | | **I** | VARCHAR(58) | Not allowed | |
| **ObjectVersion** | | **I** | VARCHAR(20) | Not allowed | |
| **ProtectionStamp** | | **I** | VARCHAR(255) | Not allowed | |
| **AXWID** | | **I** | VARCHAR(58) | Allowed | |
| **AXDOM** | | **I** | VARCHAR(58) | Allowed | |
| **AXUID** | | **I** | VARCHAR(58) | Not allowed | |
| **AXDID** | | **I** | VARCHAR(58) | Not allowed | |
| **AXCID** | | **I** | VARCHAR(58) | Not allowed | |
| **OwnerName** | | | LONGTEXT | Allowed | |
| **AXTID** | | **I** | VARCHAR(58) | Not allowed | |
| **AXLID** | | **I** | VARCHAR(58) | Not allowed | |
| **AXCSID** | | **I** | VARCHAR(58) | NotAllowed | |
| **Location** | | | LONGTEXT | Allowed | |
| **OperationDetails** | | **I** | LONGTEXT | Allowed | |
| **Operation** | | **I** | VARCHAR(255) | Not allowed | |
| **RegistrationTimestamp** | | | DATETIME | Not allowed | |
| **ExecutionTimestamp** | | **I** | DATETIME | Not allowed | |
| **InstantLastFPPA** | | | LONGTEXT | Not allowed | |
| **EstimatedHwFingerprint** | | | LONGTEXT | Not allowed | |

| **Column details** | |
|---|---|
| **1. LogID** | |
| **Physical data type:** | BIGINT(20) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Current registration ID in Action-Log Registry. Unsigned. |

**2. AXOID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent Object ID. Foreign Key of an AXMEDIS Object Table. |

**3. ObjectVersion**

| | |
|---|---|
| **Physical data type:** | VARCHAR(20) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent Object version. Foreign Key of an AXMEDIS Object Table. |

**4. ProtectionStamp**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Indicates the way to protect the related object. |

**5. AXWID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Pertinent Work Identification |

**6. AXDOM**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Pertinent User AXMEDIS Current Domain (if any) |

**7. AXUID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent User ID. The Event Report was prompted by a user with a certain AXUID (User ID). |

**8. AXDID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent Object Distributor ID |

**9. AXCID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent Object Creator ID |

**10. OwnerName**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Pertinent Object Owner |

**11. AXTID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the certified tool (the single instance of the installed tool). The Event Report was prompted from a peer (a tool) with a certain ID (AXTID). |

**12. AXLID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent Licence ID |

13. **AXCSID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent Collecting Society |

**14. Location**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Nation related to the pertinent collecting society. |

| | |
|---|---|
| **15. OperationDetails** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Details about the operation performed |
| **16. Operation** | |
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Operation performed |
| **17. RegistrationTimestamp** | |
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | timestamp of the registration in the AXCS, referred to GMT+0 |
| **18. ExecutionTimestamp** | |
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | timestamp of operation execution, referred to GMT+0 |
| **19. InstantLastFPPA** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | User related LastFPPA: Last Fingerprint of Performed Actions |
| **20. EstimatedHWFingerprint** | |
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Estimated HW fingerprint of the related terminal (PC or anything else) the software is running on at the specific action execution time |

## SupervisorInputData

| Columns | | Idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **LogID** | **PK** | **I** | BIGINT(20) | Not allowed | unsigned |
| **AXOID** | | **I** | VARCHAR(58) | Not allowed | |
| **ObjectVersion** | | **I** | VARCHAR(20) | Not allowed | |
| **ProtectionStamp** | | **I** | VARCHAR(255) | Not allowed | |
| **AXWID** | | **I** | VARCHAR(58) | Allowed | |
| **AXDOM** | | **I** | VARCHAR(58) | Allowed | |
| **AXUID** | | **I** | VARCHAR(58) | Not allowed | |
| **AXDID** | | **I** | VARCHAR(58) | Not allowed | |
| **AXCID** | | **I** | VARCHAR(58) | Not allowed | |
| **OwnerName** | | | LONGTEXT | Allowed | |
| **AXTID** | | **I** | VARCHAR(58) | Not allowed | |
| **AXLID** | | **I** | VARCHAR(58) | Not allowed | |
| **AXCSID** | | **I** | VARCHAR(58) | Not allowed | |
| **Location** | | | LONGTEXT | Allowed | |
| **OperationDetails** | | **I** | LONGTEXT | Allowed | |
| **Operation** | | **I** | VARCHAR(255) | Not allowed | |
| **RegistrationTimestamp** | | | DATETIME | Not allowed | |
| **ExecutionTimestamp** | | **I** | DATETIME | Not allowed | |
| **InstantLastFPPA** | | | LONGTEXT | Not allowed | |
| **EstimatedHwFingerprint** | | | LONGTEXT | Not allowed | |
| **AdditionalData** | | | LONGTEXT | NotAllowed | |

**Column details**

| | |
|---|---|
| **1. LogID** | |
| **Physical data type:** | BIGINT(20) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Current registration ID in Action-Log Registry. Unsigned. |

**2. AXOID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent Object ID. Foreign Key of an AXMEDIS Object Table. |

**3. ObjectVersion**

| | |
|---|---|
| **Physical data type:** | VARCHAR(20) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent Object version. Foreign Key of an AXMEDIS Object Table. |

**4. ProtectionStamp**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Indicates the way to protect the related object. |

**5. AXWID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Pertinent Work Identification |

**6. AXDOM**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Allowed |
| **Notes:** | Pertinent User AXMEDIS Current Domain (if any) |

**7. AXUID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent User ID. The Event Report was prompted by a user with a certain AXUID (User ID). |

**8. AXDID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent Object Distributor ID |

**9. AXCID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent Object Creator ID |

**10. OwnerName**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Pertinent Object Owner |

**11. AXTID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the certified tool (the single instance of the installed tool). The Event Report was prompted from a peer (a tool) with a certain ID (AXTID). |

**12. AXLID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent Licence ID |

**13. AXCSID**

| | |
|---|---|
| **Physical data type:** | VARCHAR(58) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Pertinent Collecting Society |

**14. Location**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Nation related to the pertinent collecting society. |

**15. OperationDetails**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Allowed |
| **Notes:** | Details about the operation performed |

**16. Operation**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Operation performed |

**17. RegistrationTimestamp**

| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | timestamp of the registration in the AXCS, referred to GMT+0 |

**18. ExecutionTimestamp**

| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | timestamp of operation execution, referred to GMT+0 |

**19. InstantLastFPPA**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | User related LastFPPA: Last Fingerprint of Performed Actions |

**20. EstimatedHWFingerprint**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Estimated HW fingerprint of the related terminal (PC or anything else) the software is running on at the specific action execution time |

**21. AdditionalData**

| | |
|---|---|
| **Physical data type:** | LONGTEXT |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Additional Data added to control and check that any action is being done correctly. |

# 20 Table description for User Registration database for user registration portal (DSI)

## 20.1 Entity-Relationship description

The User Registration database for User Registration Portal has to support the portal in order to play the role of an entity entitled to register users in AXMEDIS. Therefore, it has to retain user personal information and the associations between these data and the AXUIDs assigned by the system. In order to define the database schema, first we have to identify the entities and related relations. Here is reported the list of the identified entities and the related meaning.

**registrations**: this entity stores data about the registration, both data about the process itself, such as the status of the process (i.e. “confirm”, “confirmed” or “accepted”), and the association between the user data and the assigned AXUID

**users**: this entity represents the subject of the registration process, i.e. the user. Therefore it stores all the user personal data such as name, address, city, state, telephone number, etc.

## 20.2 Relational database schema extended description

Here is reported the list of identified tables came from entities and relations previously stated.

### registrations

| Columns | | Idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **registration_id** | **PK** | **I** | BIGINT(20) | Not allowed | Unsigned |
| **user** | | | BIGINT(20) | Not allowed | Unsigned |
| **phase** | | | VARCHAR(255) | Not allowed | |
| **AXUID** | | | VARCHAR(255) | Not allowed | |

| Column details | |
|---|---|
| **1. registratio_id** | |
| **Physical data type:** | BIGINT(20) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Current registration ID in the table. Unsigned |
| **2. user** | |
| **Physical data type:** | BIGINT(20) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID identifying the record in the **utenti** table storing the personal data of the user to whom the current records refers to. Unsigned |
| **3. phase** | |
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Current phase of the registration process |
| **4. AXUID** | |
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | AXMEDIS Uinique ID assigned to the user at the end of the registration proceess |

### users

| Columns | | Idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **user_id** | **PK** | **I** | BIGINT(20) | Not allowed | unsigned |
| **nickname** | | | VARCHAR(255) | Not allowed | |
| **password** | | | VARCHAR(255) | Not allowed | |
| **code** | | | VARCHAR(255) | Not allowed | |

| | | | | | |
|---|---|---|---|---|---|
| **name** | | | VARCHAR(255) | Not allowed | |
| **surname** | | | VARCHAR(255) | Not allowed | |
| **address** | | | VARCHAR(255) | Not allowed | |
| **city** | | | VARCHAR(255) | Not allowed | |
| **cap** | | | VARCHAR(255) | Not allowed | |
| **state** | | | VARCHAR(255) | Not allowed | |
| **telephone** | | | VARCHAR(255) | Not allowed | |
| **fax** | | | VARCHAR(255) | Allowed | |
| **cellular** | | | VARCHAR(255) | Allowed | |
| **email** | | | VARCHAR(255) | Not allowed | |
| **title** | | | VARCAHR(255) | Allowed | |
| **birth_date** | | | DATE | Not allowed | |
| **registration_date** | | | DATE | Not allowed | |

**Column details**

**1. user_id**

| | |
|---|---|
| **Physical data type:** | BIGINT(20) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Current user ID in the table. Unsigned |

**2. nickname**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | NickName of  the user |

**3. password**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | MD5 or other encryption of user password |

**4. code**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Temporary ID assigned to the user while his/her registration in AXMEDIS is pending |

**5. name**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | User name |

**6. surname**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | User family name |

**7. address**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | User address |

**8. city**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | City where the user lives |

**9. cap**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ZIP code of the sity where the user lives |

**10. state**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | State where the user lives |

**11. telephone**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | User telephone number |

**12. fax**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Allowed |
| **Notes:** | User fax number |

**13. cellular**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Allowed |
| **Notes:** | User cellphone number |

**14. email**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | User email address |

**15. birth_date**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | User birth date |

**16. title**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Allowed |
| **Notes:** | User qualification |

**17. registration_date**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | User registration date, referred to GMT+0 |

# 21 Table description for Active AXCSs List Database (DSI)

## 21.1 Entity-Relationship description

The Active AXCSs List Database has been conceived to contain data about all the AXCSs spread over the world. Its location depends on the AXCSs network architecture adopted, as follows:

1. **Fully peer network architecture**. In this case the Active AXCSs List Database has to be replicated on every AXCS in order to let each AXCS to know about the other peers. Information has to be propagated over the network.
2. **Two (or more) level peer network**. In this case the peer network is composed by peers located in the highest level; only these AXCSs have to host the Active AXCSs List Database. Information has to be propagated and replicated over the network, among these high level AXCSs.
3. **Network with an AXCS master (SuperAXCS)**. In this case the SuperAXCS plays the role of coordinator and has some major functions with respect to other AXCSs. The ActiveAXCSs List Database can be located only in SuperAXCS.

Here is reported the list of the identified entities and the related meaning.

**axcslist**: this entity stores data about the AXCSs in the AXMEDIS system. An AXCS can play its onw role and is active only after the registration phase that consists in recording AXCS data in the axcslist table.

## 21.2 Relational database schema extended description

Here is reported the list of identified tables came from entities and relations previously stated.

### AXCSLIST

| Columns | | idx | Data type | Allow NULLs | Value/Range |
|---|---|---|---|---|---|
| **AXCERSID** | **PK** | **I** | VARCHAR(40) | Not allowed | |
| **AXDOM** | **FK** | **I** | VARCHAR(40) | Allowed | |
| **Email** | | | VARCHAR(255) | Not allowed | |
| **NickName** | | | VARCHAR(255) | Not allowed | |
| **Password** | | | VARCHAR(255) | Not allowed | |
| **Description** | | | VARCHAR(255) | Not allowed | |
| **Location** | | | VARCHAR(255) | Not allowed | |
| **Nationality** | | **I** | VARCHAR(255) | Not allowed | |
| **PubKey** | | | LONGTEXT | Not allowed | |
| **RegDate** | | | DATETIME | Not allowed | |
| **RegDeadline** | | | DATETIME | Not allowed | |
| **Status** | | | VARCHAR(1) | Not allowed | B/U (Blocked/Unblocked) |

**Column details**

**1. AXCERSID** (PK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(40) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | ID of the AXCS |

**2. AXDOM** (FK)

| | |
|---|---|
| **Physical data type:** | VARCHAR(40) |
| **Allow NULLs:** | Allowed |
| **Notes:** | AXMEDIS Current Domain of the AXCS (if any) |

**3. Email**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | email of the related AXCS manager |

**4. NickName**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | NickName of the AXCS manager |

**5. Password**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | MD5 or other encription of AXCS manager password |

**6. Description**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Description of the AXCS |

**7. Location**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Location of the AXCS |

**8. Nationality**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Nationality of the AXCS |

**9. PubKey**

| | |
|---|---|
| **Physical data type:** | VARCHAR(255) |
| **Allow NULLs:** | Not allowed |
| **Notes:** | public key of the AXCS |

**10. RegDate**

| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of AXCS registration |

**11. RegDeadline**

| | |
|---|---|
| **Physical data type:** | DATETIME |
| **Allow NULLs:** | Not allowed |
| **Notes:** | Timestamp of AXCS registration end |

**12. Status**

| | |
|---|---|
| **Physical data type:** | VARCHAR(1) |
| **Allow NULLs:** | Not Allowed |
| **Notes:** | Status of the AXCS: B/U (Blocked/Unblocked) |

# 22 Formal description of AXMEIDS prefixes format (DSI)

AXMEDIS prefixes are used to charcaterise identifiers, in order to let tools undertand the meaning of an identifier simply parsing the prefix.
Each AXMEDIS prefix is a simple string consisting of an urn containing (among other parts) a three characters code, chosen appropriately among the following:

| **Identifier** | **AXMEDIS Prefix** | |
|---|---|---|
| AXDOM | DOM | |
| AXLID | LIC | |
| AXOID | OBJ | |
| AXRTID | RTO | |
| AXTID | ITO | |
| AXTOID | TOB | |
| AXTTID | TOT | |
| AXUID | Prefix for Final Users | USR |
| | Prefix for B2B Users | BUS |
| AXWID | WRK | |

For the meaning of each identifier please see section 23.

