Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE3.1.2F
Framework and Tools Specifications (AXEPTool and Progr. and Pub.)

Version: 3.2
Date: 15/03/2005
Responsible: CRS4 and UNIVLEEDS (DSI supervision)

Project Number: IST-2-511299 Project Title: AXMEDIS Deliverable Type: report Visible to User Groups: NO Visible to Affiliated: NO Visible to the Public: NO.

Deliverable Number: DE3.1.2 part F Contractual Date of Delivery: January 2005 Actual Date of Delivery: 15 March 2005 Title of Deliverable: Document Work-Package contributing to the Deliverable: WP3.1 Task contributing to the Deliverable: WP3, WP2 Nature of the Deliverable: report Author(s): CRS4, UNIVLEEDS,

Abstract: This document specifies the architecture of two areas of the AXMEDIS framework: the AXEPTool area and the Program and Publication Area. For each area have been identified main modules and these modules have been decomposed in subcomponents. Collaboration and sequence diagrams are provided to highlight the interdependencies among subcomponents and the dependencies with external tools and engines.
Keyword List: Peer-to-Peer, Program and Publication, Query User Interface, Automatic loading, Automatic publishing, File serving, File downloading.

AXMEDIS Copyright Notice

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

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

1. **DEFINITIONS**
   - i. "**Acceptance Date**" is the date on which these terms and conditions for entering into possession of the document have been accepted.
   - ii. "**Copyright**" stands for any content, document or portion of it that is covered by the copyright disclaimer in a Document.
   - iii. "**Licensor**" is AXMEDIS Consortium as a de-facto consortium of the EC project and any of its derivations in terms of companies and/or associations, see www.axmedis.org
   - iv. "**Document**" means the information contained in any electronic file, which has been published by the Licensor's as AXMEDIS official document and listed in the web site mentioned above or available by any other means.
   - v. "**Works**" means any works created by the licensee, which reproduce a Document or any of its part.
2. **LICENCE**
   1. The Licensor grants a non-exclusive royalty free licence to reproduce and use the Documents subject to present terms and conditions (the **Licence**) for the parts that are own and proprietary property the of AXMEDIS consortium or its members.
   2. In consideration of the Licensor granting the Licence, licensee agrees to adhere to the following terms and conditions.
3. **TERM AND TERMINATION**
   1. Granted Licence shall commence on Acceptance Date.
   2. Granted Licence will terminate automatically if licensee fails to comply with any of the terms and conditions of this Licence.
   3. Termination of this Licence does not affect either party's accrued rights and obligations as at the date of termination.
   4. Upon termination of this Licence for whatever reason, licensee shall cease to make any use of the accessed Copyright.
   5. All provisions of this Licence, which are necessary for the interpretation or enforcement of a party's rights or obligations, shall survive termination of this Licence and shall continue in full force and effect.
   6. Notwithstanding License termination, confidentiality clauses related to any content, document or part of it as stated in the document itself will remain in force for a period of 5 years after license issue date or the period stated in the document whichever is the longer.
4. **USE**
   1. Licensee shall not breach or denigrate the integrity of the Copyright Notice and in particular shall not:
      - i. remove this Copyright Notice on a Document or any of its reproduction in any form in which those may be achieved;
      - ii. change or remove the title of a Document;
      - iii. use all or any part of a Document as part of a specification or standard not emanating from the Licensor without the prior written consent of the Licensor; or
      - iv. do or permit others to do any act or omission in relation to a Document which is contrary to the rights and obligations as stated in the present license and agreed with the Licensor
5. **COPYRIGHT NOTICES**
   1. All Works shall bear a clear notice asserting the Licensor's Copyright. The notice shall use the wording employed by the Licensor in its own copyright notice unless the Licensor otherwise instructs licensees.
6. **WARRANTY**
   1. The Licensor warrants the licensee that the present licence is issued on the basis of full Copyright ownership or re-licensing agreements granting the Licensor full licensing and enforcement power.

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

7. **INFRINGEMENT**

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

8. **GOVERNING LAW AND JURISDICTION**

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

Please note that:

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

Table of Content

1 Executive Summary and Report Scope

The full AXMEDIS specification document has been decomposed in the following parts:
A. general aspects up to the description of the content model
B. Viewers and players, including plug ins, etc.
C. Content Production tools and algorithms
D. Fingerprint and descriptors algorithms and tools
E. Database area, query support and Content Crawling from CMS
F. AXEPTool area, for B2B distribution and Programme and Publication for B2C distribution
G. Workflow aspects and tools
H. Protection tools and support, Certification and Supervision and Accounting tools
I. Distribution tools and AXMEDIS Portal
J. Definitions, tables, terminology, acronyms, lists, references, links and Appendixes

This documents contains Part F with the specifications of the AXEPTool and of the Program and Publication Engine only.

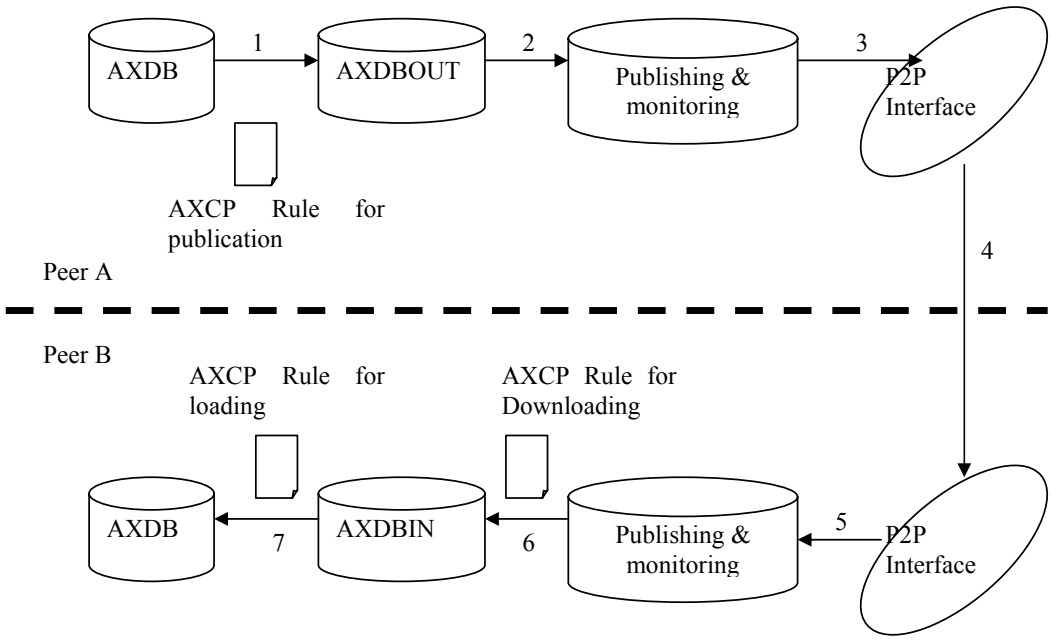
1.1 AXEPTool Summary

The AXEPTool is the tool used by factories to share their objects in an overlay network. By means of the AXEPTool is possible to perform the following operations:

- Searching objects on the network using an advanced query user interface and query model.
- Downloading objects from the network
- Loading objects in the factory internal Axmedis Database
- Publishing objects in the network making them available to other factories
- Sending/Receiving remote events in order to keep factories informed about the release of new versions of objects.

Content publishing and loading in the AXEPTool is driven by AXCP Rules. Three different types of execution are defined for publishing, loading, and downloading objects:

- Publication: when executed publishes a selection of objects in the P2P network. The component performing the publication is the **Publication Component of AXEPTool**
- Loading: when executed loads a selection of objects from AXDBIN to AXDB. The component performing the publication is the **Loading Component of AXEPTool**
- Downloading: when executed downloads a selection of objects from the P2P network to the AXDBIN. The component performing the publication is the **P2P Active Selections Component of AXEPTool**



Rules are loaded and executed by the AXCP Rule Engine. ***Notice from now on in this document we refer to publication, loading and downloading rules as AXCP rules whose script is aimed respctively at publish, load, and download objects. The AXCP Rule is the unique data structure defining a rule in the scope of AXEPTool. The section Program and Publication defines an its own rule model..***

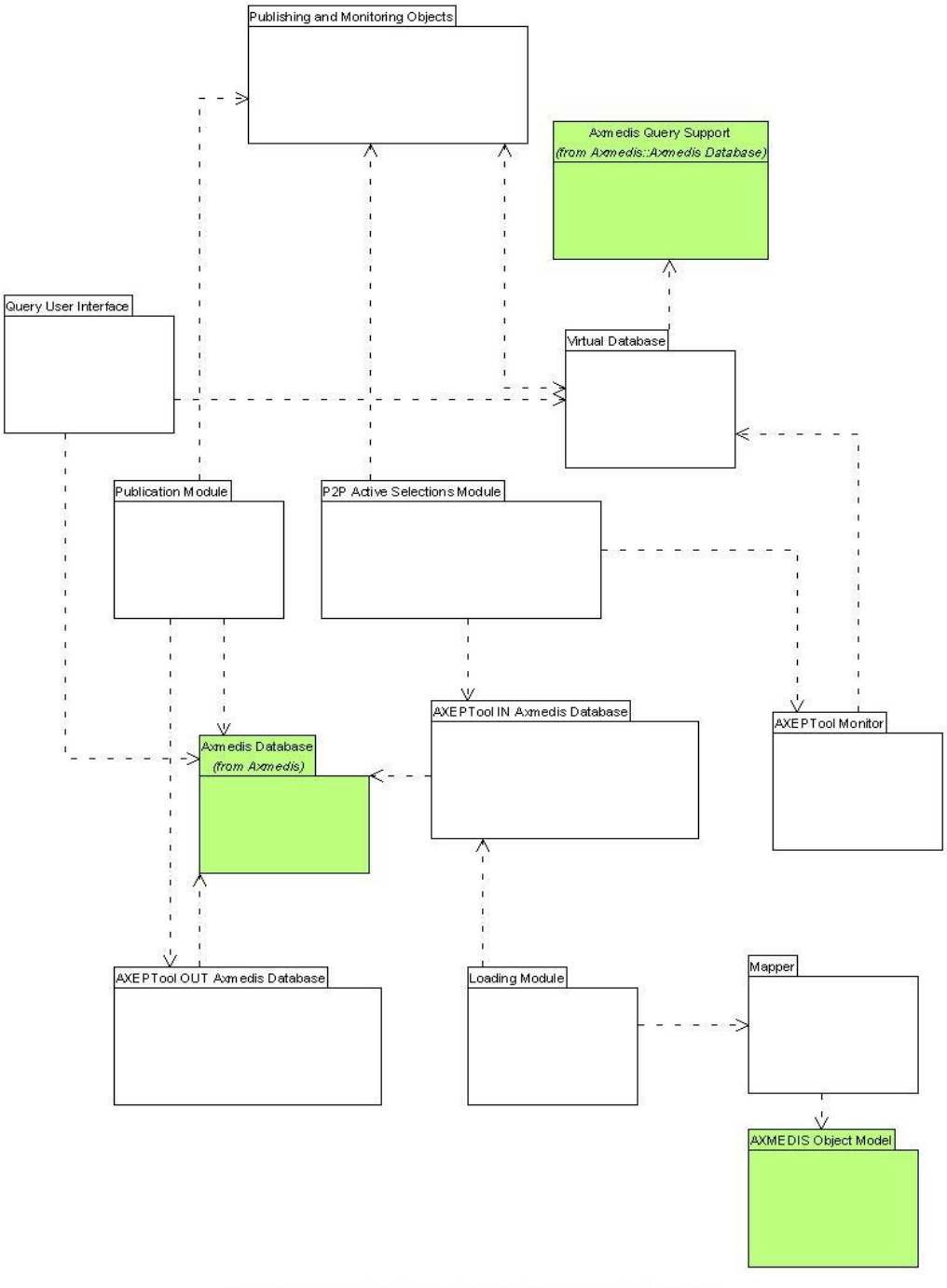
1. when a publication rule is executed a selection of objects is published. Objects are indexed in the AXDBOUT
2. a publication event is produced
3. the publication event is sent to the P2P network
4. a node B interested in such event is notified
5. the event is enqueued in peer B
6. when a Downloading rule is executed, the P2P Active Selection Component checks if some objects are missing in AXDBIN and eventually starts a download
7. when a Loading rule is executed checks if in AXDBIN there are some objects that need to be reloaded in AXDB

1.2 Summary on Program and Publication Area

The Programme and Publication (P&P) Area consists of two parts: (i) P&P Editor and (ii) P&P Engine. The P&P Editor provide the Actor a GUI To specify the P&P Programme and the P&P Engine is a continuous running engine which process the programme. The Programme and Publication process is driven by P&P programme which consists of a set of rules as specified by the Actor using the P&P Editor, to delivery specific AXMEDIS objects to the distribution channel as specified by the Actor. The P&P process also allows on-demand requests that responds to the distribution and end user needs. The P&P Engine also responsible to adopt the AXMEDIS Object for proper consumsion depending on the targeted delivery channel. It exploits the Formatting Engine if the AXMEDIS object does not match the distribution channel.

2 AXEPTool area (P2P on B2B) (WP4.4.1: CRS4, WP5.5: CRS4, WP5.5.1: CRS4)

2.1 AXEPTool Area Package Diagram and Dependency Relations

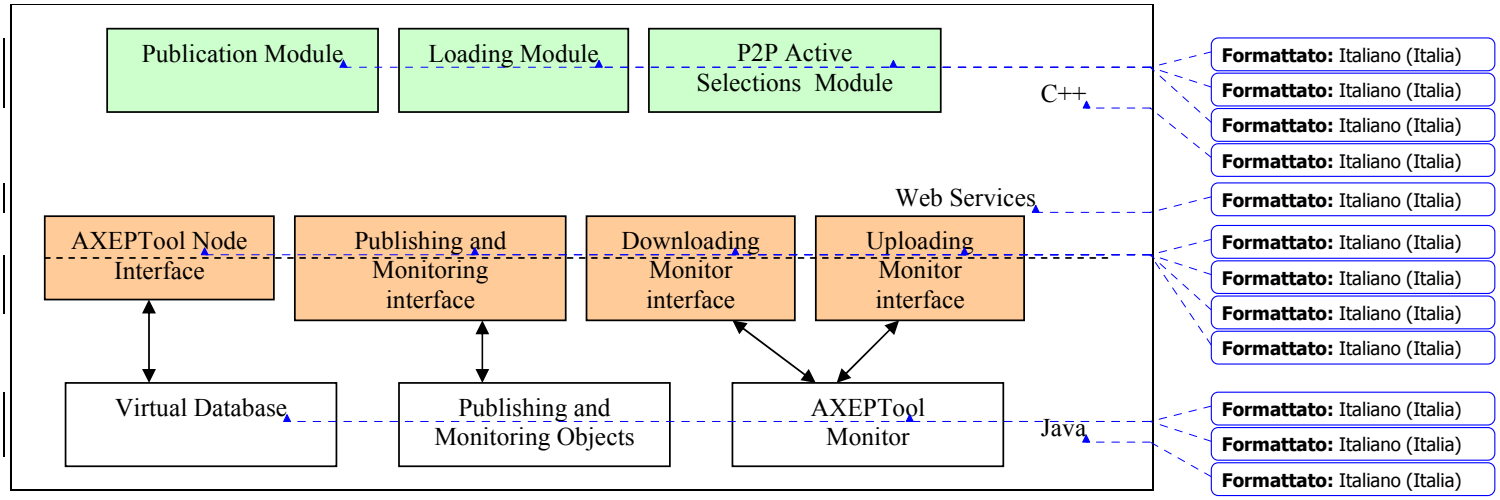


The AXEPTool area includes:

- Publication Module of AXEPTool: it provides functions to publish AXMEDIS objects from the AXMEDIS Data Base, and to move it in the AXDBOUT.
- Loading Module of AXEPTool: it load an AXMEDIS object from the AXDBIN to the AXMEDIS Data Base.
- AXEPTool IN/OUT Data Base are instances of the Axmedis database used.
- AXEPTool P2P Active Selection Module. it performs the activate the downloading of objects from the network.
- Publishing and Monitoring Objects. It defines a event broadcast/unicast/listening mechanism to keep peers informed about new objects/versions shared in the network.
- Virtual Database: It is the interface to the P2P network .
- AXEPTool Monitor: this module creates and manages upload and download session in the AXEPTool.
- Query User Interface: it defines a complete user interface for the AXMEDIS framework. It allows both simple and structured queries on metadata and granted rights of objects.

From a point of view of implementation the AXEPTool is composed by modules written in C++ and Java. The Publication, Loading, and P2P Active Selections Modules are loaded at runtime by the script engine defined in part C. Thus they are written in C++. During the execution of rules, such modules may request functionalities of the Virtual Database, Publishing and Mmonitoring Objects, and AXEPTool Monitor Modules which are written in Java and are linked in a single Java application. The interoperability between the Java part and the C++ loadable modules is obtained by means of Web Services Interfaces. For this reason we have indentified the following interfaces:

- Virtual Database exposes the AXEPTool Node Interface
- AXEPTool Monitor exposes Download Monitor Interface and Upload Monitor Interface
- Publishing and Monitoring Objects exposes the Publishing and Monitoring Interface



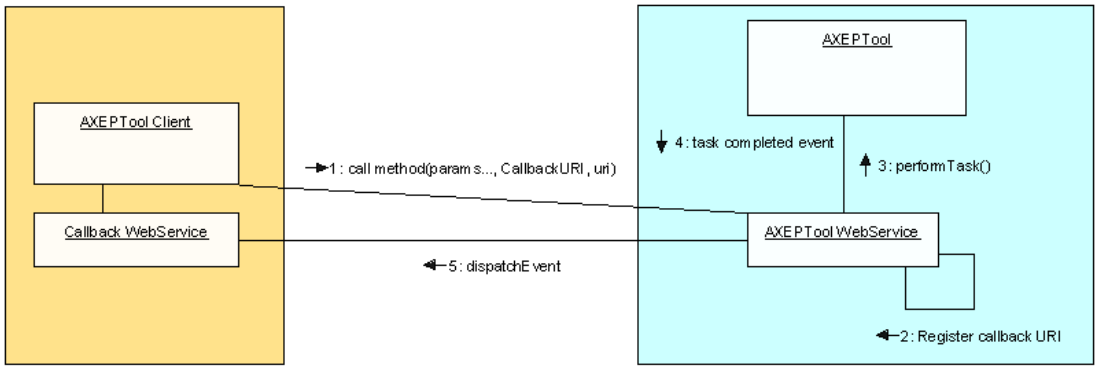
2.2 Convention on Web Services Interfaces and Callback Mechanism

For remote operation invocations of AXEPTool features and modules, Web Services are implemented and deployed.
In this section are described the conventions used in Web Service creation and a callback mechanism for asynchronous messaging through Web Services and WebService invokers.
Every AXEPTool Web Service module is well described by an Interface, so a WSDL document is provided for each of them. These interfaces describe the operations that the service can perform which can be of two types:

- Operations with synchronous feedback and results messages to invoker
- Operations with asynchronous feddback and result messages to invoker

The first type force the invoker to passive waiting for a Web Service response, so it stops its operations and wait for return messages.
With the second type of operations, which usually are time consuming operations, the invoker makes a call to request a particular operation but it not stops its tasks, simply it makes the operation call passing a "callback address" (which is a WebService), the Web Service will "send" a message to the invoker simply calling a method exposed by the callback service specified.
For example, think about a "download an AXOB from P2P" operation needed by another module. It invokes a method exposed by the AXEPTool WebService Interface. We cannot think to stop the requesting module in a passive wait until the download is complete. So we need a mechanism which allows to the invoker to be alerted only when the download operation ends, successfully or not. A callback mechanism is needed.

Next diagram shows the callback mechanism:



In the previous diagram with the name "AXEPTool Client" we refer to a generic AXMEDIS module which needs to remote request operations exposed by the AXEPTool.
The Callback WebService is a generic WebService implementing the interface for callback mechanism shown later.

The AXEPTool Client needs a particular operation that the AXEPTool can perfom. It is composed by two parts: the Client itself and a particular WebService which will respinde to callback events.
So the AXEPTool Client initiates the "conversation" for operation performance:

1. **call method(params…, CallbackURI uri)**: the AXEPTool Client calls the AXEPToolWebService operation specifying the required parameters. If a CallbackURI parameter is required it means that the operation could be called in a asynchronous way. If the client needs a synchronous operation result (it wants to wait), then the CallbackURI is passed empty (null). If the client wants an asynchronous feedback from the AXEPToolWebService, then it passes the URI of its CallbackWebService which implements the callback mechanism Interface and exposes its operations for event delivery. The AXEPToolWebService will call this operation when the operation previous requested by AXEPToolClient, finishes, successfully or not, or an event must be delivered to it asynchronously.
2. **register callback URI**: the AXEPToolWebService store the URI of the CallbackWebService to which deliver the event after operation completion.
3. **performTask:** the requested operation going to start performed by AXEPTool
4. **task completed event:** the AXEPTool informs the AXEPToolWebService about operation completion with a related event message
5. **dispatchEvent:** the AXEPToolWebService calls the operation for event delivering exposed by the CallbackWebService passing to it the event. The call is made using the CallbackURI previus passed by initiator.

The CallbackWebService implements the WSDL interface shown in the next picture:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="crs4.axmedis.ws.CallbackService"
  targetNamespace="http://crs4.com/wsdl/crs4/axmedis/ws/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:map="http:// crs4.com/mapping/"
  xmlns:ns0="http:// crs4.com/xsd/SchemaTypes/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http:// crs4.com/wsdl/crs4/axmedis/ws/">
  <types>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http:// crs4.com/wsdl/crs4/axmedis/ws/"
      xmlns:map="http:// crs4.com/mapping/"
      xmlns:tns="http:// crs4.com/wsdl/crs4/axmedis/ws/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:complexType name="CallbackEvent">
        <xsd:annotation>
          <xsd:appinfo>
            <map:java-type name="crs4.axmedis.ws.CallbackEvent"/>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence/>
      </xsd:complexType>
    </xsd:schema>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http:// crs4.com/xsd/SchemaTypes/"
      xmlns:tns="http:// crs4.com/xsd/SchemaTypes/"
      xmlns:xns4="http:// crs4.com/wsdl/crs4/axmedis/ws/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import namespace="http:// crs4.com/wsdl/crs4/axmedis/ws/"/>
```