# 23 Formal description of AXMEDIS ID format (DSI)

An AXMEDIS identifier is an URN (see http://www.ietf.org/rfc/rfc2141.txt for further details on URNs syntax) with the following syntax:

<AXMEDIS_ID> ::= "urn:" <AXMEDIS_NID> ":" <AXCS_ID> ":" <AXMEDIS_PREFIX> ":" <UUID>

where:

<AXMEDIS_NID> ::= "axmedis"
<AXCS_ID> ::= <hexDigit><hexDigit><hexDigit><hexDigit><hexDigit>
<AXMEDIS_PREFIX> ::= "DOM" | "LIC" | "OBJ" | "RTO" | "ITO" | "TOB" | "TOT"

Please note that the identifier "00000" will never be assigned to any AXCS, so temporary IDs can be used in the same format of AXMEDIS ID but only with the "00000" AXCS ID.

A UUID is an identifier that is unique across both space and time, with respect to the space of all UUIDs. The formal definition of the UUID string representation is provided by the following extended BNF (please see http://www.opengroup.org/onlinepubs/9629399/apdxa.htm for more details):

| | |
|---|---|
| UUID | = <time_low> <hyphen> <time_mid> <hyphen> <time_high_and_version> <hyphen> <clock_seq_and_reserved> <clock_seq_low> <hyphen> <node> |
| time_low | = <hexOctet> <hexOctet> <hexOctet> <hexOctet> |
| time_mid | = <hexOctet> <hexOctet> |
| time_high_and_version | = <hexOctet> <hexOctet> |
| clock_seq_and_reserved | = <hexOctet> |
| clock_seq_low | = <hexOctet> |
| node | = <hexOctet><hexOctet><hexOctet> <hexOctet><hexOctet><hexOctet> |

where:

| | |
|---|---|
| hexOctet | = <hexDigit> <hexDigit> |
| hexDigit | = <digit> \| <a> \| <b> \| <c> \| <d> \| <e> \| <f> |
| digit | = "0" \| "1" \| "2" \| "3" \| "4" \| "5" \| "6" \| "7" \| "8" \| "9" |
| hyphen | = "-" |
| a | = "a" \| "A" |
| b | = "b" \| "B" |
| c | = "c" \| "C" |
| d | = "d" \| "D" |
| e | = "e" \| "E" |
| f | = "f" \| "F" |

So, an UUID consists of a string of 36 characters split in 5 groups as follows:

xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

where "x" is an hexadecimal character. For instance:

06ebd820-91a2-11da-a1bf-0002a5d5c51b.

Therefore, an AXMEDIS identifier is in the form:

urn:axmedis:yyyy:PPP:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

where “y” and “x” are hexadecimal characters, “y” has always a value different from “0” and “PPP” is a three character code correspondent to an AXMEDIS prefix (see section 22).
Depending on the prefix used an AXMEDIS identifier is called in different ways:

| **Acronym** | **Description** | | **ID Prefix** |
|---|---|---|---|
| AXDOM | AXMEDIS Domain (associated with the PMS Home or Factory managing the Domain) | | DOM |
| AXLID | Unique AXMEDIS License ID | | LIC |
| AXOID | Unique AXMEDIS Object ID | | OBJ |
| AXRTID | AXMEDIS Registered Tool ID. It is the identifier of the class of tool, eg. the editor of producer X | | RTO |
| AXTID | AXMEDIS Tool ID, a specific instance of an AXRTID, e.g. the instance of the editor of producer X installed on the machine of the user A | | ITO |
| AXTOID | AXMEDIS Temporary Object ID, produced directly from the numbers available for the AXMEDIS Creator. It can be produced as AXCID+AXTID+AXUID+an progressive ID of the Tool | | TOB |
| AXTTID | AXMEDIS Tool Type ID Different tools are Editor, Composition Engine, Formatting Engine, Protection Tool Engine, etc. This is contained into the Profile, several other aspects and details for each tool have to be registered. Some other details depends on the specific instance, this the reason to provide and exploit a Tool Profile. | | TOT |
| AXUID | Unique AXMEDIS User ID | Prefix for Final Users | USR |
| | | Prefix for B2B Users | BUS |
| AXWID | Work Identification. For each Work you may have several different versions of AXOB, for example differing for resolution, market, format of the digital resources, etc. | | WRK |

## 24 AXMEDIS Action Log Format (DSI, FUPF)

Action Logs are used to trace every action performed (being previously authorized by the system) on objects at both business and final user level. Section 19 describes the format of an Action Log in terms of contained data. These data are provided by different AXMEDIS modules during the normal system functioning.
Here is a table describing which component is in charge of filling the pertinent data:

| FILLED BY / ACTIONLOG FIELD | AXOM | PMS Server | AXCS |
|---|---|---|---|
| **AXCID** | X | | |
| **AXCSID** | | X | |
| **AXDID** | X | | |
| **AXDOM** | | X | |
| **AXLID** | | X | |
| **AXOID** | X | | |
| **AXTID** | X | | |
| **AXUID** | X | | |
| **AXWID** | X | | |
| **EstimatedHWFingerprint** | X | | |
| **ExecutionTimestamp** | X | | |
| **historyVerificationSuccess** | | | X |
| **location** | | X | |
| **log id** | | | X |
| **objectVersion** | X | | |
| **operation detail** | X | | |
| **operation** | X | | |
| **ownername** | X | | |
| **protectionTimeStamp** | X | | |
| **registrationTimeStamp** | | | X |

As it is shown from this table, three components are in charge of filling the needed data for an Action Log: the AXOM, the PMS Server and the AXCS. Of course the AXCS stores most of the needed data, but they are not exploited because Action Logs are used to trace what happens and who is performing the action. Therefore, a different source for these data is needed, particularly a source on the "client" side from the system (i.e. AXCS and PMS Server) point of view. So, most of these data are filled by AXOM, which resides inside the tool performing the operation.

Most of these data are recovered directly by the AXOM on the basis of information included in the AXMEDIS object (*AXCID*, *AXDID*, *AXOID*, *AXWID*, *objectVersion*, *ownername*, *protectionTimeStamp*) or on the basis of the identities of the user and of the tool (*AXUID*, *AXTID*, *EstimatedHWFingerprint*, *ExecutionTimeStamp*, *operationdetail*, *operation*)

*AXLID*, *AXDOM*, *location* and *AXCSID* will be included in the license related to the AXOID contained in the Action Log. Therefore, the PMS Server will be able to retrieve them and put them into the corresponding Action Log.

There is an Action Log History Digest, called *instantLastFPPA*, for each certified tool (that is, the instantLastFPPA may involve more than one user).

AXCS from one side and PMS Client from the other side are responsible to calculate a new instantLastFPPA: it is calculated in the client side and recalculated in the server side to be verified. The instantLastFPPA in the client side is derived from the previous instantLastFPPA stored in the local cache and the data in the "current" ActionLog, then it is updated in the local cache. AXCS-AXCV verifies the instantLastFPPA consistency by recalculating it and, once verified, stores it in AXCSRegCert database.

instantLastFPPA will not be filled in the Action Log structure in the client side to avoid that a malicious user can find the hash evolution for each action in the same place where the hash is stored (in the local cache).
instantLastFPPA will not be placed in the Action Log in the server side because once verified it is not necessary to store it.
Therefore, this value is neither inserted in the Action Log nor stored in the AXCSAccounting database.

In order to calculate the instantLastFPPA, a subset of the fields contained in the Action Log is used. In particular, the following fields are involved: *AXOID*, *AXTID*, *AXUID*, *ExecutionTimestamp*, *objectVersion*, *operationID*, *protectionTimeStamp*, *estimatedHWFingerprint*, *AXLID* and *AXDOM*.
We have two different approaches to tackle the LastFPPA calculation:

1) To have a different LastFPPA for each user+tool.
2) To have a single LastFPPA common for all users in the same certified tool.

Advantages for approach 1):

- If AXCS detects that the lastFPPA is not consistent, it can block not only the tool but also the user whose history failed.

Disadvantages for approach 1):

- One solution consists on retrieving the LastFPPA for each user+tool from the corresponding last ActionLog for those user+tool in the Accounting database. This solution is dangerous as Action Logs can be removed from the Accounting database after a period (for instance, after billing is performed). Another solution would be to redesign AXCS database and PMS Client secure cache in order to allow the storage of LastFPPA for each user+tool (RegCert database)
- If a user altered a tool and the next user that performed an action on that tool was a different user, AXCS would determine that the history is not consistent (because AXCS would search the previous action log for that user (AXUID) and device (EstimatedHWFingerprint) and it would not get any, as device fingerprint would have changed) and an innocent user would be blocked.

Advantages for approach 2):

- AXCS RegCert database is designed for this case, so it can be kept as it is now. LastFPPA is retrieved for each tool from CerTools table in AXCS database. Secure cache is also designed for this case currently.

Disadvantages for approach 2):

- If AXCS detects that the lastFPPA is not consistent, it can block the tool but it does not know which user is responsible of the failure (e.g. if the check fails when verifying LastFPPA with an action log coming from user 1, it could be because user 2 had erased some action logs involved in the history). Thus, it can act in two ways:
  - a) block all users that have some action logs stored in the tool: some innocent users might be blocked, and perhaps not the malicious user if he has succeded to erase all his action logs.
  - b) do not block any user: the malicious user will not be blocked in AXMEDIS

  In both cases a) and b) the tool would be blocked

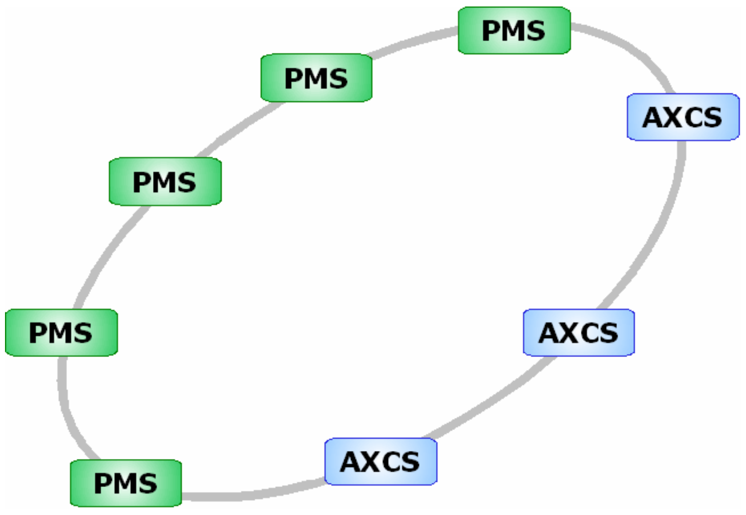
The second solution seems to be the most feasible and it is the one that will be adopted.

# 25 AXCS/PMS Data Diffusion Format (DSI)

As explined in section 13, AXMEDIS architecture is composed of various AXCSs and PMSs. Therefore, there is the need for a format to let omogeneus entities (i.e. AXCSs among themselves and PMSs among themselves) exchange data.

As for PMSs, they have to exchange licenses to replicate data in order to increase the system fault tolerance. So, they simply transfer data blocks spanning through out all the license database. Please note, this does not mean that a PMS has to transfer all its data to all others PMSs replicating data of the current one; the transfer can involve a subset of those data as regards the cardinality of the records. However, the transfer musts involve consistent data, thus data exchanged will span across all the tables of the license database, in order to correctly recreate relations among records in different tables.
Therefore, if, for instance, the license database of a PMS (let say A) is replicated on other two PMSs (let say B and C) and it contains 100000 records in each of its 5 tables (for sake of simplicity let suppose all the tables have the same number of records, for a total of 500000 records), then A could transfer 250000 records to B and the other 250000 to C, but to both B and C would be transferred 50000 records for each table related each other.

As for AXCSs, they have to exchange data about users, tools, action logs, objects, etc. This can happen for two reasons: to replicate data in order to increase the system fault tolerance or to migrate data in order to permit users to use object also among different distribution channel. It has to be reminded that an AXCS operate only on a single distribution channel.



Network of AXCSs and PMSs

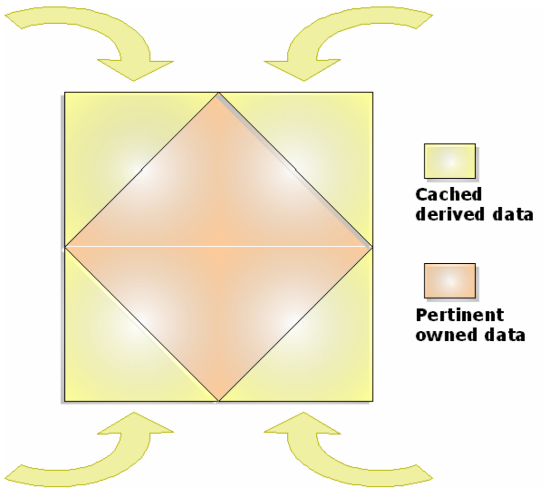
## 25.1 Different kind of data

Both AXCSs and PMSs retain an huge amount of data. As stated in the above paragraph not all data can be replicated over all the subjects in the network: only a part of information has to be migrated to other subjects. About data management, two different kind of data has to be depicted:

1. Pertinent owned data
2. Cached derived data

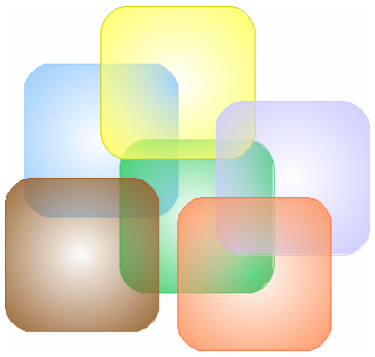
The former refers to data acquired and collected directly by the interested subject (AXCS or PMS). For example, this kind of information refers to data related to user registered through a specific AXCS or a license produced by a specific PMS; for that AXCS or that PMS these information is “Pertinent owned data”.

The latter refers to data received by an AXCS coming from other AXCSs through the network of AXCSs, or to data received by an PMS coming from other PMSs through the network of PMSs.



Pertinent owned data vs Cached derived data:
Cached derived data are Pertinent owned data coming from other network nodes

It has to be remarked that cached derived data (data transferred from an AXCS to other AXCSs or from a PMS to other PMSs through related networks), can't violate the consistency of the database: all data logically related has to be transferred. For example, if metadata related to an object has to be transferred (migrated) from an AXCS to another, also all related data such as creator, distributor, and so on has to be transferred.
In the figure below, each node of the network (AXCS or PMS) is represented as a square. Each square contains both Pertinent owned data and Cached derived data. Intersections among squares represent data shared among different nodes and are Pertinent owned data for a single square and Cached derived data for the others.



Data distribution among network nodes

The way to decide which portion of data has to be transferred among AXCSs or among PMSs through the related networks will be defined by a special kind of intelligent agents called "**Crossmovers**". In this way the knowledge retained by AXCSs and PMSs moves through the related network and reaches "adjacent" network node. The adjacency is not related with physical neighbourhood, but refers to logical relationship between nodes.

# 26 Formal description of AXS axs.properties file format (FUPF)

axs.properties is a Java properties file that contains the information necessary for the initialisation of the AXS module.

Refer to http://java.sun.com/j2se/1.4.2/docs/api/java/util/Properties.html#load(java.io.InputStream) for more information about the syntax.

Example:

```
#AXCS Accounting database DSN. This parameter will be only used if the
axcsdb.ini file from DSI database is not found in the system.
AccountingDSN=jdbc\:mysql\://193.145.45.173/axcsaccounting

#AXCS Objects ID database DSN. This parameter will be only used if the
axcsdb.ini file from DSI database is not found in the system.
ObjectsIdDSN=jdbc\:mysql\://193.145.45.173/axcsobjectsid

#Database User. This parameter will be only used if the axcsdb.ini file from DSI
database is not found in the system.
AxcsDbUser=axmedis

#Database Password. This parameter will be only used if the axcsdb.ini file from
DSI database is not found in the system.
AxcsDbPassword=axmedis
```

# 27 Formal description of AXCV nextSerial.txt file format (FUPF)

nextSerial.txt is a text file which contains the value of the next serial number to be used. The specified value will be interpreted as a java.lang.Long. As the serial number cannot be less than “0”its range will be the following: [0  to $2^{63}$].

Example of nextSerial.txt file contents:

```
1000000068
```

# 28 Formal description of AXCV AxcsCAPkcs12.p12 and axcvToolCertStore.p12 file format (FUPF)

AxcsCAPkcs12.p12 keystore contains the AXCS certificate and private key with which tool certificates are signed.

axcvToolCertStore.p12 keystore contains the generated tool certificates (tool private key is not stored)

These files are PKCS12 keystores follow the syntax specified in "PKCS #12 v1.0: Personal Information Exchange Syntax Standard", RSA Laboratories, June 24, 1999, which can be found at http://www.rsasecurity.com/

They will only work with the keytool if the password is provided on the command line, as the entire keystore is encrypted with a PBE based on SHA1 and Twofish (PBEWithSHAAndTwofish-CBC). This makes the entire keystore resistant to tampering and inspection, and forces verification. The Sun JDK provided keytool would attempt to load a keystore even if no password is given, which is impossible for this version of keystore.

# 29 Formal description of AXCV axcv.properties file format (FUPF)

axcv.properties is a Java properties file that contains the information necessary for the initialisation of the AXCV module. It can be customised to change the names and passwords of the corresponding files and key stores that the AXCV needs to access.

Refer to http://java.sun.com/j2se/1.4.2/docs/api/java/util/Properties.html#load(java.io.InputStream) for more information about the syntax.

Example:

```
#PKCS12 store where AXCV certificate and private key are stored
axcsCertStore=AxcsCAPkcs12.p12

#Password for PKCS12 store where AXCV certificate and private key are stored
axcsCertStorePasswd=AXCSpwd

#PKCS12 store where the generated tool certificates are stored (private key not
stored)
axcvToolCertStore=axcvToolCertStore.p12

#Password for PKCS12 store where the generated tool certificates are stored
axcvToolCertStorePasswd=AXCSpwd

#File name where serial number for next tool Certificate is stored
nextSerial=nextSerial.txt

#AXCS Registration and Certification database DSN. This parameter will be only
used if the axcsdb.ini file from DSI database is not found in the system.
RegCertDSN=jdbc\:mysql\://193.145.44.41/axcsregcert

#Database User. This parameter will be only used if the axcsdb.ini file from DSI
database is not found in the system.
AxcsDbUser=axmedis

#Database Password. This parameter will be only used if the axcsdb.ini file from
DSI database is not found in the system.
AxcsDbPassword=axmedis
```

# 30 Formal description of AXCV toolBase64PKCS12 output parameter format (FUPF)

PKCS12 stream bytes encoded in Base 64. The PKCS12 format used is the same as that explained for AxcsCAPkcs12.p12 and axcvToolCertStore.p12 files.

See RFC 1521, RFC 2045 and RFC 3548 for further information about Base64 encoding.

toolBase64PKCS12 includes the tool certificate signed by the AXCS CA Root Certificate and tool private key together and protected with a password.

The tool certificate will have the format explained in document AXMEDIS-DE3-1-2-2-3-Spec-of-AXOM-and-ProtProc-upB-v1-5.doc.

The certificate will include as an extension the certified tool activation code (or enabling code). The extension will be identified with the Object Identifier 1.3.6.1.4.1.25576.1.1, where 1.3.6.1.4.1.25576 is the Private Enterprise Number assigned by IANA to AXMEDIS Organisation, as described in next section.
If the Unrestricted policy files for Sun JCE were available at the server (default configuration), the password used 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.
The steps followed to obtain toolBase64PKCS12 from a PKCS12 object can be summarized as follows:

1. PKCS12 object
2. get PKCS12 byte stream
3. encode in Base64

The steps you can follow to obtain a the PKCS12 object from the toolBase64PKCS12 output parameter can be summarized as follows:

1. Decode toolBase64PKCS12 in Base 64

2. Get the PKCS12 KeyStore object from the bytes using the already mentioned password

# 31 Formal description of the distribution of the OIDs tree assigned by IANA to AXMEDIS

The IANA has assigned 1.3.6.1.4.1.7547 Private Enterprise Number to AXMEDIS Organisation, which can be found at http://www.iana.org/assignments/enterprise-numbers.

Upper references:

1.3.6.1.4.1 - IANA-registered Private Enterprises.
1.3.6.1.4 - Internet Private.
1.3.6.1 - OID assignments from 1.3.6.1 - Internet.
1.3.6 - US Department of Defense.
1.3 - ISO Identified Organization.
1 - ISO assigned OIDs.

The distribution of the AXMEDIS tree corresponding to the 1.3.6.1.4.1.7547 branch will be the following:

- 1.3.6.1.4.1.7547.0: reserved
- 1.3.6.1.4.1.7547.1: AXMEDIS PKI-X.509 related objects
  - 1.3.6.1.4.1.7547.1.1: AXMEDIS Tool certificate extensions
    - 1.3.6.1.4.1.7547.1.1.1: AXMEDIS Tool activation code (or enabling code)

# 32 Formal description of toolFingerprint input parameter format in AXCV certify method (FUPF)

toolFingerprint is an XML file serialized as a string, which corresponds to the SoftwareFingerprint part of ToolFingerprint XML Schema defined in the Protection Processor section of Framework and Tools Specifications document.

AXCV toolFingerprint parameter is used in certify method to ensure that the installed tool corresponds to the original one registered in AXCS database.

AXCV will extract the information in the Category, FullFileName, Signature, CreationDate and LastModificatioDate tags, if present, and compare it to the information stored in AXCS RegCert database. PhysicalPosition information will be omitted because it refers to the installation of the tool, and thus is installation dependent.

# 33 Formal description of toolFingerprint input parameter format in AXCV reverify method (FUPF)

toolFingerprint is an XML file serialized as a string, which corresponds to the full  ToolFingerprint XML Schema defined in the Protection Processor section of  Framework and Tools Specifications document.

# 34 Formal description of toolFingerprintDigest input parameter format in AXCV verify method (FUPF)

toolFingerprint is the SHA1 hash bytes encoded in Base64 corresponding to the relevant data (tags not included) in the XML file used to describe the AXMEDIS Tool Fingerprint, which corresponds to the full ToolFingerprint XML Schema mentioned in previous section.
The steps to obtain toolFingerprintDigest from toolFingerprint XML file can be summarized as follows:

1. Get XML file
2. Get relevant data (exclude XML tags)
3. Concatenate the different fields obtained in previous steps without spaces and create a string

4. Calculate the SHA1 Digest of the previous string
5. Encode in Base64

Refer to RFC 3714 for more information about the SHA1 Message-Digest Algorithm.

# 35 Formal description of regDeadline input parameter format in AXCV certify method (FUPF)

RegDeadline input parameter is a string which denotes the date when the tool will stop working and is expressed in the following format:

"yyyy-MM-ddThh:mm:ss".

Where,

yyyy: year
MM: month
dd: day
T is the character used as separator between the date and the time
hh: hour
mm: minute
ss: seconds

Example: "2020-01-31 23:59:59"