```
        <xsd:element maxOccurs="1" minOccurs="0" name="event"
          type="xns4:CallbackEvent"/>
      </xsd:schema>
    </types>
    <message name="CallbackService_receiveEvent_Response_Soap"/>
    <message name="CallbackService_receiveEvent__Request_Soap">
      <part element="ns0:event" name="event"/>
    </message>
    <portType name="CallbackService">
      <operation name="receiveEvent" parameterOrder="event">
        <input message="tns:CallbackService_receiveEvent__Request_Soap"/>
        <output message="tns:CallbackService_receiveEvent_Response_Soap"/>
      </operation>
    </portType>
    <binding name="CallbackService" type="tns:CallbackService">
      <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="receiveEvent">
        <map:java-operation name="receiveEvent"
          signature="KExjcnM0L2F4bWVkaXMvd3MvQ2FsbGJhY2tFdmVudDspVg=="/>
        <soap:operation
          soapAction="http://
crs4.com/wsdl/crs4/axmedis/ws/CallbackService#receiveEvent?KExjcnM0L2F4bWVkaXMvd3Mv
Q2FsbGJhY2tFdmVudDspVg=="
          style="document"/>
        <input>
          <soap:body parts="event" use="literal"/>
        </input>
        <output>
          <soap:body use="literal"/>
        </output>
      </operation>
    </binding>
    <service name="CallbackService">
      <port binding="tns:CallbackService" name="CallbackService">
        <soap:address location="http://server:6060/CallbackService/"/>
      </port>
    </service>
</definitions>
```

2.3 Publication Module of AXEPTool (WP4.4.5: CRS4, WP5.5.3: CRS4)

Module Profile		
Publication Module of AXEPTool		
Executable or Library(Support)	Library	
Single Thread or Multithread	--	
Language of Development	C++/Javascript/XML	
Responsible Name	Carboni	
Responsible Partner	CRS4	
Status (proposed/approved)	proposed	
Platforms supported	Win32, Linux, MacOS	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
	AXCP Rule Engine	
	Publishing and Monitoring Objects AXDB Protection Tool engine	
	AXDBOUT	
File Formats Used	Shared with	File format name or reference to a section
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

The Publication Module is loaded whenever the Script Engine executes a Publication Rules. It moves Axmedis objects from the AXMEDIS Data Base to the AXOUTDB using the related Database Manager.
The Publication Module contains the following sub-components:

- JS_Publisher
- Publisher Component
- Publication Rules
- AXDB Client
- AXDBOUT Client

2.3.1 Publisher Component

- The **Publisher** is the class loaded by the script engine and wrapped by a JS_Publisher Javascript class, so that its methods can be invoked in scripts.

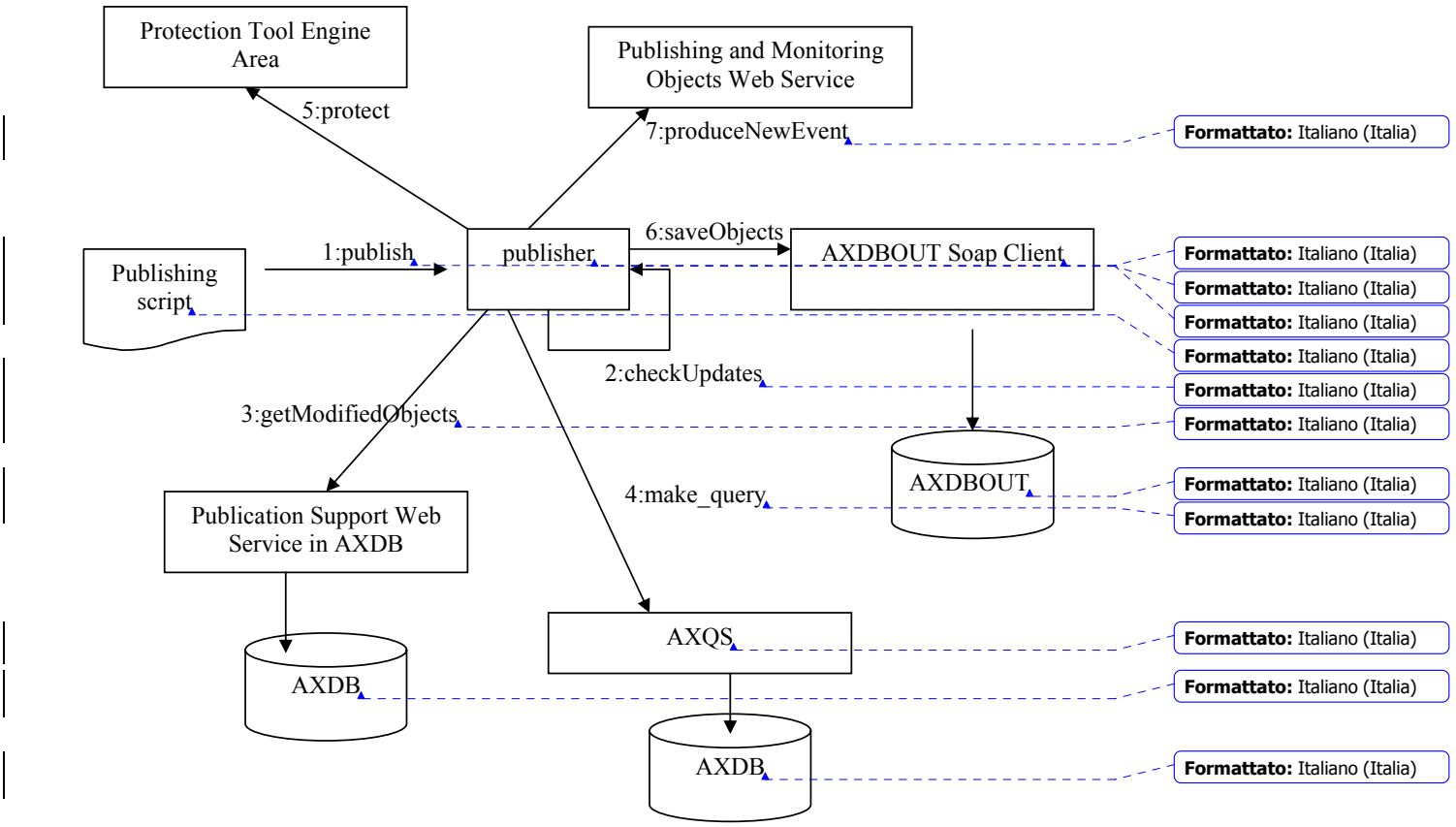
  - Publish() publishes the actualized selection
  - Unpublish() un-publishes the actualized selection
  - publishedObjects() returns the ID of objects published by the current instance of the Publisher
- The **publication rule** is not a class, it is an instance of the AXCP Rule class.

2.3.2 Collaboration Diagram: Publication of Objects in a super-peer

A super-peer is a peer able to solve queries from other peers. See the Virtual Database Area for details.
When a publication rule is scheduled for execution the script could invoke he method publish. The sequence of actions are the followings:

1. *publish*: the publisher instance is commanded to publish the objects in the selection of the rule. This command usually comes from the script.
2. *checkUpdates*: the publisher checks if new objects/versions belonging to the selection must be published
3. *getModifiedObjects*: the publisher asks the publication support which objects have been added or modified from the last publication date for this rule
4. *make_query*: the publisher queries the AXDB in order to establish which new objects/version belongs to the actual selection
5. *protect*: the publisher invokes the Protection Tool Engine in order to protect unprotected objects before the publication.
6. *saveObjects*: the AXDBOUT client saves new objects/versions on the AXDBOUT
7. *producePublicationEvent*: the publisher asks the Publishing and Monitoring Objects module to send a message on the P2P network regarding the availability of new objects/versions

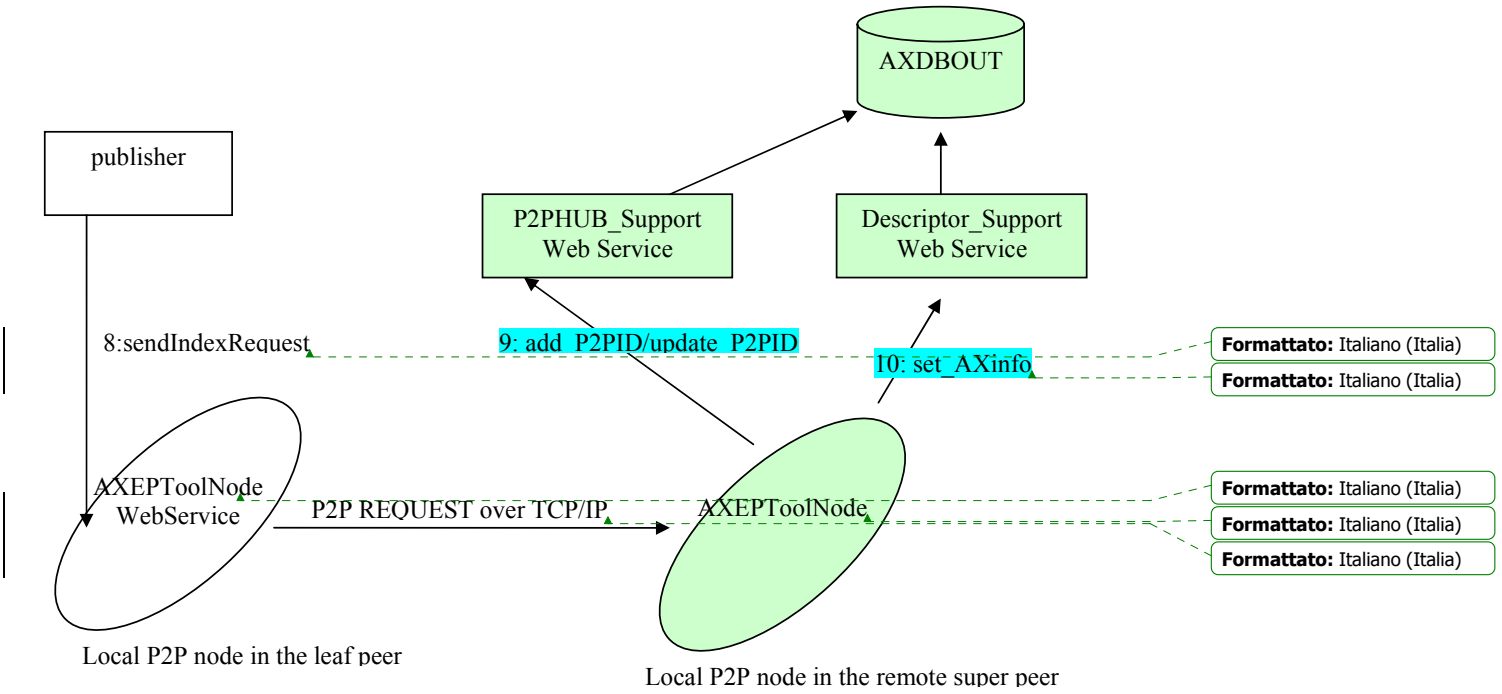
This sequence is repeated periodically as publishing rules have a periodicity in order to re-publish new versions/objects as soon as they are ready in the AXDB.



2.3.3 Collaboration Diagram: Publishing a selection of Objects in a leaf peer

A leaf peer is a node of the P2P which does not solve remote queries. It sends its metadata to a super-peer (also called hub) which is commited to solve the queries for all its connected nodes. See virtual database area for details. The sequence of actions is the same of super-peers up to step 7 included. The further steps are:

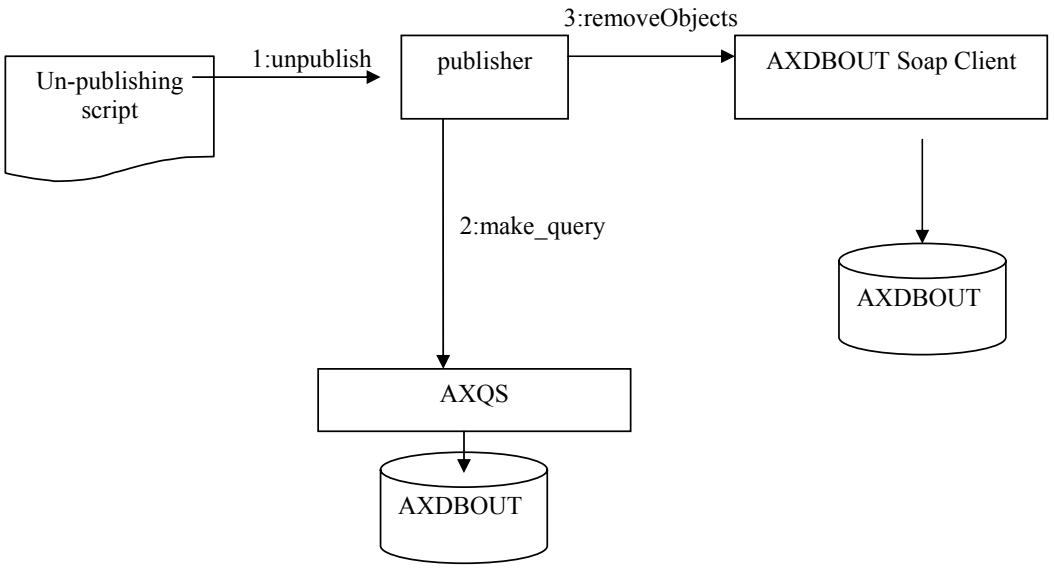
8. sendIndexRequest: the publisher connects to the AXEPToolNode and asks to produce a indexer request to the super-peer.
9. add_P2PID/update_P2PID: the super-peer AXEPToolNode connects to the P2Phub_Support Web Service in the super-peer and add/update items
10. set_AXinfo: the AXEPToolNode asks the Descriptor_Support Web Service to index the objects in the leaf node



2.3.4 Collaboration Diagram: Un-publishing a selection of Objects

A publication rule can be executed to unpublish objects. In this case the sequence of actions is the following:

1. unpublish: the publisher class is commanded to unpublish the objects in the selection of the rule
2. make_query: the publisher obtains the IDs of objects to be unpublished from the AXQS of the AXDBOUT
3. removeObjects: the AXDBOUT client removes the objects to be removed from the AXDBOUT



2.3.5 Publication Rules (CRS4)

A publication rule is a AXCP rule whose script is aimed at publishing objects. A AXCP Rule contains the following information:

- WHO is the author and the owner of the rule
- WHEN the publication must occur
- WHICH objects must be published

For the above reasons the publication rule contains:
An header, a schedule and a definition.
See the Part C for a complete description of AXCP Rules.

2.3.6 How to Modify the AXINFO during the publication

After the execution of a publication rule the rule creator can insert in the script other commands to modify the AXINFO of published objects. This could be done using the method publishedObjects() of Publisher. This method returns the selection of object published by this rule. The JS_AXOM wrapper can be used to modify AXINFO data. As example, the creatorURL of an object could be removed for privacy reasons. For details of AXOM see part A.

2.3.6.1 Example of Publication Rule

The rule below forces the Publication Engine to publish any object whose author is equal to “Smith” and will become active from 12:00 of 24/12/2004 to 01/01/2005 and it is scheduled every 30minutes in order to publish new versions/objects.

```
<Rule>
        <Header>
                <Rule_Name> Foo </Rule_Name>
                <AXRID> 0010001 </AXRID>
                <Rule_Version> 1.0 </Rule_Version>
                <Rule_Type> AXCP </Rule_Type>
                <Software_Name> Axmedis Rule Editor </Software_Name>
                <Version_of_software> 2.0 </Version_of_software>
                <Date_of_production> 24.12.2004 </Date_of_production>
                <Time_of_production> 12:00 am </Time_of_production>
                <Author> Dave Charcoal </Author>
                <Affiliation> CRS4 </Affiliation>
                <URL> http://www.crs4.it </URL>
                <Comment> This rule publish only audio file </Comment>
                <Last_Modifications> 25.12.2004 </Last_Modifications>
                <Terminal_ID> </Terminal_ID>
                <Cost> </Cost>
                <Work_Item_ID> 0000 </Work_Item_ID>
        </Header>
        <Schedule>
                <Run>
                        <Date> 24.12.2004 <Date>
                        <Time> 12:00 pm </Time>
                        <Periodicity> 30m </Periodicity>
                        <Expiration_Date> 01.01.2005 </Expiration_Date>
                        <Expiration_Time> 12:00 pm </Expiration_Time>
                 </Run>
                <Status> Active </Status>
        </Schedule>
        <Definition>
                <AXCP_Rule>
                        <Arguments>
                                <selection name="TEST" timestamp="2005-01-20T18:20:46.275+01:00">

                                        ...SEE SELECTION BELOW...
                                 </selection>
                                                        ……
                         <Parameter>  ……</ Parameter >
                        </ Arguments>
                        <Preconditions>
                         <Plug_In_Name> Publisher </Plug_In_Name >
                         <Version> 1.0 </Version>
                        </Preconditions>
                        <Preconditions>
                        …….
                        </Preconditions>
                        <Rule_Body>
                            …
                        </Rule_Body>
                </AXCP_Rule>
        </Definition>
</Rule>
```

The selection is defined as …