# 36 Formal description of communication protocol AXCSUserRegistration (DSI)

| AXCSUserRegistration web service | |
|---|---|
| Method | registration |
| Description | It is the only public method of the user registration web service. It collects Distributor credentials needful to access the system and uses the verifyLogin() (a DataManager method) to verify requestor credentials. It collects also registration data (regInfo) provided by the requesting distributor and uses the other methods to insert them in the database and to provide the result to the requesting distributor. |
| Input parameters | UserRegistrationInfo regInfo – a data structure containing the nickname and password of the requesting distributor and a nested data structure containing the data to be inserted/updated |
| Output parameters | RegistrationResult – The result output parameter is a data structure containing three fields:<br>• "resultStatus": it is set to 0 if the registration is successful otherwise is set to 1.<br>• "definitiveUID": this field returns the definitive AXUID in case of success, a *null* value otherwise<br>• "errorMessage": this filed returns the error message in case of failure, a *null* value otherwise |
| Request Sample Message | <?xml version="1.0" encoding="UTF-8"?><br><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><br><soapenv:Body><br><regInfo xsi:type="ns1:UserRegistrationInfo" xmlns="org:axmedis:axcs:services:userregistrator" xmlns:ns1="org:axmedis:axcs:services:userregistrator"><br><ns1:nickName xsi:type="xsd:string">mario</ns1:nickName><br><ns1:password xsi:type="xsd:string">xxxxx</ns1:password><br><ns1:regData xsi:type="ns1:UserDataType"><br><ns1:axdom xsi:type="xsd:string" xsi:nil="true"/><br><ns1:axuid xsi:type="xsd:string">urn:axmedis:00000:BUS:d0719d28-e695-4db7-841c-f078ae7fdfb6</ns1:axuid><br><ns1:cmpAddress xsi:type="xsd:string" xsi:nil="true"/><br><ns1:cmpFax xsi:type="xsd:string" xsi:nil="true"/><br><ns1:cmpPhone1 xsi:type="xsd:string" xsi:nil="true"/><br><ns1:cmpPhone2 xsi:type="xsd:string" xsi:nil="true"/><br><ns1:collectSocRegDeadline xsi:type="xsd:dateTime" xsi:nil="true"/><br><ns1:collectSocStatus xsi:type="xsd:string" xsi:nil="true"/><br><ns1:company xsi:type="xsd:string" xsi:nil="true"/><br><ns1:creatorRegDeadline xsi:type="xsd:dateTime" xsi:nil="true"/><br><ns1:creatorStatus xsi:type="xsd:string" xsi:nil="true"/><br><ns1:distributorRegDeadline xsi:type="xsd:dateTime" xsi:nil="true"/><br><ns1:distributorStatus xsi:type="xsd:string" xsi:nil="true"/><br><ns1:email xsi:type="xsd:string">myaddress@email.com</ns1:email><br><ns1:final xsi:type="xsd:boolean">true</ns1:final><br><ns1:finalRegDeadline xsi:type="xsd:dateTime">2006-12-31T23:59:59</ns1:finalRegDeadline><br><ns1:finalStatus xsi:type="xsd:string">U</ns1:finalStatus><br><ns1:location xsi:type="xsd:string" xsi:nil="true"/><br><ns1:nationDomain xsi:type="xsd:string" xsi:nil="true"/><br><ns1:nationality xsi:type="xsd:string">nationality</ns1:nationality><br><ns1:nickName xsi:type="xsd:string">nick</ns1:nickName><br><ns1:password xsi:type="xsd:string">passw</ns1:password><br><ns1:phone xsi:type="xsd:string" xsi:nil="true"/> |

```
          <ns1:pubKey xsi:type="xsd:string">pubkey</ns1:pubKey>
          <ns1:refName xsi:type="xsd:string" xsi:nil="true"/>
          <ns1:toolProdRegDeadline xsi:type="xsd:dateTime" xsi:nil="true"/>
          <ns1:toolProdStatus xsi:type="xsd:string" xsi:nil="true"/>
          <ns1:typeOfUser xsi:type="xsd:string" xsi:nil="true"/>
          <ns1:webSite xsi:type="xsd:string" xsi:nil="true"/>
        </ns1:regData>
        <ns1:replace xsi:type="xsd:boolean">false</ns1:replace>
      </regInfo>
    </soapenv:Body>
  </soapenv:Envelope>
```

Response Sample Message

```
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <registrationReturn xmlns="org:axmedis:axcs:services:userregistrator">
        <definitiveUID>urn:axmedis:34f5a:USR:812e6edc-7061-351f-b1ee-
209bd940b513</definitiveUID>
        <resultStatus>0</resultStatus>
        <errorMessage/>
      </registrationReturn>
    </soapenv:Body>
  </soapenv:Envelope>
```

AXCSUserRegistration web service WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="org:axmedis:axcs:services:userregistrator"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="org:axmedis:axcs:services:userregistrator"
xmlns:intf="org:axmedis:axcs:services:userregistrator" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="org:axmedis:axcs:services:userregistrator"
xmlns="http://www.w3.org/2001/XMLSchema">
   <complexType name="UserDataType">
    <sequence>
     <element name="axdom" nillable="true" type="xsd:string"/>
     <element name="axuid" type="xsd:string"/>
     <element name="cmpAddress" nillable="true" type="xsd:string"/>
     <element name="cmpFax" nillable="true" type="xsd:string"/>
     <element name="cmpPhone1" nillable="true" type="xsd:string"/>
     <element name="cmpPhone2" nillable="true" type="xsd:string"/>
     <element name="collectSocRegDeadline" nillable="true" type="xsd:dateTime"/>
     <element name="collectSocStatus" nillable="true" type="xsd:string"/>
     <element name="company" nillable="true" type="xsd:string"/>
     <element name="creatorRegDeadline" nillable="true" type="xsd:dateTime"/>
     <element name="creatorStatus" nillable="true" type="xsd:string"/>
     <element name="distributorRegDeadline" nillable="true" type="xsd:dateTime"/>
     <element name="distributorStatus" nillable="true" type="xsd:string"/>
     <element name="email" nillable="true" type="xsd:string"/>
     <element name="final" type="xsd:boolean"/>
     <element name="finalRegDeadline" nillable="true" type="xsd:dateTime"/>
     <element name="finalStatus" nillable="true" type="xsd:string"/>
     <element name="location" nillable="true" type="xsd:string"/>
     <element name="nationDomain" nillable="true" type="xsd:string"/>
     <element name="nationality" nillable="true" type="xsd:string"/>
     <element name="nickName" nillable="true" type="xsd:string"/>
     <element name="password" nillable="true" type="xsd:string"/>
     <element name="phone" nillable="true" type="xsd:string"/>
     <element name="pubKey" nillable="true" type="xsd:string"/>
     <element name="refName" nillable="true" type="xsd:string"/>
     <element name="toolProdRegDate" nillable="true" type="xsd:dateTime"/>
     <element name="toolProdStatus" nillable="true" type="xsd:string"/>
     <element name="typeOfUser" nillable="true" type="xsd:string"/>
     <element name="webSite" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="UserRegistrationInfo">
    <sequence>
     <element name="nickName" type="xsd:string"/>
     <element name="password" type="xsd:string"/>
     <element name="regData" type="impl:UserDataType"/>
     <element name="replace" type="xsd:boolean"/>
    </sequence>
   </complexType>
   <element name="regInfo" type="impl:UserRegistrationInfo"/>
   <complexType name="RegistrationResult">
    <sequence>
     <element name="definitiveUID" nillable="true" type="xsd:string"/>
     <element name="resultStatus" type="xsd:int"/>
     <element name="errorMessage" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="registrationReturn" type="impl:RegistrationResult"/>
  </schema>
 </wsdl:types>
 <wsdl:message name="registrationResponse">
    <wsdl:part element="impl:registrationReturn" name="registrationReturn"/>
 </wsdl:message>
 <wsdl:message name="registrationRequest">
    <wsdl:part element="impl:regInfo" name="regInfo"/>
 </wsdl:message>
 <wsdl:portType name="RequestManager">
    <wsdl:operation name="registration" parameterOrder="regInfo">
       <wsdl:input message="impl:registrationRequest" name="registrationRequest"/>
       <wsdl:output message="impl:registrationResponse" name="registrationResponse"/>
    </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="AXCSUserRegistratorSoapBinding" type="impl:RequestManager">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="registration">
       <wsdlsoap:operation soapAction=""/>
       <wsdl:input name="registrationRequest">
          <wsdlsoap:body use="literal"/>
       </wsdl:input>
       <wsdl:output name="registrationResponse">
          <wsdlsoap:body use="literal"/>
       </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="AXCSUserRegistrator">
    <wsdl:port binding="impl:AXCSUserRegistratorSoapBinding" name="AXCSUserRegistrator">
       <wsdlsoap:address location="http://flauto.dsi.unifi.it:8080/axis/services/AXCSUserRegistrator"/>
    </wsdl:port>
  </wsdl:service>
 </wsdl:definitions>
```

# 37 Formal description of communication protocol AXCSObjectRegistration (DSI)

| **AXCSObjectRegistration web service** | |
|---|---|
| Method | generateAxoid |
| Description | generates a definitive AXOID to be used in the registration process |
| Input parameters | GenerationInfo genInfo – a data structure containing the nickname and password of the requesting creator and a temporary identifier (which must have the 00000 AXCS ID) |
| Output parameters | GenOutputType **-** The result output parameter is a data structure containing three fields:<br>• “resultStatus”: it is set to 0 if the registration is successful otherwise is set to 1.<br>• “axoid”: this field returns the definitive AXOID in case of success, a *null* value otherwise<br>• “errorMessage”: this filed returns the error message in case of failure, a *null* value otherwise |
| Request Sample Message | <?xml version="1.0" encoding="UTF-8"?><br>  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><br>    <soapenv:Body><br>      <genInfo xsi:type="ns1:GenerationInfo" xmlns="org:axmedis:axcs:services:objectregistrator" xmlns:ns1="org:axmedis:axcs:services:objectregistrator"><br>        <ns1:nick xsi:type="xsd:string">mario</ns1:nick><br>        <ns1:passw xsi:type="xsd:string">xxxxx</ns1:passw><br>        <ns1:tmpID xsi:type="xsd:string">urn:axmedis:00000:OBJ:3fbc293a-42fa-e7c4-15e4-a2b2f36016af</ns1:tmpID><br>      </genInfo><br>    </soapenv:Body><br>  </soapenv:Envelope> |
| Response Sample Message | <?xml version="1.0" encoding="utf-8"?><br>  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><br>    <soapenv:Body><br>      <generateAxoidReturn xmlns="org:axmedis:axcs:services:objectregistrator"><br>        <axoid>urn:axmedis:2b5a3:OBJ:66a56714-22d9-306b-9ebf-3439ac354e8c</axoid><br>        <result>0</result><br>        <errorMessage/><br>      </generateAxoidReturn><br>    </soapenv:Body><br>  </soapenv:Envelope> |

| **AXCSObjectRegistration web service** | |
|---|---|
| Method | registration |
| Description | It collects requestor credentials needful to access the system and uses the verifyLogin() (a DataManager method) to verify requestor credentials. It also collects object metadata (regInfo) provided by the requestor and uses the AXCS-DB-INTERFACE API methods provided by AXCS Database Interface to insert received information into database. Finally, it signs the object’s hash and return it to the requestor. |
| Input parameters | RegistrationInfo regInfo – a data structure containing the nickname and password of the requesting creator and a nested data structure containing the data to be inserted/updated |
| Output parameters | RegOutputType – The result output parameter is a data structure containing three fields:<br>• “resultStatus”: it is set to 0 if the registration is successful otherwise is set to 1. |

| | |
|---|---|
| | • “signature”: this field returns the hash of the object signed by the AXCS in case of success, a *null* value otherwise<br>• “errorMessage”: this filed returns the error message in case of failure, a *null* value otherwise |
| Request Sample Message | |

```
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <regInfo xsi:type="ns1:RegistrationInfo"
xmlns="org:axmedis:axcs:services:objectregistrator"
xmlns:ns1="org:axmedis:axcs:services:objectregistrator">
        <ns1:nick xsi:type="xsd:string">mario</ns1:nick>
        <ns1:objData xsi:type="ns1:ObjectDataType">
          <ns1:axcid xsi:type="xsd:string">urn:axmedis:a4bfe:CRE:4f184a98-062b-3608-
9189-680500ece26d</ns1:axcid>
          <ns1:axdid>
            <item xsi:type="xsd:string" xmlns="">urn:axmedis:cd2a5:DIS:4f184a98-062b-
3608-9189-680500ece26d</item>
          </ns1:axdid>
          <ns1:axoid xsi:type="xsd:string">urn:axmedis:2b5a3:OBJ:66a56714-22d9-306b-
9ebf-3439ac354e8c</ns1:axoid>
          <ns1:axwid xsi:type="xsd:string">urn:axmedis:00000:WRK:578afe41-3a9d-0af5-
439a-af4b563fcda1</ns1:axwid>
          <ns1:dc>
            <item xsi:type="ns1:DublinCoreData" xmlns="">
              <contributor xsi:type="xsd:string">contributor</contributor>
              <coverage xsi:type="xsd:string">coverage</coverage>
              <creatorValues>
                <item xsi:type="xsd:string">Mario</item>
                <item xsi:type="xsd:string">Luigi</item>
              </creatorValues>
              <date xsi:type="xsd:dateTime">2005-10-10T12:00:00</date>
              <description xsi:type="xsd:string">descript</description>
              <format xsi:type="xsd:string">format</format>
              <identifier xsi:type="xsd:string">id</identifier>
              <language xsi:type="xsd:string">ITA</language>
              <publisher xsi:type="xsd:string">publisher</publisher>
              <relation xsi:type="xsd:string">relattion</relation>
              <rights xsi:type="xsd:string">rights</rights>
              <source xsi:type="xsd:string">source</source>
              <subject xsi:type="xsd:string">subject</subject>
              <title xsi:type="xsd:string">title</title>
              <type xsi:type="xsd:string">type</type>
              <xmlRefFile xsi:type="xsd:string">file1.xml</xmlRefFile>
            </item>
          </ns1:dc>
          <ns1:extendedMetadata>
            <item xsi:type="ns1:ExtendedMetadataData" xmlns="">
              <axoid xsi:type="xsd:string">urn:axmedis:2b5a3:OBJ:66a56714-22d9-306b-
9ebf-3439ac354e8c</axoid>
              <metadataFieldName xsi:type="xsd:string">Casa
Discografica</metadataFieldName>
              <metadataFieldValue xsi:type="xsd:string">BMG</metadataFieldValue>
              <metadataLang xsi:type="xsd:string">ITA</metadataLang>
            </item>
            <item xsi:type="ns1:ExtendedMetadataData" xmlns="">
              <axoid xsi:type="xsd:string">urn:axmedis:2b5a3:OBJ:66a56714-22d9-306b-
9ebf-3439ac354e8c</axoid>
              <metadataFieldName xsi:type="xsd:string">Dicographic
Editor</metadataFieldName>
              <metadataFieldValue xsi:type="xsd:string">SONY</metadataFieldValue>
              <metadataLang xsi:type="xsd:string">ENG</metadataLang>
            </item>
          </ns1:extendedMetadata>
          <ns1:fingerprintInfo xsi:type="xsd:string">fingerprintInfo</ns1:fingerprintInfo>
          <ns1:hash
xsi:type="xsd:hexBinary">75465daf546d3764f3a54fa73376543546ad3f56a3f546a3fa5f35af3
5af354a3f5af3546af356a3f</ns1:hash>
          <ns1:includedObjAxoid xsi:type="soapenc:Array" xsi:nil="true"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"/>
          <ns1:includedObjProtectionStamp xsi:type="soapenc:Array" xsi:nil="true"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"/>
          <ns1:includedObjVersion xsi:type="soapenc:Array" xsi:nil="true"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"/>
          <ns1:objectNewVersion xsi:type="xsd:string" xsi:nil="true"/>
          <ns1:objectStatus xsi:type="xsd:string">U</ns1:objectStatus>
          <ns1:objectVersion xsi:type="xsd:string">1.0</ns1:objectVersion>
          <ns1:protectionInfo xsi:type="xsd:string">rot13</ns1:protectionInfo>
          <ns1:protectionStamp xsi:type="xsd:string">LOF</ns1:protectionStamp>
          <ns1:regDeadline xsi:type="xsd:dateTime">2008-12-
01T17:10:00</ns1:regDeadline>
        </ns1:objData>
        <ns1:passw xsi:type="xsd:string">xxxxx</ns1:passw>
      </regInfo>
    </soapenv:Body>
  </soapenv:Envelope>
```

Response Sample Message

```
<?xml version="1.0" encoding="utf-8"?>
  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
      <registrationReturn xmlns="org:axmedis:axcs:services:objectregistrator">

<signature>98b9ea56bcd74132ad43f2afa0f9f0656df33d11463dfd678d55a332</signature>
        <result>0</result>
        <errorMessage/>
      </registrationReturn>
    </soapenv:Body>
  </soapenv:Envelope>
```

AXCSObjectRegistration web service WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions                    targetNamespace="org:axmedis:axcs:services:objectregistrator"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="org:axmedis:axcs:services:objectregistrator"
xmlns:intf="org:axmedis:axcs:services:objectregistrator"      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
 <wsdl:types>
  <schema   elementFormDefault="qualified"   targetNamespace="org:axmedis:axcs:services:objectregistrator"
xmlns="http://www.w3.org/2001/XMLSchema">
   <complexType name="ArrayOf_xsd_string">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="item" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="DublinCoreData">
    <sequence>
     <element name="contributor" nillable="true" type="xsd:string"/>
     <element name="coverage" nillable="true" type="xsd:string"/>
     <element name="creatorValues" nillable="true" type="impl:ArrayOf_xsd_string"/>
     <element name="date" nillable="true" type="xsd:string"/>
     <element name="description" nillable="true" type="xsd:string"/>
     <element name="format" nillable="true" type="xsd:string"/>
     <element name="identifier" nillable="true" type="xsd:string"/>
     <element name="language" nillable="true" type="xsd:string"/>
     <element name="publisher" nillable="true" type="xsd:string"/>
     <element name="relation" nillable="true" type="xsd:string"/>
     <element name="rights" nillable="true" type="xsd:string"/>
     <element name="source" nillable="true" type="xsd:string"/>
     <element name="subject" nillable="true" type="xsd:string"/>
     <element name="title" nillable="true" type="xsd:string"/>
     <element name="type" nillable="true" type="xsd:string"/>
     <element name="xmlRefFile" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="ArrayOfDublinCoreData">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="item" type="impl:DublinCoreData"/>
    </sequence>
   </complexType>
   <complexType name="ExtendedMetadataData">
    <sequence>
     <element name="axoid" type="xsd:string"/>
     <element name="metadataFieldName" type="xsd:string"/>
     <element name="metadataFieldValue" nillable="true" type="xsd:string"/>
     <element name="metadataLang" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="ArrayOfExtendedMetadataData">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="item" type="impl:ExtendedMetadataData"/>
    </sequence>
   </complexType>
   <complexType name="ObjectDataType">
    <sequence>
     <element name="axcid" type="xsd:string"/>
     <element name="axdid" nillable="true" type="impl:ArrayOf_xsd_string"/>
     <element name="axoid" type="xsd:string"/>
     <element name="axwid" nillable="true" type="xsd:string"/>
     <element name="dc" type="impl:ArrayOfDublinCoreData"/>
     <element name="extendedMetadata" nillable="true" type="impl:ArrayOfExtendedMetadataData"/>
     <element name="fingerprintInfo" nillable="true" type="xsd:string"/>
     <element name="hash" type="xsd:hexBinary"/>
     <element name="includedObjAxoid" nillable="true" type="impl:ArrayOf_xsd_string"/>
     <element name="objectNewAXOID" nillable="true" type="xsd:string"/>
     <element name="objectStatus" type="xsd:string"/>
     <element name="objectVersion" nillable="true" type="xsd:string"/>
     <element name="protectionInfo" nillable="true" type="xsd:string"/>
     <element name="protectionStamp" nillable="true" type="xsd:string"/>
     <element name="regDeadline" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="RegistrationInfo">
    <sequence>
     <element name="nick" type="xsd:string"/>
     <element name="objData" type="impl:ObjectDataType"/>
     <element name="passw" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="regInfo" type="impl:RegistrationInfo"/>
   <complexType name="RegOutputType">
    <sequence>
     <element name="signature" nillable="true" type="xsd:hexBinary"/>
     <element name="result" type="xsd:int"/>
     <element name="errorMessage" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="registrationReturn" type="impl:RegOutputType"/>
   <complexType name="GenerationInfo">
    <sequence>
     <element name="nick" type="xsd:string"/>
     <element name="passw" type="xsd:string"/>
     <element name="tmpID" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="genInfo" type="impl:GenerationInfo"/>
   <complexType name="GenOutputType">
    <sequence>
     <element name="axoid" nillable="true" type="xsd:string"/>
     <element name="result" type="xsd:int"/>
     <element name="errorMessage" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="generateAxoidReturn" type="impl:GenOutputType"/>
  </schema>
 </wsdl:types>
 <wsdl:message name="generateAxoidResponse">
    <wsdl:part element="impl:generateAxoidReturn" name="generateAxoidReturn"/>
 </wsdl:message>
 <wsdl:message name="registrationResponse">
    <wsdl:part element="impl:registrationReturn" name="registrationReturn"/>
 </wsdl:message>
 <wsdl:message name="registrationRequest">
    <wsdl:part element="impl:regInfo" name="regInfo"/>
 </wsdl:message>
 <wsdl:message name="generateAxoidRequest">
    <wsdl:part element="impl:genInfo" name="genInfo"/>
 </wsdl:message>
 <wsdl:portType name="OIDGen_dealManager">
    <wsdl:operation name="registration" parameterOrder="regInfo">
      <wsdl:input message="impl:registrationRequest" name="registrationRequest"/>
      <wsdl:output message="impl:registrationResponse" name="registrationResponse"/>
    </wsdl:operation>
    <wsdl:operation name="generateAxoid" parameterOrder="genInfo">
      <wsdl:input message="impl:generateAxoidRequest" name="generateAxoidRequest"/>
      <wsdl:output message="impl:generateAxoidResponse" name="generateAxoidResponse"/>
    </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="AXCSObjectRegistratorSoapBinding" type="impl:OIDGen_dealManager">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="registration">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="registrationRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="registrationResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="generateAxoid">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="generateAxoidRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="generateAxoidResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="AXCSObjectRegistrator">
    <wsdl:port binding="impl:AXCSObjectRegistratorSoapBinding" name="AXCSObjectRegistrator">
      <wsdlsoap:address location="http://flauto.dsi.unifi.it:8080/axis/services/AXCSObjectRegistrator"/>
    </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

# 38 Formal description of communication protocol AXCSReporting (DSI)

| Call name | |
|---|---|
| Method | acceptRequest |
| Description | It is the only public method of this web service. It collects CAMART credentials needful to access the system and uses the “verifyLogin” (a DataManager method) to verify requestor credentials. It also collects the CAMART query (in a field called CAMARTQuery). CAMARTQuery contains criteria to be used to filter data returned. They are to be expressed in standard SQL syntax as a where clause (without the WHERE keyword) using only the following operators:<br><br>equal (=), not equal (<>), logical and (AND), logical or (OR)<br><br>to filter with respect to these fields: any identifier (*AXUID*, *AXDID*, *AXOID*, ecc.), *Location*, *Operation*, *RegistrationTimeStamp*, *ExecutionTimeStamp*<br>In case that a filtering with respect to dates is needed (i.e. with respect to *RegistrationTimeStamp* and/or *ExecutionTimeStamp*), the following additional operators are allowed:<br><br>greater than (>), greater than or equal (>=), less than (<), less than or equal (<=).<br>Eventually, an empty caluse can be used if no filtering is desired. |
| Input parameters | ReportingInfo repInfo – a data structure containing the nickname and password of the requesting distributor and a CAMARTQuery |
| Output parameters | ReportingResponse – The result output parameter is a data structure containing three fields:<br>• “resultStatus”: it is set to 0 if the registration is successful otherwise is set to 1.<br>• “logDataTypes”: this field returns the requested information as a set of structured data in case of success, a *null* value otherwise<br>• “errorMessage”: this filed returns the error message in case of failure, a *null* value otherwise |
| Request Sample Message | <?xml version="1.0" encoding="UTF-8"?><br>  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><br>    <soapenv:Body><br>      <repInfo xsi:type="ns1:ReportingInfo" xmlns="org:axmedis:axcs:services:reporting" xmlns:ns1="org:axmedis:axcs:services:reporting"><br>        <ns1:CAMARTQuery xsi:type="xsd:string">RegistrationTimestamp >= 2005-02-12T03:04:45</ns1:CAMARTQuery><br>        <ns1:nickName xsi:type="xsd:string">mario</ns1:nickName><br>        <ns1:password xsi:type="xsd:string">xxxxx</ns1:password><br>      </repInfo><br>    </soapenv:Body><br>  </soapenv:Envelope> |
| Response Sample Message | <?xml version="1.0" encoding="utf-8"?><br>  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><br>    <soapenv:Body><br>      <acceptRequestReturn xmlns="org:axmedis:axcs:services:reporting"><br>        <logDataTypes><br>          <item><br>            <axcid>urn:axmedis:345af:BUS:4f184a98-062b-3608-9189-680500ece26d</axcid><br>            <axcsid>urn:axmedis:575a6:BUS:a4fecd47-f315-6fc6-427b- |