```
<selection xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="D:\Documents and
Settings\davide\Desktop\axmedis\Selection-v1-3.xsd" name="" timestamp="">
    <query>
        <source>
            <location>AXDB</location>
        </source>
        <AXinfoQuery>
            <querycondition>
                <nesting>
                    <test>
                        <field>author</field>
                        <operator>EQ</operator>
                        <value>Smith</value>
                    </test>
                </nesting>
            </querycondition>
        </AXinfoQuery>
    </query>
</selection>
```

2.3.7 Publication Engine WF Interface

The Publication Rules are stored in the AXCP Rule Storage, so they can be activated, inserted, removed and listed using commands and reports already defined and identified in Part C.

2.4 Loading Module of AXEPTool (WP4.4.4: CRS4, WP5.5.2: CRS4)

Module Profile		
Loading Module of AXEPTool		
Executable or Library(Support)	Library	
Single Thread or Multithread	-	
Language of Development	C++/XML/Javascript	
Responsible Name	Carboni	
Responsible Partner	CRS4	
Status (proposed/approved)		
Platforms supported	Win32, Linux, MacOS	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
	AXDBIN	
	AXCP Rule Engine	
File Formats Used	Shared with	File format name or reference to a section
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

The Loading Module moves Axmedis objects from the AXINDB to the AXDB using the related Database Manager. The process is guided by rules handled by AXCP Rule Engine. The Loading Module provides the Java Script Interface and the C++ runtime library.
The Loading Module interacts with the following components:

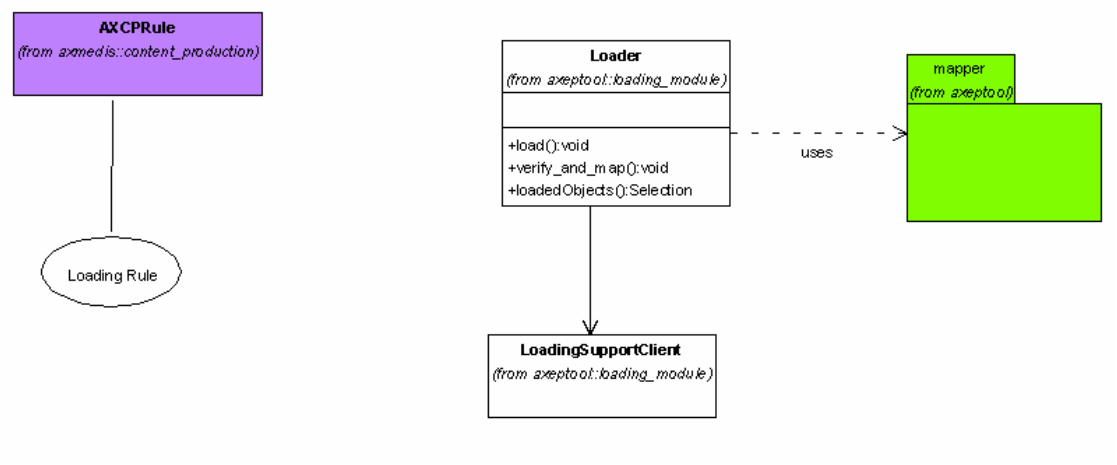
- AXCP Rule Storage
- AXCP Rule Engine
- JS_Loader script for the AXCP Rule Engine + runtime library
- AXDB Client of the Loading Module

- AXDBIN Client of the Loading Module

2.4.1 Class Diagram of Loading Module of the AXEPTool

The main class is the **loader**. It provides the proper Java Script interface and the C++ implementation classes for AXOBs movements.

2.4.1.1 Loader



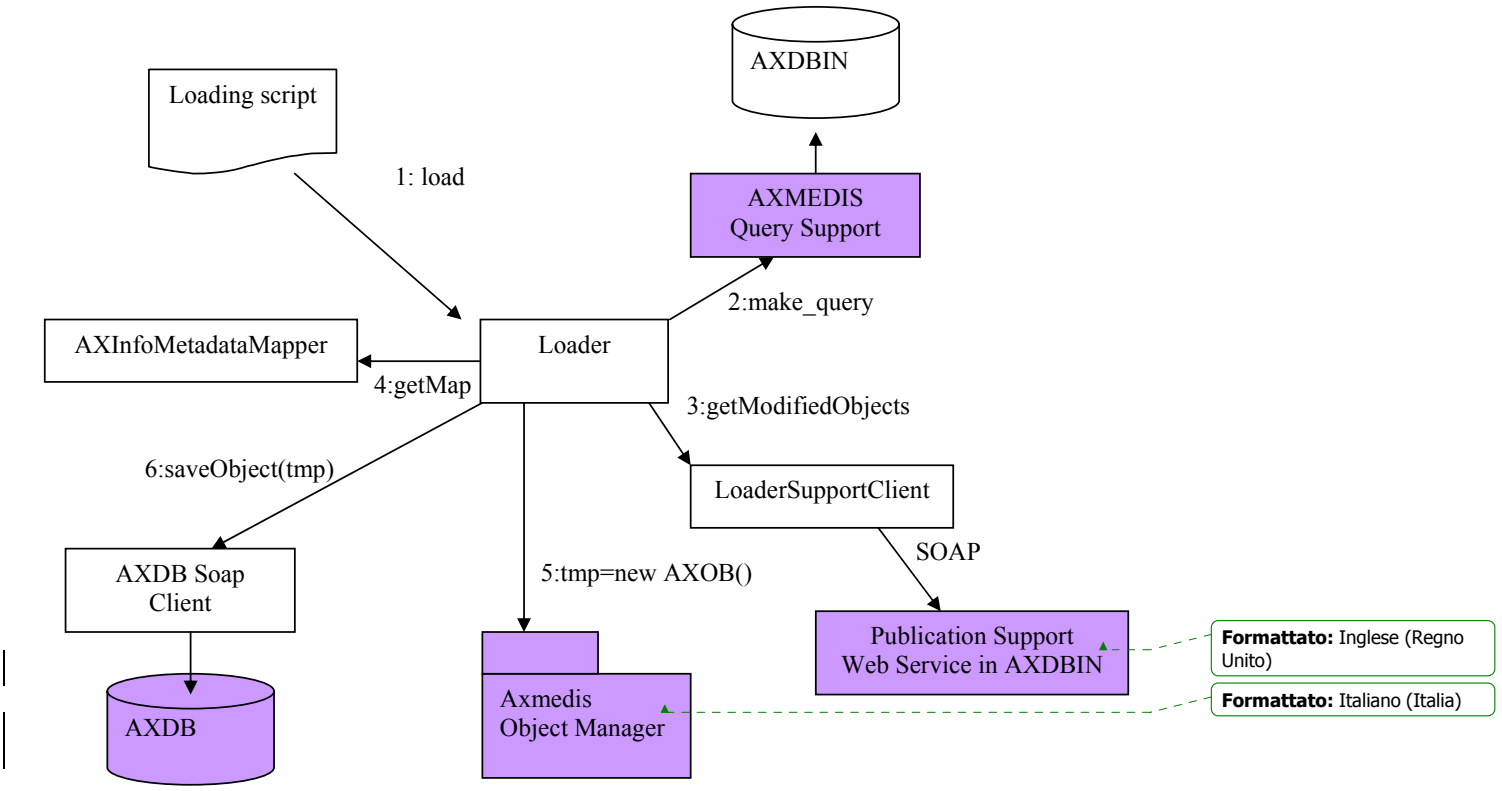
- The **Loading Rule** is not a class, it is an instance of the AXCP Rule class.
- The **Loader** is the class loaded by the script engine and wrapped by a JS_Loader Javascript class, so that its methods can be invoked in scripts.
    - load(): loads the actualized selection
    - loadedObjects(): returns the ID of objects loaded by the current instance of the Loader

- The **AXDBSoapClient** is instantiated whenever a connection to the AXDB or AXDBIN is needed by the loading module.
- The **mapper** is the package used to perform the mapping between customer metadata.

2.4.2 Loading a selection of objects

When a rule is scheduled for execution the script invokes the method load.
The Rule states which selection of objects have to be involved in the process. The sequence of steps are:

1. *load*: the AXCP Rule Engine activates the script invoking the load method.
2. *make_query*: the loader queries the AX Query Support in the AXDBIN to identify which new objects/version belongs to the actual selection.
3. *getModifiedObjects*: the loaded uses the LoadingSupportClient to identify which objects/versions have been committed since the last period
4. *getMap*: the loader invokes the AXInfoMetadataMapper to load the appropriate metadata mapping table for every object
5. *tmp=new AXOM:* A new temporary object is created. It contains the original object to be loaded and adds some local metadata information.
6. *saveObject(tmp):* the temporary object is loaded into the AXDB.



2.4.3 Loading new versions of AXMEDIS Objects

In case of new version of AXOBs in AXDBIN is necessary a new process of loading. From the point of view of loading, the scenario is the same of that previously explained. In this case is required a particular rule able to check periodically updating in AXDBIN. Distinguishing information are related to *schedule* part of the rule:

```
<Schedule>
        <Run>
                <Date> 24.12.2004 <Date>
                <Time> 12:00 pm </Time>
                <Periodicity> 30m </Periodicity>
                <Expiration_Date> 01.01.2005 </Expiration_Date>
                <Expiration_Time> 12:00 pm </Expiration_Time>
         </Run>
        <Status> Active </Status>
</Schedule>
```

2.4.4 Loading Rules/Selections Format

Loading rules are simply AXCP rules with a script aimed at loading objects (see Part C of specification).

2.4.5 Example of Loading rule

```
<?xml version="1.0" encoding="UTF-8"?>
<Rule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="comp-format-rules.xsd">
    <Header>
        <Rule_Name>Foo</Rule_Name>
        <AXRID>0000000000</AXRID>
        <Rule_Version>1.5</Rule_Version>
        <Rule_Type>AXCP</Rule_Type>
        <Software_Name>Bar</Software_Name>
        <Version_of_software>1.0</Version_of_software>
        <Date_of_production>none</Date_of_production>
        <Author>Pat Porter</Author>
        <Affiliation>FBM</Affiliation>
        <URL>http://foo-bar.com</URL>
        <Comment>none</Comment>
        <Last_Modifications>01/01/2005</Last_Modifications>
        <Terminal_ID>00000</Terminal_ID>
        <Cost>0</Cost>
        <Work_Item_ID>00000</Work_Item_ID>
    </Header>
    <Schedule>
        <Run>
            <Date>21/02/2005</Date>
            <Time>18:00</Time>
        </Run>
        <Status>Inactive</Status>
    </Schedule>
    <Definition>
        <AXCP_Rule>
            <Arguments>
                <Selection>
                    <query>
                        <source>
                            <location>AXINDB</location>
                        </source>
                        <AXinfoQuery>
                            <querycondition>
                                <nesting>
                                    <test>
                                        <field>author</field>
                                        <operator>EQ</operator>
                                        <value>Smith</value>
                                    </test>
                                </nesting>
                            </querycondition>
                        </AXinfoQuery>
                    </query>
                </Selection>
                <Parameter/>
            </Arguments>
            <Rule_Body>
                <JS_Script> file://loading_JS_script_001.js</JS_Script>
            </Rule_Body>
        </AXCP_Rule>
    </Definition>
</Rule>
```

2.4.6 Loading Engine WF Interface

The Loading Rules are stored in the AXCP Rule Storage, so they can be activated, inserted, removed and listed using workflow commands and reports already defined and identified in Part C.

2.4.7 AXInfo Metadata Mapper (CRS4, DSI, EXITECH)

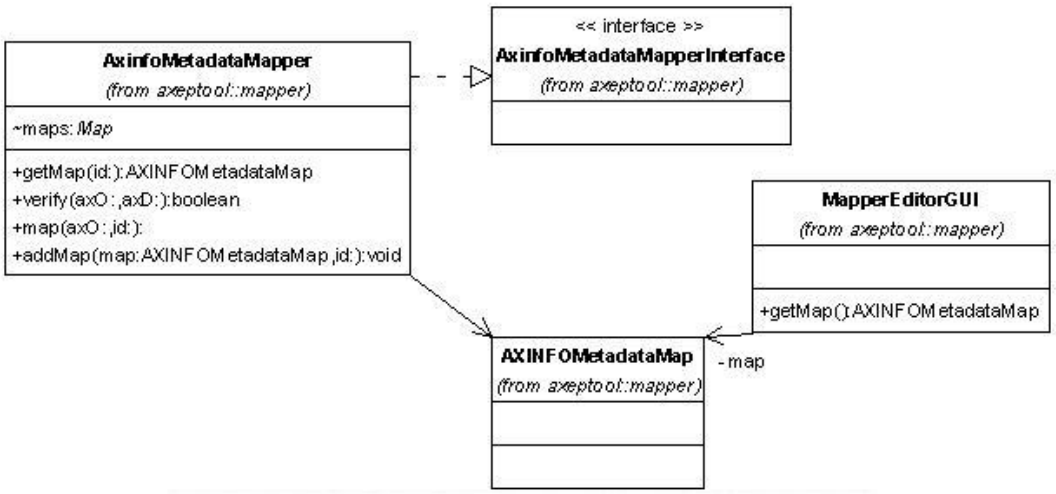
Objects becoming from different factories may not have the same set of metadata. For this reason is necessary a process of "translation" from one set to the other.

Specification of AXInfo Metadata Mapper starts from Metadata description reported in document "Framework and Tools Specifications (General Aspects and Model)" (part A).
Metadata information related to an object is split among various Descriptors.
These Descriptors can contain:

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

The AXInfo contains information to manage the object in its entire life-cycle. Metadata are mainly composed of mandatory fields (common to every object) and optional ones; optional fields can depend on the specific content creator. This approach is intended to give to the factories certain kind of "flexibility" in using metadata for objects.
AXInfo Metadata Mapper is the module of AXEPTool dedicated to map metadata between objects becoming from different origins. Factory "X" interested in objects owned by Factory "Y" may be forced to convert the set of custom metadata of incoming objects. Mapping process starts before saving in AXDB. In fact while AXOB are contained into AXDBIN they are not effectively acquired.

2.4.8 Class Diagram



Axmedis Metadata Mapper involves the following classes:
AXINFOMetadataMapperInterface, the interface of the Mapper.
AXINFOMetadataMapper, the class implementing the logic for mapping
MapperEditoGUI, the graphical user interface used by the local operator to creates the mapping file if no-mapping is available.
AXINFOMetadataMap, represents the map for the translation.

2.4.9 Mapping table

Mapping table

To map means to associate metadata among origin and destination formats. Let's consider the case in which Factory F2 gets an AXOB from Factory F1, and custom F2 metadata are different from custom F1 Metadata. Factory F2 needs the conversion table, which consists of an XML file containing a list of associations.

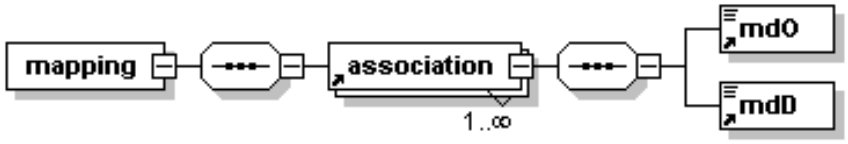
<association>
<mdO>…</mdO>
<mdD>…</mdD>
</association>

Every time F2 gets data from F1 the same table may be applied. Getting objects from Factory F3, Factory F2 needs a new conversion table, which may be applied every time F2 download from F3.
In this way Facrtory F2 creates a list of several mapping tables, concerning partners related with.
Mapping table contains IDs and a reference to mapping files:

ID	Mapping File
map_1	map_1.xml
map_2	map_2.xml
…	…

2.4.10 Mapping XML formalization

Mapping structure can be formalized by the following XML Schema.

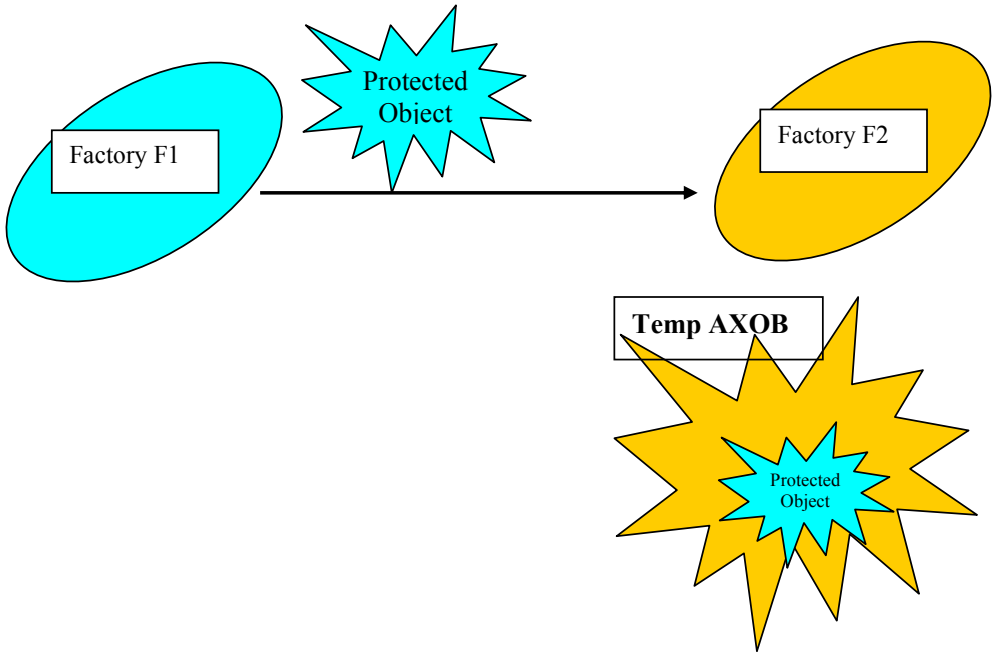


The same formalization is also provided in plain text.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:element name="association">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="mdO"/>
                <xs:element ref="mdD"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="mapping">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="association" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="mdD" type="xs:string"/>
    <xs:element name="mdO" type="xs:string"/>
</xs:schema>
```