```
af0b31498ca</axcsid>
            <axdid>urn:axmedis:af6bcd:BUS:578afe41-3a9d-0af5-439a-
af4b563fcda1</axdid>
            <axdom>urn:axmedis:745af:DOM:a46a5a42-5a3b-387d-93ff-
4ad26524bd34</axdom>
            <axlid>urn:axmedis:ac20b:LIC:4f184a98-062b-3608-9189-680500ece26d</axlid>
            <axoid>urn:axmedis:ac20b:OBJ:66a56714-22d9-306b-9ebf-
3439ac354e8c</axoid>
            <axtid>urn:axmedis:ac20b:ITO:9834657a-bfda-3426-46ec-6756848effdd</axtid>
            <axuid>urn:axmedis:ac20b:USR:1b4e28ba-2fa1-11d2-883f-
b9a761bde3fb</axuid>
            <axwid/>
            <estimatedHwFingerprint>EstimatedFingeprint</estimatedHwFingerprint>
            <executionTimestamp>2005-02-12T03:05:45</executionTimestamp>
            <histVerSuccess>1</histVerSuccess>
            <location>Italy</location>
            <logID>1</logID>
            <objectVersion>1.0</objectVersion>
            <operationDetails/>
            <operation>play</operation>
            <ownerName>owner</ownerName>
            <protectionStamp>protectioStamp</protectionStamp>
            <registrationTimestamp>2005-06-16T13:50:15.0</registrationTimestamp>
          </item>
        </logDataTypes>
        <resultStatus>0</resultStatus>
        <errorMesage/>
      </acceptRequestReturn>
    </soapenv:Body>
  </soapenv:Envelope>
```

AXCSReporting web service WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="org:axmedis:axcs:services:reporting"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="org:axmedis:axcs:services:reporting"
xmlns:intf="org:axmedis:axcs:services:reporting" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="org:axmedis:axcs:services:reporting"
xmlns="http://www.w3.org/2001/XMLSchema">
   <complexType name="ReportingInfo">
    <sequence>
     <element name="CAMARTQuery" nillable="true" type="xsd:string"/>
     <element name="nickName" type="xsd:string"/>
     <element name="password" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="repInfo" type="impl:ReportingInfo"/>
   <complexType name="ReportingLogDataType">
    <sequence>
     <element name="axcid" type="xsd:string"/>
     <element name="axcsid" type="xsd:string"/>
     <element name="axdid" type="xsd:string"/>
     <element name="axdom" nillable="true" type="xsd:string"/>
     <element name="axlid" type="xsd:string"/>
     <element name="axoid" type="xsd:string"/>
     <element name="axtid" type="xsd:string"/>
     <element name="axuid" type="xsd:string"/>
     <element name="axwid" nillable="true" type="xsd:string"/>
     <element name="estimatedHwFingerprint" type="xsd:string"/>
     <element name="executionTimestamp" type="xsd:dateTime"/>
     <element name="histVerSuccess" type="xsd:string"/>
     <element name="location" type="xsd:string"/>
     <element name="objectVersion" nillable="true" type="xsd:string"/>
     <element name="operationDetails" nillable="true" type="xsd:string"/>
     <element name="operation" type="xsd:string"/>
     <element name="ownerName" type="xsd:string"/>
     <element name="protectionStamp" nillable="true" type="xsd:string"/>
     <element name="registrationTimestamp" type="xsd:dateTime"/>
    </sequence>
   </complexType>
   <complexType name="ArrayOfReportingLogDataType">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="item" type="impl:ReportingLogDataType"/>
    </sequence>
   </complexType>
   <complexType name="ReportingResponse">
    <sequence>
     <element name="logDataTypes" nillable="true" type="impl:ArrayOfReportingLogDataType"/>
     <element name="resultStatus" type="xsd:string"/>
     <element name="errorMessage" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="acceptRequestReturn" type="impl:ReportingResponse"/>
  </schema>
 </wsdl:types>
 <wsdl:message name="acceptRequestRequest">
    <wsdl:part element="impl:repInfo" name="repInfo"/>
 </wsdl:message>
 <wsdl:message name="acceptRequestResponse">
    <wsdl:part element="impl:acceptRequestReturn" name="acceptRequestReturn"/>
 </wsdl:message>
 <wsdl:portType name="RequestManager">
    <wsdl:operation name="acceptRequest" parameterOrder="repInfo">
       <wsdl:input message="impl:acceptRequestRequest" name="acceptRequestRequest"/>
       <wsdl:output message="impl:acceptRequestResponse" name="acceptRequestResponse"/>
    </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="AXCSReportingSoapBinding" type="impl:RequestManager">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="acceptRequest">
       <wsdlsoap:operation soapAction=""/>
       <wsdl:input name="acceptRequestRequest">
          <wsdlsoap:body use="literal"/>
       </wsdl:input>
       <wsdl:output name="acceptRequestResponse">
          <wsdlsoap:body use="literal"/>
       </wsdl:output>
    </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="AXCSReporting">
    <wsdl:port binding="impl:AXCSReportingSoapBinding" name="AXCSReporting">
       <wsdlsoap:address location="http://flauto.dsi.unifi.it:8080/axis/services/AXCSReporting"/>
    </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

# 39 Formal description of communication protocol AXCSStatistics (DSI)

| Call name | |
|---|---|
| Method | acceptRequest |
| Description | It is the only public method of this web service. It collects CAMART credentials needful to access the system and uses the “verifyLogin” (a DataManager method) to verify requestor credentials. It also collects the CAMART query (in a field called CAMARTQuery). CAMARTQuery contains criteria to be used to filter data returned. They are to be expressed in standard SQL syntax as a where clause (without the WHERE keyword) using only the following operators: equal (=), not equal (<>), logical and (AND), logical or (OR) to filter with respect to these fields: any identifier (*AXUID*, *AXDID*, *AXOID*, ecc.), *Location*, *Operation*, *RegistrationTimeStamp*, *ExecutionTimeStamp* In case that a filtering with respect to dates is needed (i.e. with respect to *RegistrationTimeStamp* and/or *ExecutionTimeStamp*), the following additional operators are allowed: greater than (>), greater than or equal (>=), less than (<), less than or equal (<=). Eventually, an empty caluse can be used if no filtering is desired. |
| Input parameters | StatisticsInfo statInfo - a data structure containing the nickname and password of the requesting distributor and a CAMARTQuery |
| Output parameters | StatisticsResponse – The result output parameter is a data structure containing three fields: • “resultStatus”: it is set to 0 if the registration is successful otherwise is set to 1. • “logDataTypes”: this field returns the requested information as a set of structured data in case of success, a *null* value otherwise • “errorMessage”: this filed returns the error message in case of failure, a *null* value otherwise |
| Request Sample Message | `<?xml version="1.0" encoding="UTF-8"?> <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <soapenv:Body> < statInfo xsi:type="ns1:StatisticsInfo" xmlns="org:axmedis:axcs:services:statistics" xmlns:ns1="org:axmedis:axcs:services:statistics"> < ns1:CAMARTQuery xsi:type="xsd:string">RegistrationTimestamp <= 2008-02-12T03:05:45</ns1:CAMARTQuery> < ns1:nickName xsi:type="xsd:string">mario</ns1:nickName> < ns1:password xsi:type="xsd:string">xxxxx</ns1:password> </statInfo> </soapenv:Body> </soapenv:Envelope>` |
| Response Sample Message | `<?xml version="1.0" encoding="utf-8"?> <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> <soapenv:Body> <acceptRequestReturn xmlns="org:axmedis:axcs:services:statistics"> <logDataTypes> <item> <axcid>urn:axmedis:67a4c:BUS:4f184a98-062b-3608-9189-680500ece26d</axcid> <axcsid>urn:axmedis:3453b:BUS:a4fecd47-f315-6fc6-427b-` |

```
af0b31498ca</axcsid>
            <axdid>urn:axmedis:adf08:BUS:578afe41-3a9d-0af5-439a-af4b563fcda1</axdid>
            <axdom>urn:axmedis:745af:DOM:a46a5a42-5a3b-387d-93ff-
4ad26524bd34</axdom>
            <axoid>urn:axmedis:675da:OBJ:66a56714-22d9-306b-9ebf-
3439ac354e8c</axoid>
            <axwid/>
            <executionTimestamp>2005-06-09T14:43:42.0</executionTimestamp>
            <location>Italy</location>
            <objectVersion>1.0</objectVersion>
            <operationDetails/>
            <operation>play</operation>
            <ownerName>owner</ownerName>
            <protectionStamp/>
            <registrationTimestamp>2005-06-16T13:50:15.0</registrationTimestamp>
          </item>
        </logDataTypes>
        <resultStatus>0</resultStatus>
        <errorMesage/>
      </acceptRequestReturn>
    </soapenv:Body>
  </soapenv:Envelope>
```

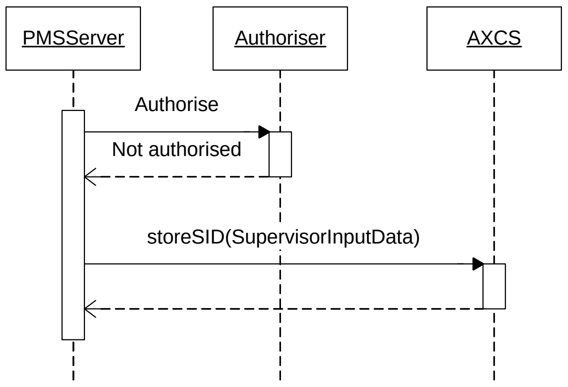
AXCSStatistics web service WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="org:axmedis:axcs:services:statistics"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="org:axmedis:axcs:services:statistics"
xmlns:intf="org:axmedis:axcs:services:statistics" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="org:axmedis:axcs:services:statistics"
xmlns="http://www.w3.org/2001/XMLSchema">
   <complexType name="StatisticsInfo">
    <sequence>
     <element name="CAMARTQuery" nillable="true" type="xsd:string"/>
     <element name="nickName" type="xsd:string"/>
     <element name="password" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="statInfo" type="impl:StatisticsInfo"/>
   <complexType name="StatisticsLogDataType">
    <sequence>
     <element name="axcid" type="xsd:string"/>
     <element name="axcsid" type="xsd:string"/>
     <element name="axdid" type="xsd:string"/>
     <element name="axdom" nillable="true" type="xsd:string"/>
     <element name="axoid" type="xsd:string"/>
     <element name="axwid" nillable="true" type="xsd:string"/>
     <element name="executionTimestamp" type="xsd:dateTime"/>
     <element name="location" type="xsd:string"/>
     <element name="objectVersion" nillable="true" type="xsd:string"/>
     <element name="operationDetails" nillable="true" type="xsd:string"/>
     <element name="operation" type="xsd:string"/>
     <element name="ownerName" type="xsd:string"/>
     <element name="protectionStamp" nillable="true" type="xsd:string"/>
     <element name="registrationTimestamp" type="xsd:dateTime"/>
    </sequence>
   </complexType>
   <complexType name="ArrayOfStatisticsLogDataType">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="item" type="impl:StatisticsLogDataType"/>
    </sequence>
   </complexType>
   <complexType name="StatisticsResponse">
    <sequence>
     <element name="logDataTypes" nillable="true" type="impl:ArrayOfStatisticsLogDataType"/>
     <element name="resultStatus" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <element name="acceptRequestReturn" type="impl:StatisticsResponse"/>
  </schema>
 </wsdl:types>
 <wsdl:message name="acceptRequestResponse">
    <wsdl:part element="impl:acceptRequestReturn" name="acceptRequestReturn"/>
 </wsdl:message>
 <wsdl:message name="acceptRequestRequest">
    <wsdl:part element="impl:statInfo" name="statInfo"/>
 </wsdl:message>
 <wsdl:portType name="RequestManager">
    <wsdl:operation name="acceptRequest" parameterOrder="statInfo">
       <wsdl:input message="impl:acceptRequestRequest" name="acceptRequestRequest"/>
       <wsdl:output message="impl:acceptRequestResponse" name="acceptRequestResponse"/>
    </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="AXCSStatisticsSoapBinding" type="impl:RequestManager">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="acceptRequest">
       <wsdlsoap:operation soapAction=""/>
       <wsdl:input name="acceptRequestRequest">
          <wsdlsoap:body use="literal"/>
       </wsdl:input>
       <wsdl:output name="acceptRequestResponse">
          <wsdlsoap:body use="literal"/>
       </wsdl:output>
    </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="AXCSStatistics">
    <wsdl:port binding="impl:AXCSStatisticsSoapBinding" name="AXCSStatistics">
       <wsdlsoap:address location="http://flauto.dsi.unifi.it:8080/axis/services/AXCSStatistics"/>
    </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

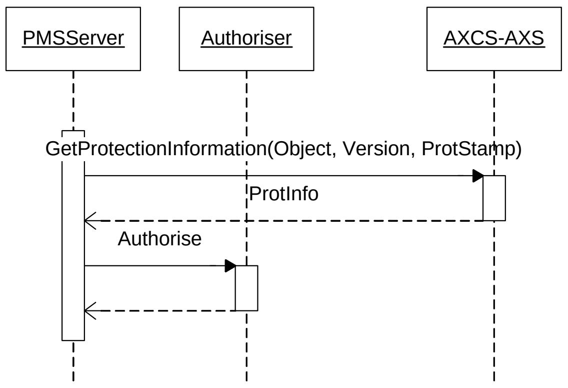
# 40 Formal description of communication protocol for AXS Web Services (FUPF)

Refer to section 5, “AXMEDIS Supervisor, AXS” to find a formal description of the methods provided by AXS library (storeSID, storeListActionLog, storePMSActionLog, getProtectionInfo, updateProtectionInfo) and AXS Web Service (storeSID,  getProtectionInfo, updateProtectionInfo). The sequence diagram for AXS Web Service methods diagrams are provided next.

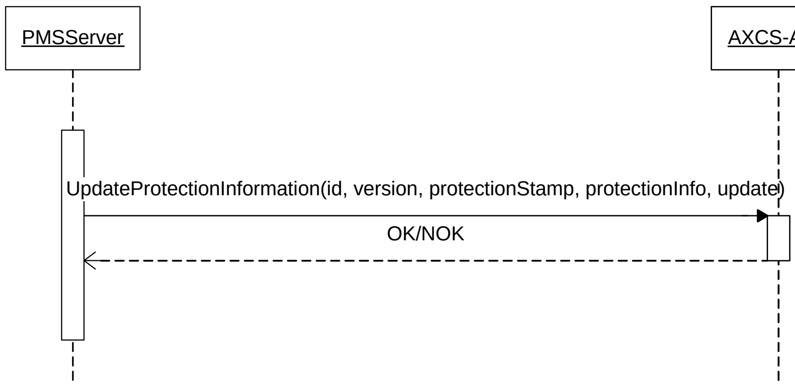
storeSID protocol



getProtectionInfo protocol



updateProtectionInfo protocol


PMSServer
AXCS-AXS
UpdateProtectionInformation(id, version, protectionStamp, protectionInfo, update)
OK/NOK

AXS Web Services WSDL:

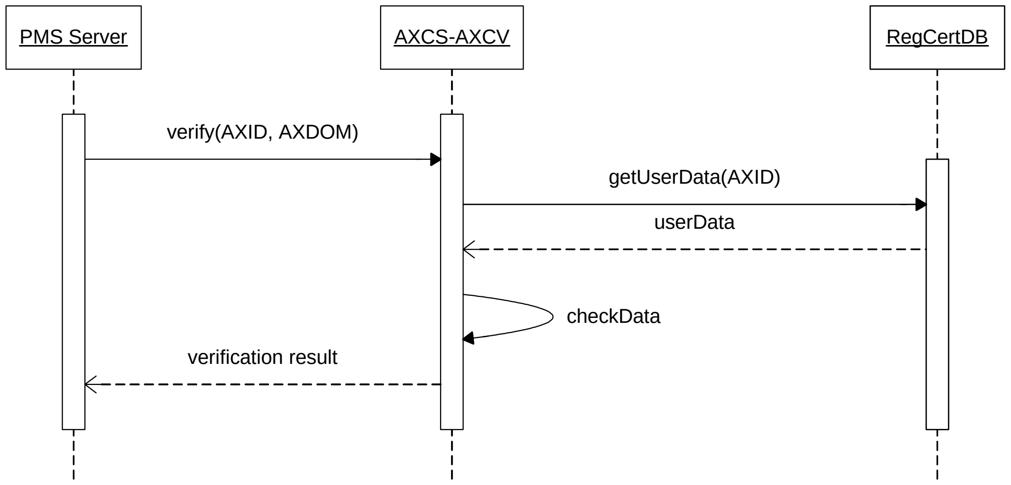
```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:AXS" xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="urn:AXS"
xmlns:intf="urn:AXS" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!--WSDL created by Apache Axis version: 1.2.1
Built on Jun 14, 2005 (09:15:57 EDT)-->
    <wsdl:types>
        <schema elementFormDefault="qualified" targetNamespace="urn:AXS" xmlns="http://www.w3.org/2001/XMLSchema">
            <element name="storeSID">
                <complexType>
                    <sequence>
                        <element name="mySid" type="impl:SupervisorInputData"/>
                    </sequence>
                </complexType>
            </element>
            <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="instantLastFPPA" 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="operationDetailsID" nillable="true" type="xsd:string"/>
                    <element name="operationID" 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="SupervisorInputData">
                <complexContent>
                    <extension base="impl:ActionLog">
                        <sequence>
                            <element name="additionalData" nillable="true" type="xsd:string"/>
                        </sequence>
                    </extension>
                </complexContent>
            </complexType>
            <element name="storeSIDResponse">
                <complexType>
                    <sequence>
                        <element name="storeSIDReturn" type="xsd:int"/>
                    </sequence>
                </complexType>
            </element>
            <element name="getProtectionInfo">
                <complexType>
                    <sequence>
                        <element name="object" type="xsd:string"/>
                        <element name="version" type="xsd:string"/>
                        <element name="protectionStamp" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="getProtectionInfoResponse">
                <complexType>
                    <sequence>
                        <element name="getProtectionInfoReturn" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="updateProtectionInfo">
                <complexType>
                    <sequence>
                        <element name="id" type="xsd:string"/>
                        <element name="version" type="xsd:string"/>
                        <element name="protectionStamp" type="xsd:string"/>
                        <element name="protectionInfo" type="xsd:string"/>
                        <element name="update" type="xsd:int"/>
                    </sequence>
                </complexType>
            </element>
            <element name="updateProtectionInfoResponse">
                <complexType>
                    <sequence>
                        <element name="updateProtectionInfoReturn" type="xsd:int"/>
                    </sequence>
                </complexType>
            </element>
        </schema>
    </wsdl:types>
    <wsdl:message name="updateProtectionInfoRequest">
        <wsdl:part element="impl:updateProtectionInfo" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="updateProtectionInfoResponse">
        <wsdl:part element="impl:updateProtectionInfoResponse" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="storeSIDRequest">
        <wsdl:part element="impl:storeSID" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="storeSIDResponse">
        <wsdl:part element="impl:storeSIDResponse" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="getProtectionInfoRequest">
        <wsdl:part element="impl:getProtectionInfo" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="getProtectionInfoResponse">
        <wsdl:part element="impl:getProtectionInfoResponse" name="parameters"/>
    </wsdl:message>
    <wsdl:portType name="Supervisor">
        <wsdl:operation name="storeSID">
            <wsdl:input message="impl:storeSIDRequest" name="storeSIDRequest"/>
            <wsdl:output message="impl:storeSIDResponse" name="storeSIDResponse"/>
        </wsdl:operation>
        <wsdl:operation name="getProtectionInfo">
            <wsdl:input message="impl:getProtectionInfoRequest" name="getProtectionInfoRequest"/>
```

```
                  <wsdl:output message="impl:getProtectionInfoResponse" name="getProtectionInfoResponse"/>
            </wsdl:operation>
            <wsdl:operation name="updateProtectionInfo">
                  <wsdl:input message="impl:updateProtectionInfoRequest" name="updateProtectionInfoRequest"/>
                  <wsdl:output message="impl:updateProtectionInfoResponse" name="updateProtectionInfoResponse"/>
            </wsdl:operation>
      </wsdl:portType>
      <wsdl:binding name="AXSSoapBinding" type="impl:Supervisor">
            <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
            <wsdl:operation name="storeSID">
                  <wsdlsoap:operation soapAction=""/>
                  <wsdl:input name="storeSIDRequest">
                        <wsdlsoap:body use="literal"/>
                  </wsdl:input>
                  <wsdl:output name="storeSIDResponse">
                        <wsdlsoap:body use="literal"/>
                  </wsdl:output>
            </wsdl:operation>
            <wsdl:operation name="getProtectionInfo">
                  <wsdlsoap:operation soapAction=""/>
                  <wsdl:input name="getProtectionInfoRequest">
                        <wsdlsoap:body use="literal"/>
                  </wsdl:input>
                  <wsdl:output name="getProtectionInfoResponse">
                        <wsdlsoap:body use="literal"/>
                  </wsdl:output>
            </wsdl:operation>
            <wsdl:operation name="updateProtectionInfo">
                  <wsdlsoap:operation soapAction=""/>
                  <wsdl:input name="updateProtectionInfoRequest">
                        <wsdlsoap:body use="literal"/>
                  </wsdl:input>
                  <wsdl:output name="updateProtectionInfoResponse">
                        <wsdlsoap:body use="literal"/>
                  </wsdl:output>
            </wsdl:operation>
      </wsdl:binding>
      <wsdl:service name="SupervisorService">
            <wsdl:port binding="impl:AXSSoapBinding" name="AXS">
                  <wsdlsoap:address location="<endpoint>/AXS"/>
            </wsdl:port>
      </wsdl:service>
</wsdl:definitions>
```

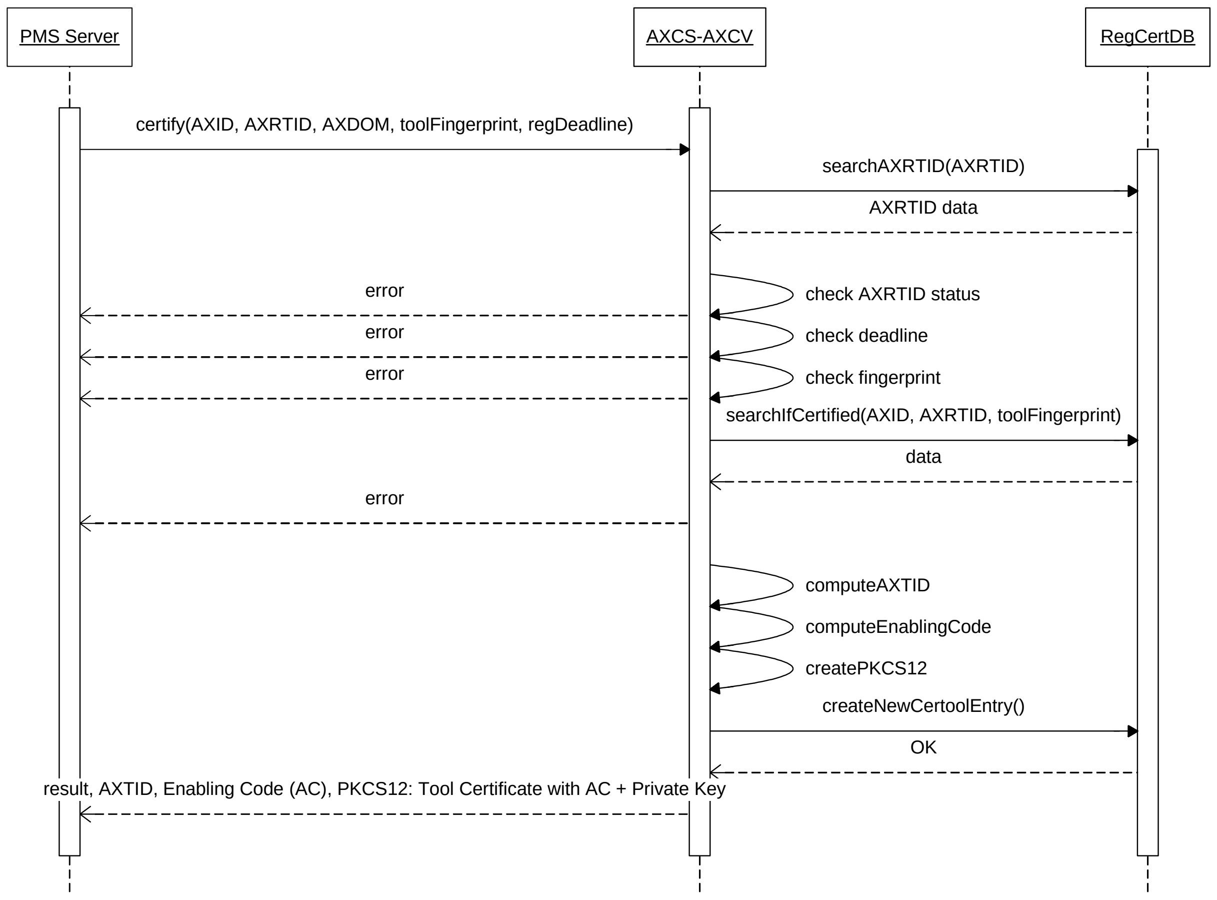
# 41 Formal description of communication protocol for AXCV Web Services (FUPF)

Refer to section 4, “AXMEDIS Certification and Certification, AXCV” to find a formal description of the methods provided by AXCV library and Web Service (verifyUser, certify, verify, reverify, verifyPmsActionLog and ping). Next, the AXCV Web Service methods diagrams are provided.

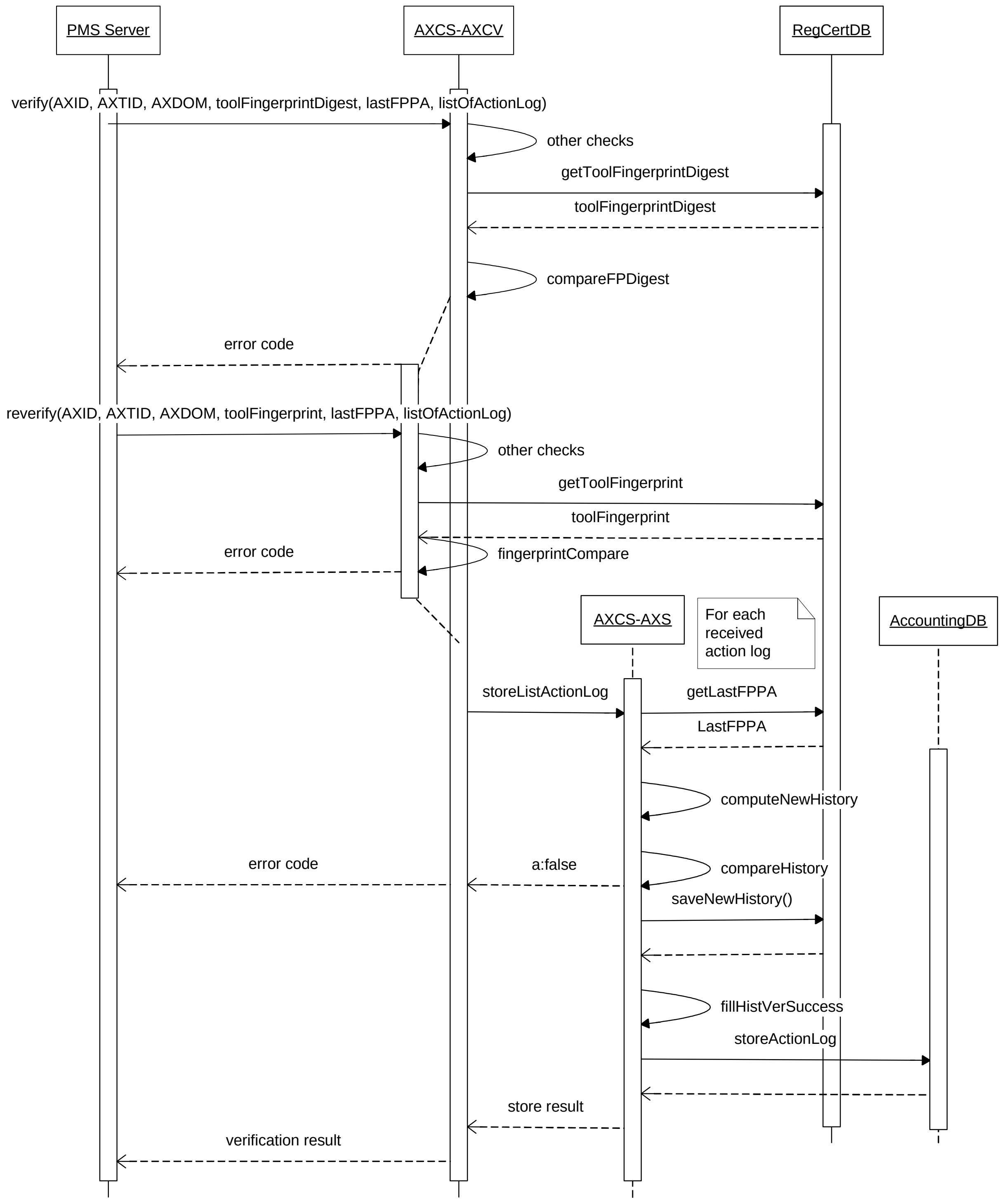
VerifyUser protocol



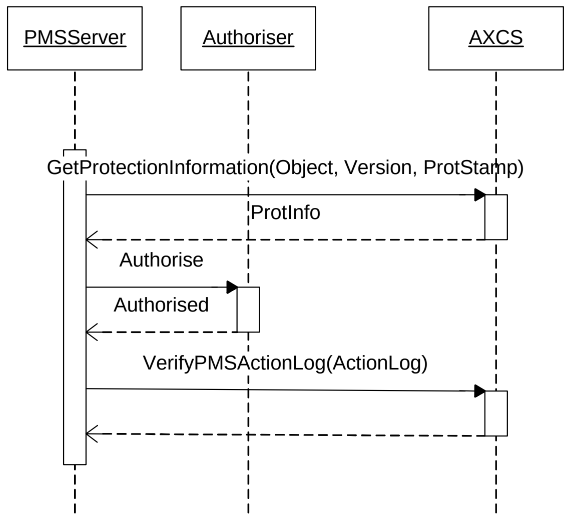
certify protocol



verify and reverify protocol



VerifyPmsActionLog protocol


PMSServer
Authoriser
AXCS
GetProtectionInformation(Object, Version, ProtStamp)
ProtInfo
Authorise
Authorised
VerifyPMSActionLog(ActionLog)

AXCV Web Services WSDL:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:AXCV" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="urn:AXCV" xmlns:intf="urn:AXCV" xmlns:tns2="http://axs.axmedis.dmag.fupf.es"
xmlns:tns3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!--WSDL created by Apache Axis version: 1.2.1
Built on Jun 14, 2005 (09:15:57 EDT)-->
    <wsdl:types>
        <schema elementFormDefault="qualified" targetNamespace="urn:AXCV"
xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="http://axs.axmedis.dmag.fupf.es"/>
            <element name="ping">
                <complexType>
                    <sequence>
                        <element name="input" type="xsd:int"/>
                    </sequence>
                </complexType>
            </element>
            <element name="pingResponse">
                <complexType>
                    <sequence>
                        <element name="pingReturn" type="xsd:int"/>
                    </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="impl:VerificationResult"/>
                    </sequence>
                </complexType>
            </element>
            <complexType name="VerificationResult">
                <sequence>
                    <element name="storeListActionLogResult" type="xsd:int"/>
                    <element name="verificationResult" type="xsd:int"/>
                </sequence>
```