2.4.11 Mapping through a temporary AXOB

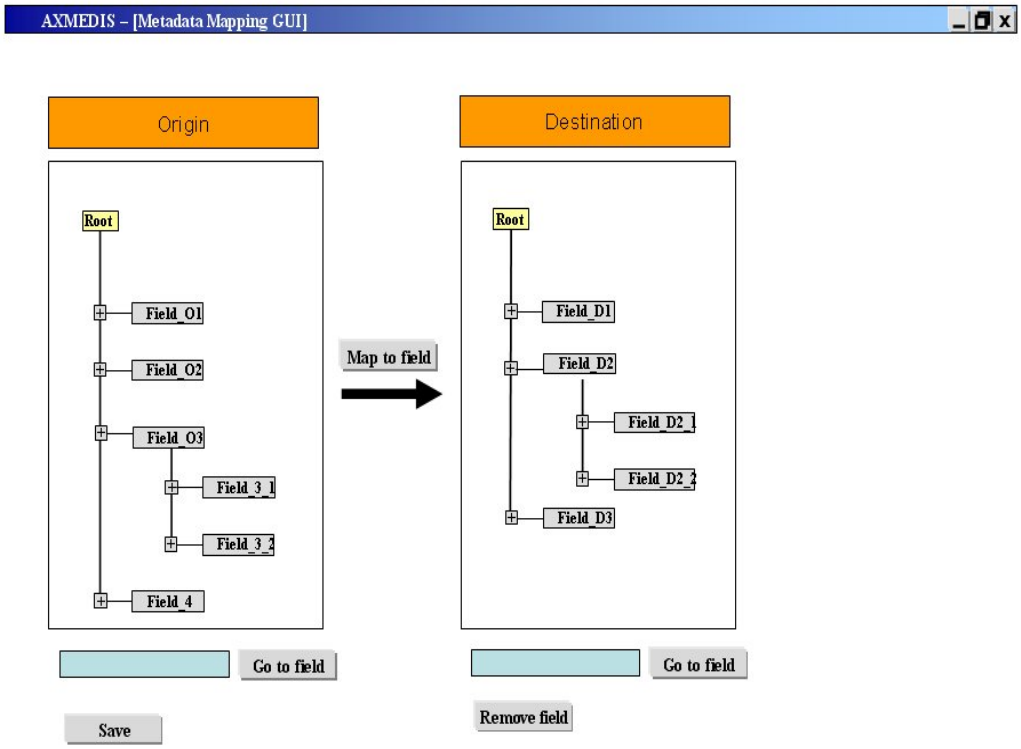
Modification of metadata (and so of an object) relies to object's rights. Not always the factory which gets objects is allowed to perform modifications. On the other hand, mapping is necessary every time there is no matching of metadata. In this perspective, the factory which gets the object (F2) stores the objects in its database but creates a temporary AXOB (Temp AXOB) which contains the original (see following figure). New Temp AXOB defines custom metadata of Factory F2 applying the mapping from metadata of the original Protected Object. In this way the object is compatible with searching features of Factory F2.



The new object needs to be remarked as temporary. So is used a particular prefix on the AXInfo/Creator/AXCID element. To be handed out again, the object needs to be extracted from the temporary one.

2.4.12 Mapper Editor GUI

Mapper editor GUI is the interface used to creates the map for incoming metadata translation. The GUI allows the user to decide which origin fields have to be converted in destination fields. At the end of the process will be produced the mapping file. The GUI is integrated on the AXCP Editor.


AXMEDIS – [Metadata Mapping GUI]
Origin
Destination
Root
Field_O1
Field_O2
Field_O3
Field_3_1
Field_3_2
Field_4
Map to field
Root
Field_D1
Field_D2
Field_D2_1
Field_D2_2
Field_D3
Go to field
Go to field
Save
Remove field

2.5 AXEPTool AXMEDIS Databases (AXDBIN, AXDBOUT, AXDB) (CRS4, EXITECH)

The AXEPTool makes use of three databases: the AXDB, the AXDBIN, and the AXDBOUT.

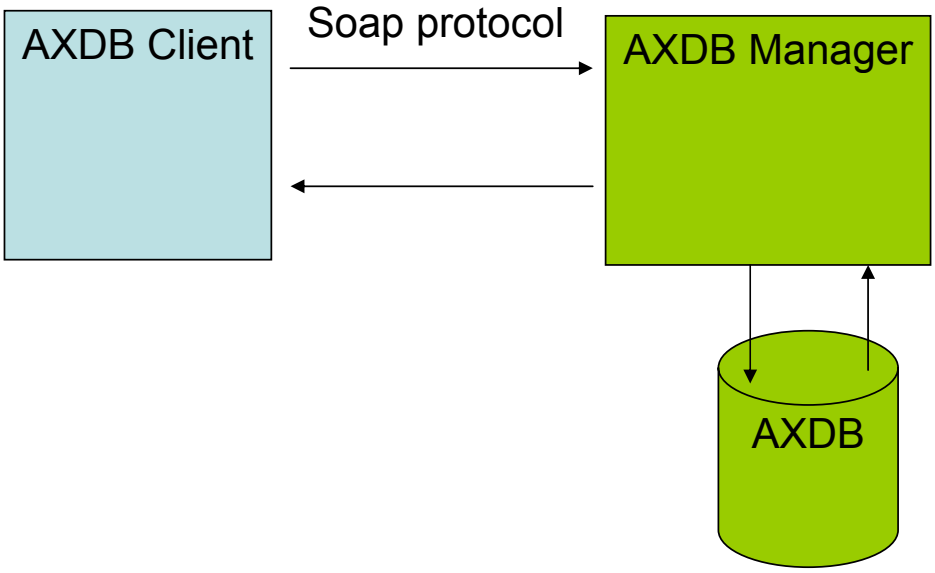
All these database are instances of the AXMEDIS Database described in the part E.

- The AXDB is deployed in the factory as main container of AXMEDIS Objects
- The AXDBIN is deployed in the AXEPTool and contains the AXMEDIS objects downloaded from other peers
- The AXDBOUT is deployed in the AXEPTool and contains the AXMEDIS objects ready to be uploaded to other peers.

2.5.1 AXDB Client of AXEPTool (CRS4, EXITECH)

This component is a client able to connect to the AXDB Manager Web Services interface which is defined in (part E). This component is loaded and linked as a binary library in the code of the Publication Module of the AXEPTool and exposes an API directly mapped onto the Web Services interface needed by the AXDB Manager.
The AXDB Client can connect and interact with:

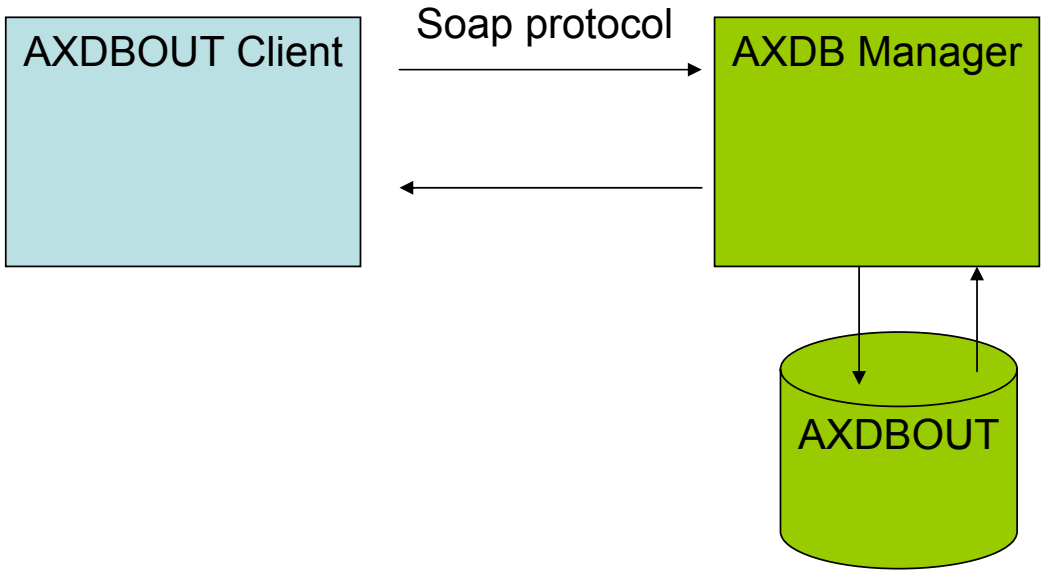
- Loader WebService
- Saver/Indexer Web Service
- P2PHub Web Service
- Publication Support Web Service
- Descriptor Support Web Service



2.5.2 AXDBOUT Client of the AXEPTool (CRS4, EXITECH)

This component is an instance of **AXDB Client** connected to the AXDBOUT Manager Web Services interface which is defined in (part E). This component is loaded and linked as a binary library in the code of

the Publication module of the AXEPTool and exposes an API directly mapped onto the Web Services interface needed by the AXDB Manager. It is necessary to include a configuration file in the Publication Engine stating the IP:PORT values to connect to the AXDBOUT Web Service.



2.5.3 AXDBIN Client of the Loading Engine

The AXDBIN Client is a client which uses the Soap protocol to load/save and query on the AXDBIN. It is mainly an instance of **AXDB Client.**

2.6 AXMEDIS Query GUI (CRS4, EXITECH, ILABS, ACIT)

Module Profile		
AXMEDIS Query GUI		
Executable or Library(Support)	Library	
Single Thread or Multithread	Not applicable	
Language of Development	XUL or Javascript or C++ depending on deployment	
Responsible Name	D.Carboni	
Responsible Partner	CRS4	
Status (proposed/approved)	proposed	
Platforms supported	All	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
	Axmedis Query Support	
File Formats Used	Shared with	File format name or reference to a section
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

This section describes the unified Query User Interface (QUI) for the AXMEDIS platform. By means of the QUI will be possible to query the AXDB, the local CMS, and the P2P Network.

The underlying query model is defined in Part E and it is based on a XML specification. In part E are also specified the functions for saving/loading queries and selections in the database.

The QUI is composed by a number of components:

- the Flat QUI is a form-based UI which allows common “fill-the-form and send” operations
- the Tree QUI is a tree-based UI which allows the composition of queries in a structured form with arbitrary complexity

2.6.1 Flat Query UI

The flat QUI is generated on the fly according to a configuration file that can be edited by an expert operator in order to modify the number, the position, and the set of data editable in the form.
It is composed by two tabs: the Dublin Core tab and the Available Rights Tab.