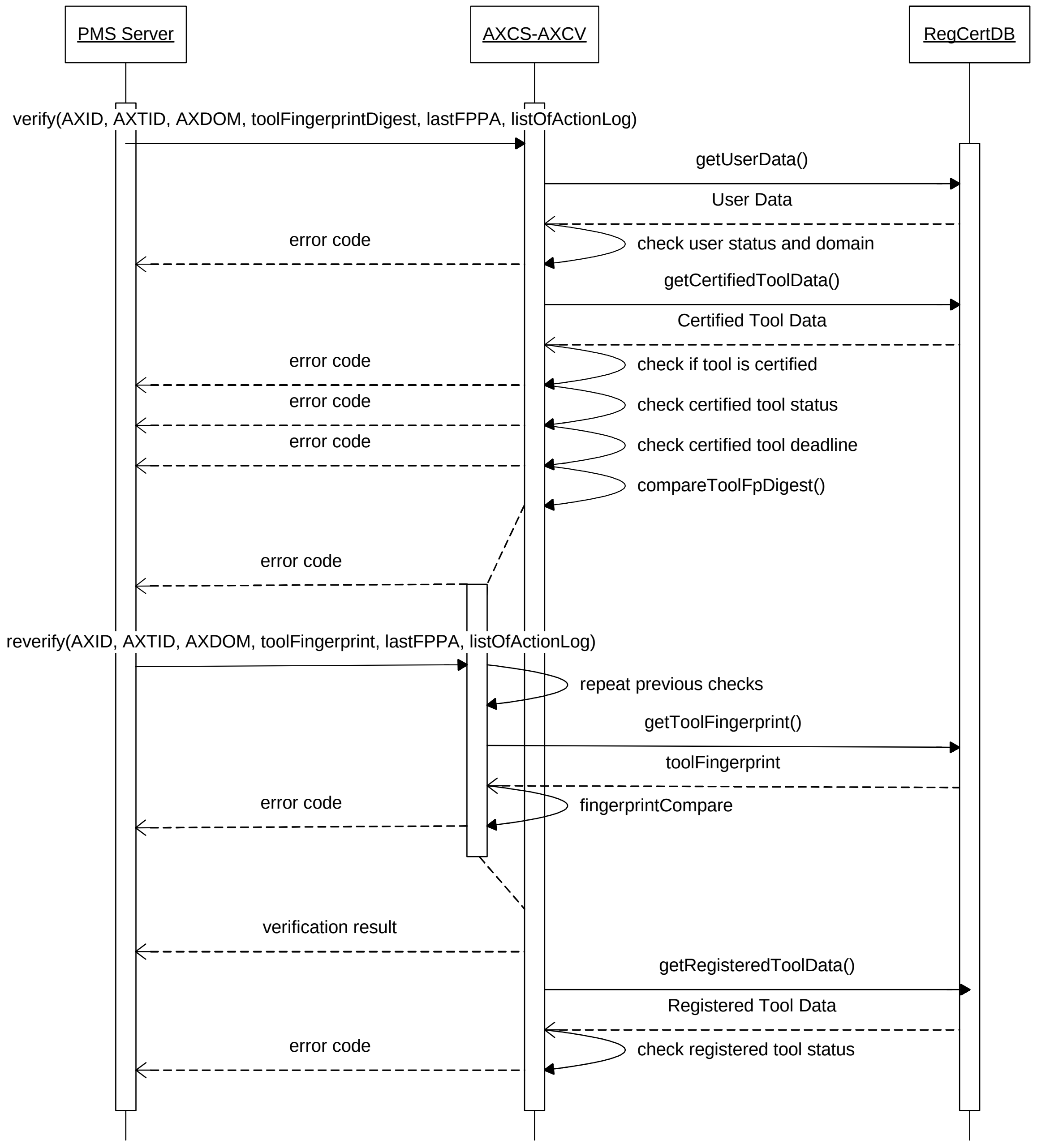
```
			</complexType>
			<element name="certify">
				<complexType>
					<sequence>
						<element name="axid" type="xsd:string"/>
						<element name="axrtid" type="xsd:string"/>
						<element name="axdom" type="xsd:string"/>
						<element name="toolFingerprint" type="xsd:string"/>
						<element name="regDeadline" type="xsd:string"/>
					</sequence>
				</complexType>
			</element>
			<element name="certifyResponse">
				<complexType>
					<sequence>
						<element name="certifyReturn" type="impl:CertificationResult"/>
					</sequence>
				</complexType>
			</element>
			<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>
			<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="tns2:ActionLog"/>
					</sequence>
				</complexType>
			</element>
			<element name="reverifyResponse">
				<complexType>
					<sequence>
						<element name="reverifyReturn" type="impl:VerificationResult"/>
					</sequence>
				</complexType>
			</element>
			<element name="verifyPmsActionLog">
				<complexType>
					<sequence>
						<element name="pmsActionLog" type="tns2:ActionLog"/>
					</sequence>
				</complexType>
			</element>
			<element name="verifyPmsActionLogResponse">
				<complexType>
					<sequence>
						<element name="verifyPmsActionLogReturn" type="impl:VerificationResult"/>
					</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="tns2:ActionLog"/>
					</sequence>
				</complexType>
```

```
            </element>
            <element name="verifyResponse">
                <complexType>
                    <sequence>
                        <element name="verifyReturn" type="impl:VerificationResult"/>
                    </sequence>
                </complexType>
            </element>
        </schema>
        <schema elementFormDefault="qualified" targetNamespace="http://axs.axmedis.dmag.fupf.es"
xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="urn:AXCV"/>
            <complexType name="ActionLog">
                <sequence>
                    <element name="AXCID" nillable="true" type="tns3:string"/>
                    <element name="AXCSID" nillable="true" type="tns3:string"/>
                    <element name="AXDID" nillable="true" type="tns3:string"/>
                    <element name="AXDOM" nillable="true" type="tns3:string"/>
                    <element name="AXLID" nillable="true" type="tns3:string"/>
                    <element name="AXOID" nillable="true" type="tns3:string"/>
                    <element name="AXTID" nillable="true" type="tns3:string"/>
                    <element name="AXUID" nillable="true" type="tns3:string"/>
                    <element name="AXWID" nillable="true" type="tns3:string"/>
                    <element name="estimatedHwFingerprint" nillable="true" type="tns3:string"/>
                    <element name="executionTimestamp" nillable="true" type="tns3:string"/>
                    <element name="histVerSuccess" nillable="true" type="tns3:string"/>
                    <element name="instantLastFPPA" nillable="true" type="tns3:string"/>
                    <element name="location" nillable="true" type="tns3:string"/>
                    <element name="logID" nillable="true" type="tns3:string"/>
                    <element name="objectVersion" nillable="true" type="tns3:string"/>
                    <element name="operationDetailsID" nillable="true" type="tns3:string"/>
                    <element name="operationID" nillable="true" type="tns3:string"/>
                    <element name="ownerName" nillable="true" type="tns3:string"/>
                    <element name="protectionStamp" nillable="true" type="tns3:string"/>
                    <element name="registrationTimestamp" nillable="true" type="tns3:string"/>
                </sequence>
            </complexType>
        </schema>
    </wsdl:types>
    <wsdl:message name="verifyUserRequest">
        <wsdl:part element="impl:verifyUser" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="verifyRequest">
        <wsdl:part element="impl:verify" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="verifyPmsActionLogRequest">
        <wsdl:part element="impl:verifyPmsActionLog" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="verifyUserResponse">
        <wsdl:part element="impl:verifyUserResponse" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="pingRequest">
        <wsdl:part element="impl:ping" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="reverifyResponse">
        <wsdl:part element="impl:reverifyResponse" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="pingResponse">
        <wsdl:part element="impl:pingResponse" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="verifyResponse">
        <wsdl:part element="impl:verifyResponse" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="certifyResponse">
        <wsdl:part element="impl:certifyResponse" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="reverifyRequest">
        <wsdl:part element="impl:reverify" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="certifyRequest">
        <wsdl:part element="impl:certify" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="verifyPmsActionLogResponse">
        <wsdl:part element="impl:verifyPmsActionLogResponse" name="parameters"/>
    </wsdl:message>
    <wsdl:portType name="CertificationAndVerification">
        <wsdl:operation name="ping">
            <wsdl:input message="impl:pingRequest" name="pingRequest"/>
            <wsdl:output message="impl:pingResponse" name="pingResponse"/>
        </wsdl:operation>
        <wsdl:operation name="verifyUser">
            <wsdl:input message="impl:verifyUserRequest" name="verifyUserRequest"/>
            <wsdl:output message="impl:verifyUserResponse" name="verifyUserResponse"/>
        </wsdl:operation>
        <wsdl:operation name="certify">
            <wsdl:input message="impl:certifyRequest" name="certifyRequest"/>
            <wsdl:output message="impl:certifyResponse" name="certifyResponse"/>
        </wsdl:operation>
        <wsdl:operation name="reverify">
            <wsdl:input message="impl:reverifyRequest" name="reverifyRequest"/>
            <wsdl:output message="impl:reverifyResponse" name="reverifyResponse"/>
        </wsdl:operation>
        <wsdl:operation name="verifyPmsActionLog">
            <wsdl:input message="impl:verifyPmsActionLogRequest" name="verifyPmsActionLogRequest"/>
            <wsdl:output message="impl:verifyPmsActionLogResponse" name="verifyPmsActionLogResponse"/>
        </wsdl:operation>
        <wsdl:operation name="verify">
            <wsdl:input message="impl:verifyRequest" name="verifyRequest"/>
            <wsdl:output message="impl:verifyResponse" name="verifyResponse"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="AXCVSoapBinding" type="impl:CertificationAndVerification">
        <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="ping">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="pingRequest">
                <wsdlsoap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="pingResponse">
                <wsdlsoap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="verifyUser">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="verifyUserRequest">
                <wsdlsoap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="verifyUserResponse">
                <wsdlsoap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="certify">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="certifyRequest">
                <wsdlsoap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="certifyResponse">
                <wsdlsoap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="reverify">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="reverifyRequest">
                <wsdlsoap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="reverifyResponse">
                <wsdlsoap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="verifyPmsActionLog">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="verifyPmsActionLogRequest">
```

```
                <wsdlsoap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="verifyPmsActionLogResponse">
                <wsdlsoap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="verify">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="verifyRequest">
                <wsdlsoap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="verifyResponse">
                <wsdlsoap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="CertificationAndVerificationService">
        <wsdl:port binding="impl:AXCVSoapBinding" name="AXCV">
            <wsdlsoap:address location="<endpoint>/AXCV"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

## 42 Asynchronous Tool Verification (Authentication, no action log only verify) (FUPF)

Here is reported the diagram for “Asynchronous tool verification”, which is a particular case for the verify and reverify protocol described in previous section, where the input Action Log list is empty, but represented in more detail.



## 43 Bibliography (mandatory)

UUID information: http://www.opengroup.org/onlinepubs/9629399/apdxa.htm

Javadoc of AXCS-DB-Interface library: https://cvs.axmedis.org/repos/Framework/doc/code/axcs.

## 44 Glossary (mandatory)

See sections 22 and 23 for the meanings of the various prefixes and IDs.