The elements in a tab are logically related with the AND operator
The two tabs are related with the AND operator, so the logical structure of a Flat query is the following:

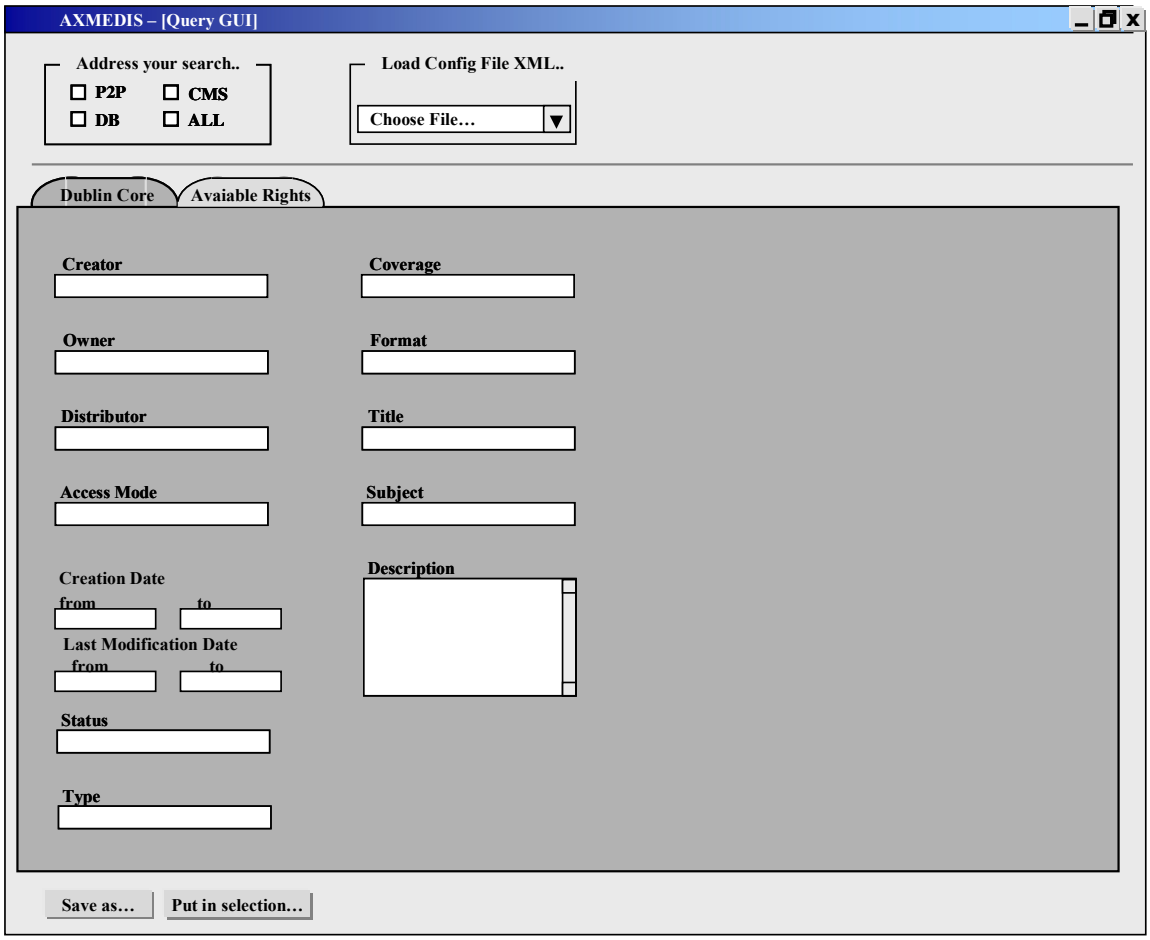
DC_Query = (field1 **OP1** value1) **AND** ... (fieldN **OPN** valueN)

AR_Query = (field1 **OP1** value1) **AND** ... (fieldN **OPN** valueN)

Flat_Query = AR_Query **AND** DC_Query

2.6.1.1 Dublin Core Tab

The Dublin Core tab contains a subset of all possible metadata described in the DC specification. To add new metadata the user must edit a new config file and load it during the interaction.



2.6.1.2 The Configuration File

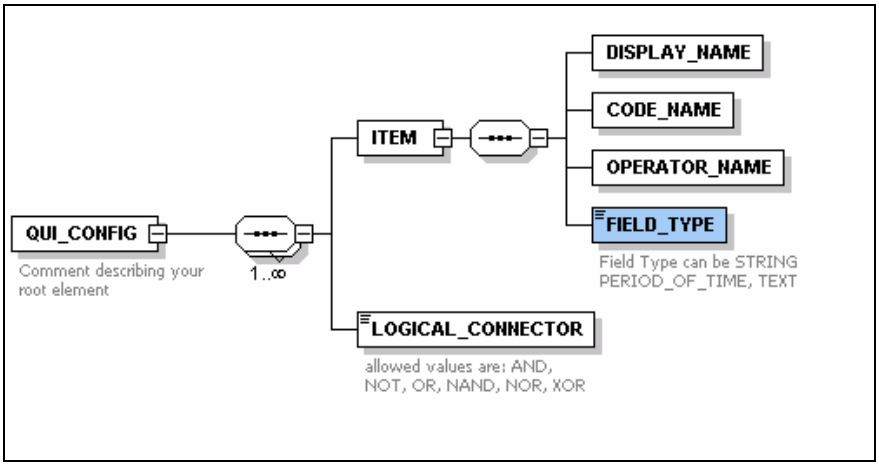
A Configuration File is showed in the example below which generates a Flat QUI showed in the figures of this section.

A default configuration covering all main metadata will be provided with the implementation of the GUI.

2.6.1.3 Example of Configuration File Used to generate on the fly the Flat QUI

```
<?xml version="1.0" encoding="UTF-8"?>
<QUI_CONFIG xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="E:\axmedis\part
F\v2-9\QUI_CONFIG.xsd">
    <ITEM>
        <DISPLAY_NAME>CREATOR</DISPLAY_NAME>
        <CODE_NAME>/foo/bar/.../creator</CODE_NAME>
        <OPERATOR_NAME>CONTAINS</OPERATOR_NAME>
        <FIELD_TYPE>STRING</FIELD_TYPE>
    </ITEM>
    <LOGICAL_CONNECTOR>AND</LOGICAL_CONNECTOR>
    <ITEM>
        <DISPLAY_NAME>OWNER</DISPLAY_NAME>
        <CODE_NAME>/foo/bar/.../OWNER</CODE_NAME>
        <OPERATOR_NAME/>
        <FIELD_TYPE/>
    </ITEM>
    <LOGICAL_CONNECTOR>AND</LOGICAL_CONNECTOR>

[…SNIP… MORE ELEMENTS HERE]


    <ITEM>
        <DISPLAY_NAME>LAST MODIFICATION DATE</DISPLAY_NAME>
        <CODE_NAME>/foo/bar/.../modificationDate</CODE_NAME>
        <OPERATOR_NAME>IN_BETWEEN</OPERATOR_NAME>
        <FIELD_TYPE>PERIOD_OF_TIME</FIELD_TYPE>
    </ITEM>
    <LOGICAL_CONNECTOR>AND</LOGICAL_CONNECTOR>
    <ITEM>
        <DISPLAY_NAME>DESCRIPTION</DISPLAY_NAME>
        <CODE_NAME>/foo/bar/.../description</CODE_NAME>
        <OPERATOR_NAME>CONTAINS</OPERATOR_NAME>
        <FIELD_TYPE>STRING</FIELD_TYPE>
    </ITEM>
</QUI_CONFIG>
```

2.6.1.4 Configuration File XML Schema



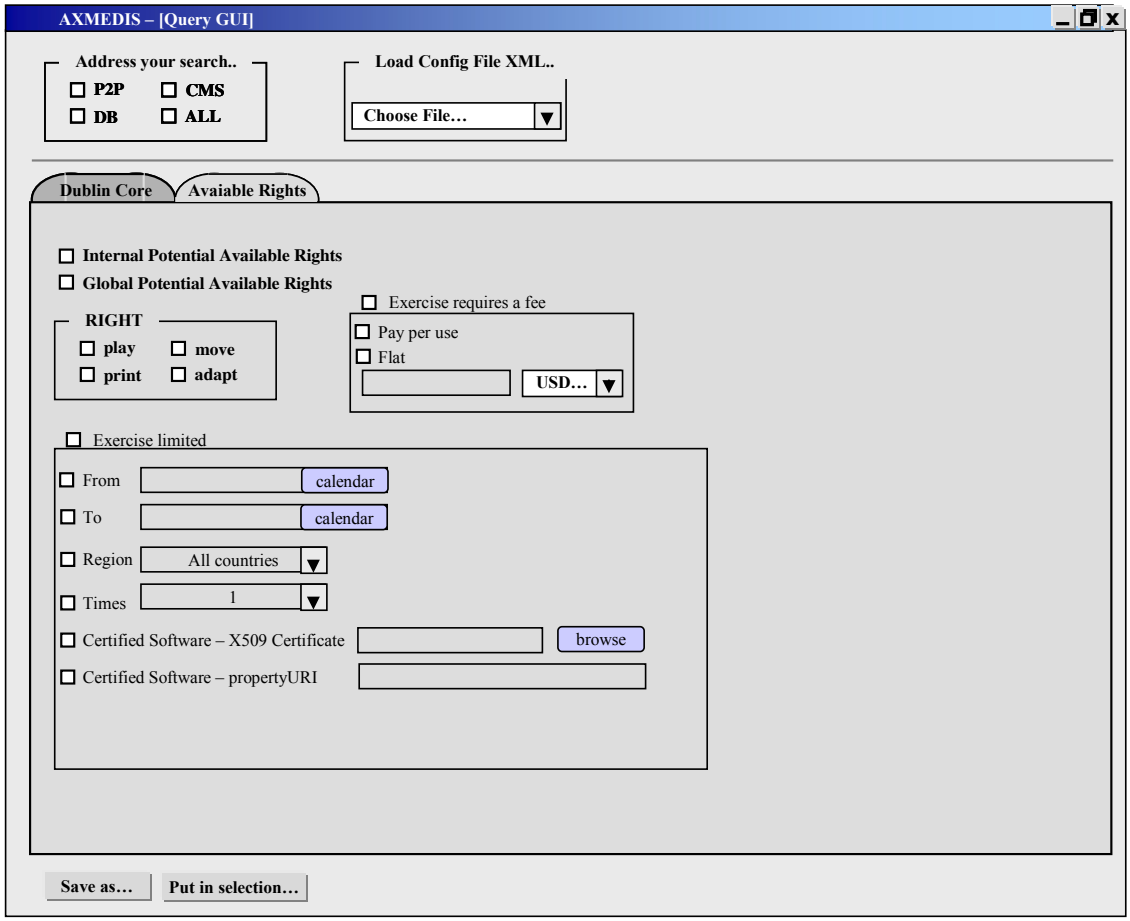
The XML schema of the configuration file in text format is showed below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by c (CRS4 ICT/NDA) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <xs:element name="QUI_CONFIG">
        <xs:annotation>
            <xs:documentation>Comment describing your root element</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
                <xs:element name="ITEM">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="DISPLAY_NAME"/>
                            <xs:element name="CODE_NAME"/>
                            <xs:element name="OPERATOR_NAME"/>
                            <xs:element name="FIELD_TYPE">
                                <xs:annotation>
                                    <xs:documentation>Field Type can be STRING  PERIOD_OF_TIME,
TEXT</xs:documentation>
                                </xs:annotation>
                                <xs:simpleType>
                                    <xs:restriction base="xs:string">
                                        <xs:enumeration value="STRING"/>
                                        <xs:enumeration value="PERIOD_OF_TIME"/>
                                        <xs:enumeration value="TEXT"/>
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="LOGICAL_CONNECTOR">
                    <xs:annotation>
                        <xs:documentation>allowed values are: AND, NOT, OR, NAND, NOR, XOR</xs:documentation>
                    </xs:annotation>
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="AND"/>
                            <xs:enumeration value="OR"/>
                            <xs:enumeration value="NOT"/>
```

```
                        <xs:enumeration value="NAND"/>
                        <xs:enumeration value="NOR"/>
                        <xs:enumeration value="XOR"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

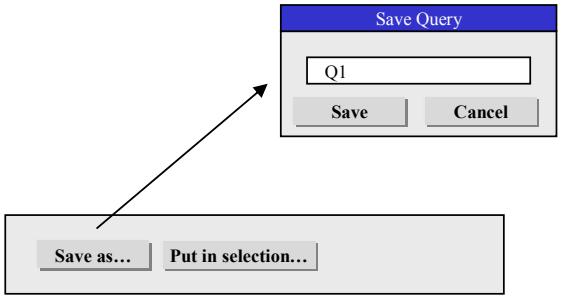
2.6.1.5 Available Rights tab

The Available Rights tab contains a set of data relevant for querying licenses according to the format of REL.



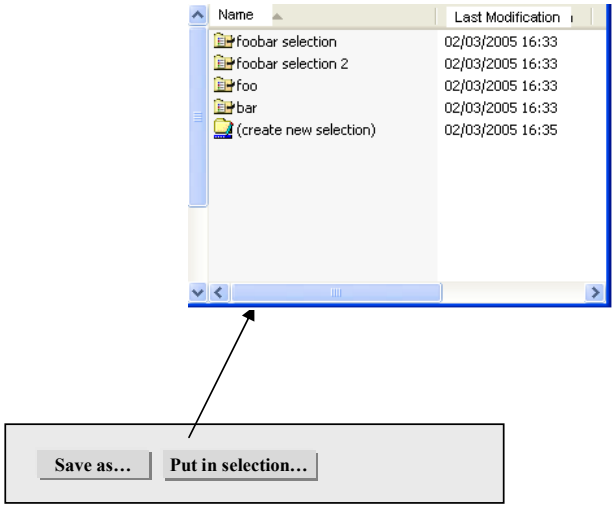
2.6.1.6 Saving a Query

A query can be saved using the button “Save as...” in this case it can be named and its name used as reference in the Tree QUI.



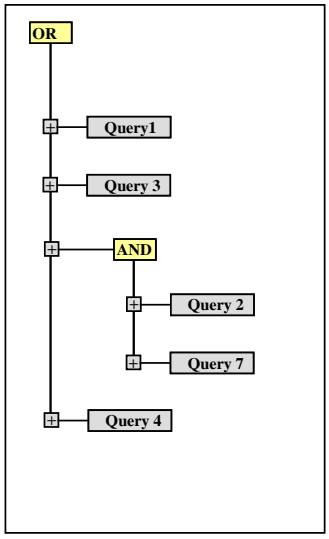
2.6.1.7 Putting a Query in a Selection

A query can be inserted in an existing selection or in a new selection created on the fly.



2.6.2 Tree Query User Interface

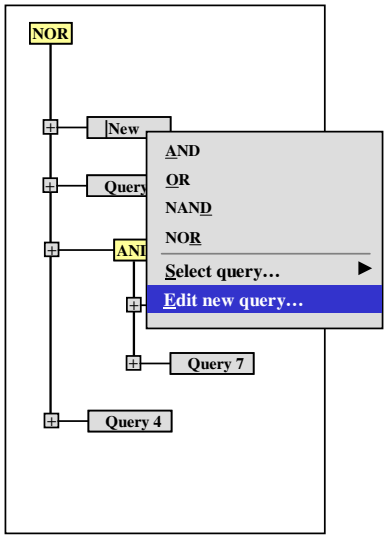
The Tree Query User Interface allows the user to build composite queries using logical operators. The tree structure allows to exploit the full complexity of the query model defined in Part E.



In the figure above the components queries Query1, Query2, ..., are queries previously saved in the DB. The query in the example is equal to:

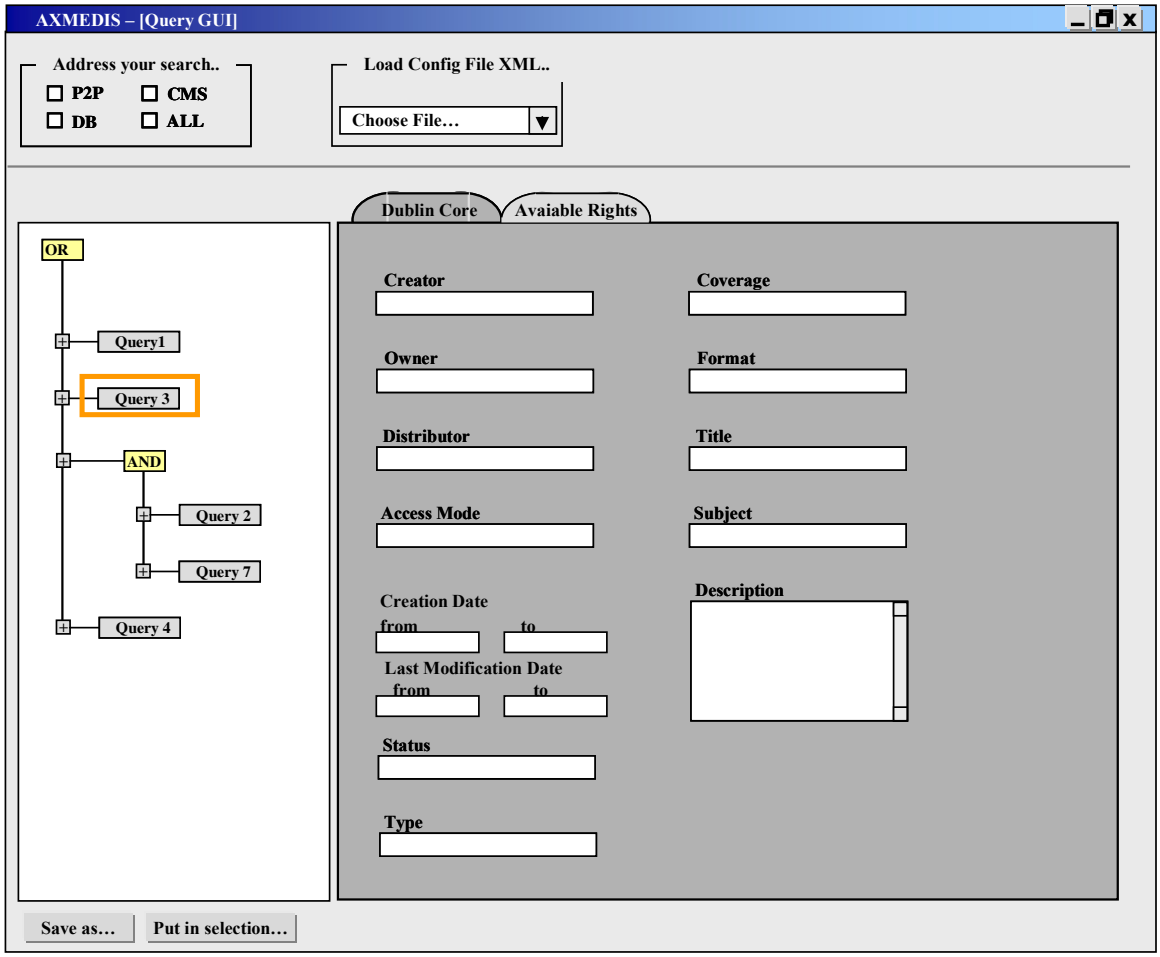
Query1 **OR** *Query3* **OR** (*Query2* **AND** *Query7*) **OR** *Query4*

In the picture below, the user can select a node in the tree and append a new level of nesting or a new flat query or select an existing query from the database.



2.6.3 Composing all together

The Flat QUI and the Tree QUI can be composed together to produce a general Query User Interface. The left panel contains the Tree QUI, and the selected item (the *Query3*) is displayed in the right panel.

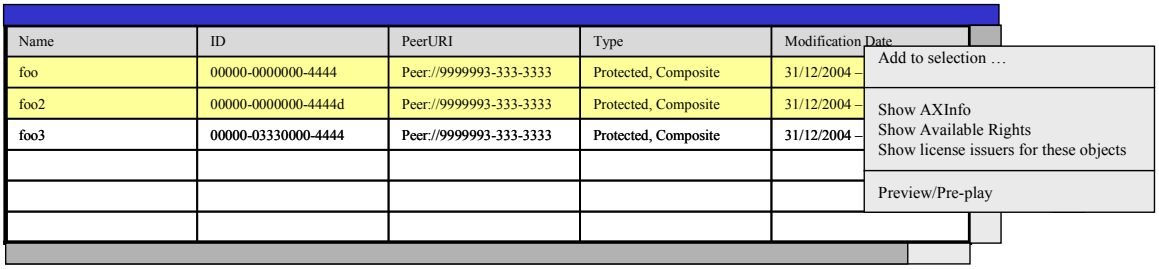


2.6.4 Query Results User Interface

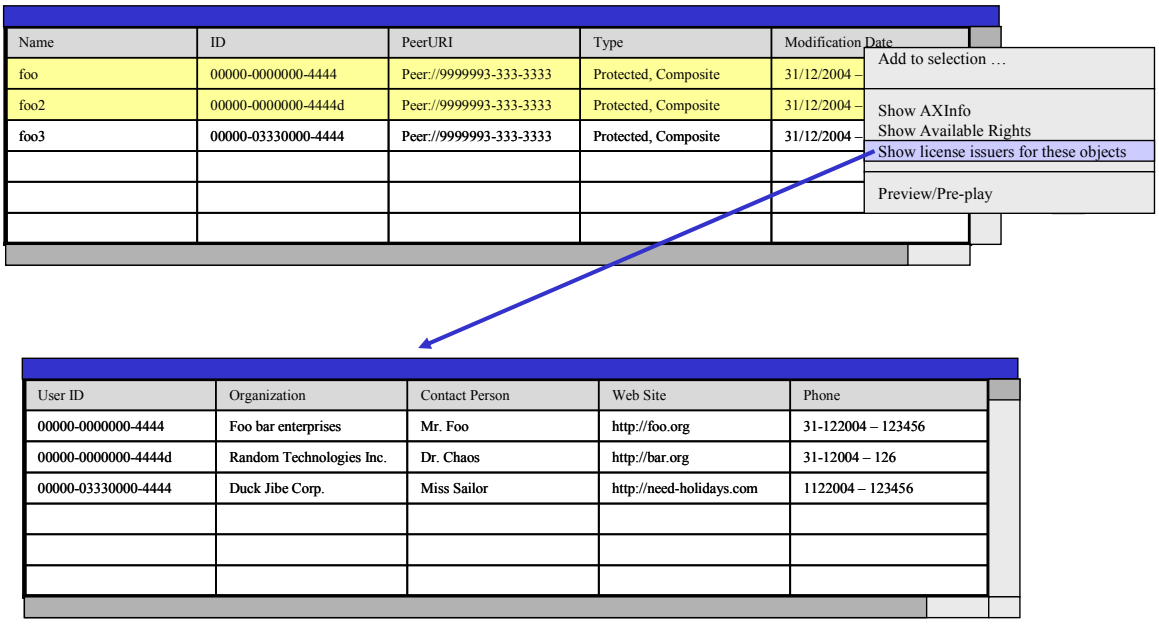
The results of query can be displayed in form of a table

Name	ID	PeerURI	Type	Modification Date
foo	00000-0000000-4444	Peer://9999993-333-3333	Protected, Composite	31/12/2004 – 12:34:56
foo2	00000-0000000-4444d	Peer://9999993-333-3333	Protected, Composite	31/12/2004 – 12:34:56
foo3	00000-03330000-4444	Peer://9999993-333-3333	Protected, Composite	31/12/2004 – 12:34:56

A single item or selection of items can be right-clicked and a popup exposes some actions to the user:



If the query contains a search on particular licensing aspects, the user can select objects, right-click on them and perform a query on the license database in order to retrieve the list of actors able to issue a license for these objects. The UI are showed below.



2.7 AXEPTool P2P Active Selections Module (CRS4)

Module Profile		
AXEPTool P2P Active Selections Module		
Executable or Library(Support)	Library	
Single Thread or Multithread	--	
Language of Development	C++/Javascript/XML	
Responsible Name	Carboni	
Responsible Partner	CRS4	
Status (proposed/approved)	proposed	
Platforms supported	Win32, Linux, MacOS	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
	AXCP Rule Engine	
	Publishing and Monitoring Objects	
	Download Monitor AXDBINM	
File Formats Used	Shared with	File format name or reference to a section
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

2.7.1 P2P Active Selections Format

A P2P Active Selection is a Rule of type "AXCP" which contains a script that activates the download of a selection. As AXCP rule it has the following properties:

- A **date** from which the Selection starts to be active
- A **periodicity** which specifies its execution periodicity for Rule Engine
- An **expire time** which specifies when the Rule is expired and it will not be executed anymore.

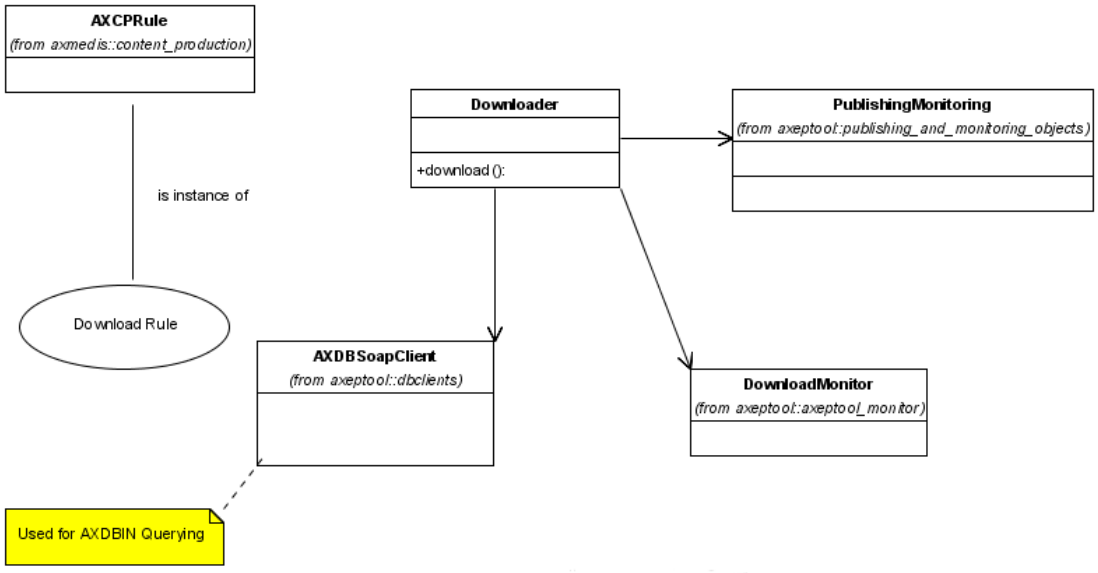
2.7.2 P2P Active Selections Component

The P2P Active Selections module contains the following sub-components:

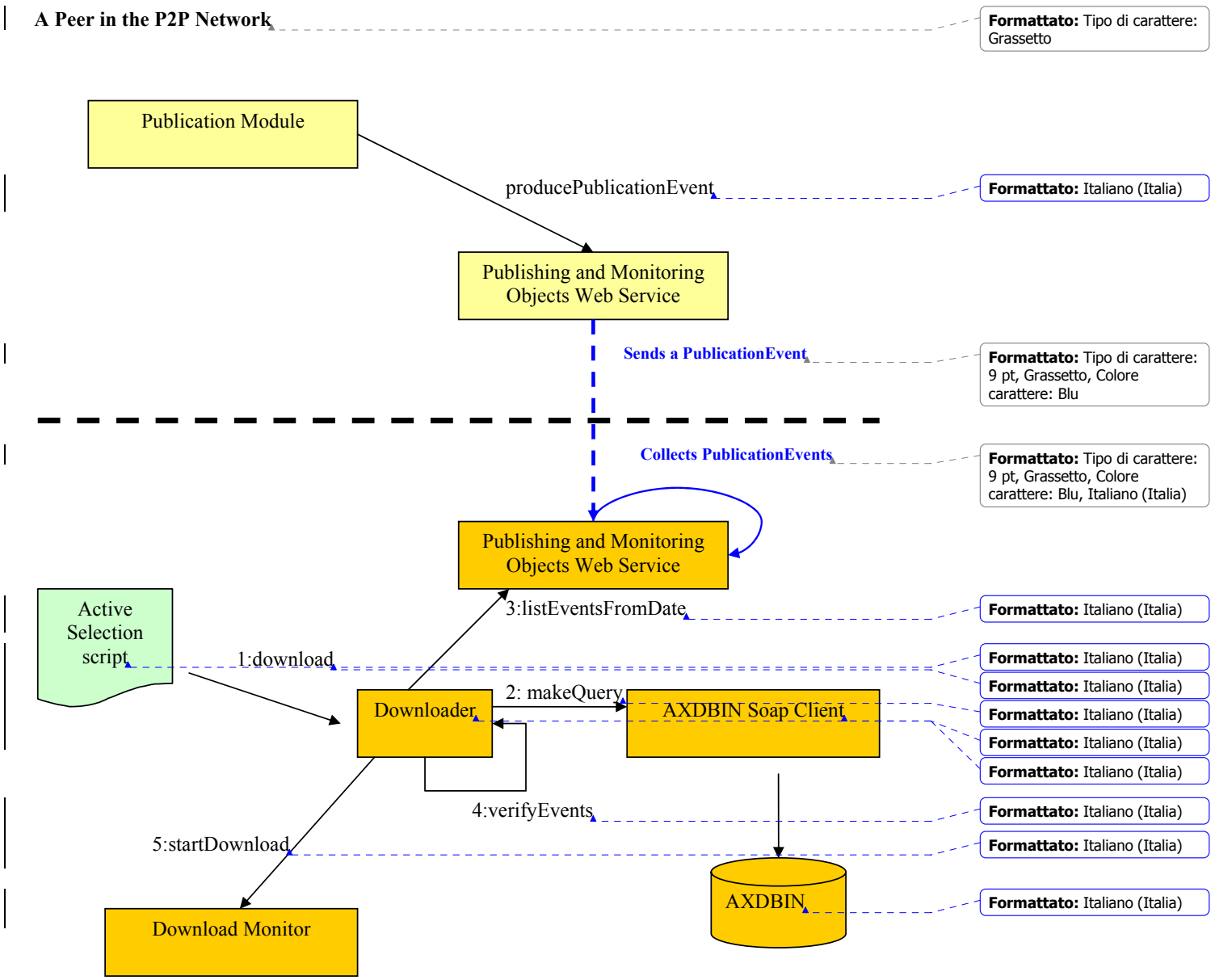
- JS_Downloader
- Downloader Component
- Active Selection (Download Rule)
- DownloadMonitor Client
- AXDBIN Client
- Publishing and Monitoring Objects Client

- The **Downloader** is the class loaded by the script engine and wrapped by a JS_Downloader Javascript class, so that its methods can be invoked in scripts. It contains one fundamental method:
  - download(), download the actualized selection of objects
- The **download rule** is the Active Selection and is not a class, it is an instance of the AXCP Rule class.

2.7.3 Class Diagram



2.7.4 Collaboration Diagram: Active Selections execution



An Active Selection execution involves the following steps: first of all the Rule Engine executes the script for the Active Selection using its date respecting the periodicity and the expire time; second:

1. **download**: the Downloader instance is commanded to download the objects in the selection of the rule. This command usually comes from the JS script.

2. **makeQuery**: the Downloader invokes an operation to the AXDBIN Soap Client in order to obtain a list of AXOB actually in AXDBIN
3. **listEventsFromDate**: the Downloader asks the Publishing and Monitoring Objects WebService for events came from the P2P Network. An event in the Publishing and Monitoring Objects module is be a PublicationEvent, it means that an AXOB has been published by another peer in the P2P Network. The Publishing and Monitoring Objects of the publishing peer sends a PublicationEvent to the Publishing and Monitoring Objects module of other peers, these modules collect all received events. The invoked method returns a list of events since the last activation period.
4. **verifyEvents:** for each event received in step 3, the Downloader Component verifies if it belongs to a new AXOB PublicationEvent, it checks if the AXOB under exam satisfies the Active Selection and if a new download is necessary.

5. **startDownload:** if a download action is required, the DownloadMonitor WebService is invoked and it starts downloading the AXOB.

2.8 Publishing and Monitoring Objects (CRS4)

Module Profile		
Publishing and Monitoring Objects		
Executable or Library(Support)	Library	
Single Thread or Multithread	Not applicable	
Language of Development	Java	
Responsible Name	D. Carboni	
Responsible Partner	CRS4	
Status (proposed/approved)	proposed	
Platforms supported	All	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
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

This component performs two tasks:

when an object is published or updated (by the Publication Module during the execution of a Publication Rule) a PublicationEvent is generated and sent to the P2P Network. To keep a low amount of traffic, only Peers that have already downloaded a old version of objects are recipients of such events.

it receives events from its counterpart in another peer and enqueue such events in a log. The P2P Active Selection Module will retrieve such events during the execution of Downloading Rules in order to establish whether or not new download session must be started.

2.8.1 Components

2.8.1.1 Publishing Monitoring Objects Web Service.

The Publishing and Monitoring Objects WebService exposes the following methods:

listEventsFromDate	Given a date it returns all PublicationEvent that happened starting from the specified date
producePublicationEvent	Given a collection of published AXOBs and a peerID it generates an event to alert the identified peer that a new collection of objects have been published

The WSDL interface is presented below:

```
<?xml version='1.0' encoding='utf-8' ?>
<definitions
name='axeptool.publishing_and_monitoring_objects.PublishingMonitoringInterface'
targetNamespace='http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/'
    xmlns:tns='http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/'
    xmlns:ns0='http://axmedis.org/xsd/SchemaTypes/'
    xmlns:map='http://axmedis.org/mapping/'
    xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
    xmlns='http://schemas.xmlsoap.org/wsdl/'>
    <types>
        <xsd:schema elementFormDefault="qualified"
            targetNamespace="http://axmedis.org/wsdl/axeptool/"
            xmlns:map="http://axmedis.org/mapping/"
            xmlns:tns="http://axmedis.org/wsdl/axeptool/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:complexType name="Event">
                <xsd:annotation>
                    <xsd:appinfo>
                        <map:java-type name="axeptool.Event"/>
                    </xsd:appinfo>
                </xsd:annotation>
                <xsd:sequence/>
            </xsd:complexType>
        </xsd:schema>

        <xsd:schema elementFormDefault="qualified"
            targetNamespace="http://axmedis.org/wsdl/axmedis/"
            xmlns:map="http://axmedis.org/mapping/"
            xmlns:tns="http://axmedis.org/wsdl/axmedis/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:complexType name="AXOBID">
                <xsd:annotation>
                    <xsd:appinfo>
                        <map:java-type name="axmedis.AXOBID"/>
                    </xsd:appinfo>
                </xsd:annotation>
                <xsd:sequence/>
            </xsd:complexType>
            <xsd:complexType name="ArrayOfAXOBID">
                <xsd:annotation>
                    <xsd:appinfo>
                        <map:java-type name="[Laxmedis.AXOBID;"/>
                    </xsd:appinfo>
                </xsd:annotation>
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="AXOBID" nillable="true" type="tns:AXOBID"/>
                </xsd:sequence>
```

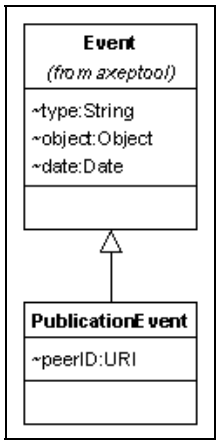
```
            </xsd:complexType>
        </xsd:schema>

        <xsd:schema elementFormDefault="qualified"

targetNamespace="http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/"
            xmlns:map="http://axmedis.org/mapping/"

xmlns:tns="http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/"
            xmlns:xns5="http://axmedis.org/wsdl/axeptool/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import namespace="http://axmedis.org/wsdl/axeptool/"/>
            <xsd:complexType name="PublicationEvent">
                <xsd:annotation>
                    <xsd:appinfo>
                        <map:java-type

name="axeptool.publishing_and_monitoring_objects.PublicationEvent"/>
                    </xsd:appinfo>
                </xsd:annotation>
                <xsd:complexContent>
                    <xsd:extension base="xns5:Event">
                        <xsd:sequence/>
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>
            <xsd:complexType name="ArrayOfPublicationEvent">
                <xsd:annotation>
                    <xsd:appinfo>
                        <map:java-type

name="[Laxeptool.publishing_and_monitoring_objects.PublicationEvent;"/>
                    </xsd:appinfo>
                </xsd:annotation>
                <xsd:sequence>
                    <xsd:element maxOccurs="unbounded" minOccurs="0"
                        name="PublicationEvent" nillable="true"
                        type="tns:PublicationEvent"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>

        <xsd:schema elementFormDefault="qualified"
            targetNamespace="http://axmedis.org/xsd/SchemaTypes/"
            xmlns:tns="http://axmedis.org/xsd/SchemaTypes/"

xmlns:xns4="http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/"
            xmlns:xns5="http://axmedis.org/wsdl/axmedis/"
            xmlns:xns6="http://axmedis.org/wsdl/java/net/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import namespace="http://axmedis.org/wsdl/java/net/"/>
            <xsd:import namespace="http://axmedis.org/wsdl/axmedis/"/>
            <xsd:import

namespace="http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/"/>
            <xsd:element name="p0" nillable="true" type="xsd:dateTime"/>
            <xsd:element name="ArrayOfPublicationEvent_Response" nillable="true"
                type="xns4:ArrayOfPublicationEvent"/>
            <xsd:element name="p0_1" nillable="true" type="xns5:ArrayOfAXOBID"/>
            <xsd:element name="p1" nillable="true" type="xns6:URI"/>
            <xsd:element name="int_Response" type="xsd:int"/>
        </xsd:schema>

        <xsd:schema elementFormDefault="qualified"
            targetNamespace="http://axmedis.org/wsdl/java/net/"
            xmlns:map="http://axmedis.org/mapping/"
            xmlns:tns="http://axmedis.org/wsdl/java/net/"
```

```
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:complexType name="URI">
                <xsd:annotation>
                    <xsd:appinfo>
                        <map:java-type name="java.net.URI"/>
                    </xsd:appinfo>
                </xsd:annotation>
                <xsd:sequence>
                    <xsd:element name="absolute" type="xsd:boolean"/>
                    <xsd:element name="authority" nillable="true"
                        type="xsd:string"/>
                    <xsd:element name="fragment" nillable="true"
                        type="xsd:string"/>
                    <xsd:element name="host" nillable="true" type="xsd:string"/>
                    <xsd:element name="opaque" type="xsd:boolean"/>
                    <xsd:element name="path" nillable="true" type="xsd:string"/>
                    <xsd:element name="port" type="xsd:int"/>
                    <xsd:element name="query" nillable="true"
                        type="xsd:string"/>
                    <xsd:element name="rawAuthority" nillable="true"
                        type="xsd:string"/>
                    <xsd:element name="rawFragment" nillable="true"
                        type="xsd:string"/>
                    <xsd:element name="rawPath" nillable="true"
                        type="xsd:string"/>
                    <xsd:element name="rawQuery" nillable="true"
                        type="xsd:string"/>
                    <xsd:element name="rawSchemeSpecificPart" nillable="true"
                        type="xsd:string"/>
                    <xsd:element name="rawUserInfo" nillable="true"
                        type="xsd:string"/>
                    <xsd:element name="scheme" nillable="true"
                        type="xsd:string"/>
                    <xsd:element name="schemeSpecificPart" nillable="true"
                        type="xsd:string"/>
                    <xsd:element name="userInfo" nillable="true"
                        type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>

    </types>
    <message name='PublishingMonitoringInterface_listEventsFromDate_Response_Soap'>
        <part name='response' element='ns0:ArrayOfPublicationEvent_Response'/>
    </message>
    <message name='PublishingMonitoringInterface_listEventsFromDate__Request_Soap'>
        <part name='p0' element='ns0:p0'/>
    </message>
    <message name='PublishingMonitoringInterface_producePublicationEvent_Response_Soap'>
        <part name='response' element='ns0:int_Response'/>
    </message>
    <message name='PublishingMonitoringInterface_producePublicationEvent__Request_Soap'>
        <part name='p0' element='ns0:p0_1'/>
        <part name='p1' element='ns0:p1'/>
    </message>
    <portType name='PublishingMonitoringInterface'>
        <operation name='listEventsFromDate' parameterOrder='p0'>
            <input
message='tns:PublishingMonitoringInterface_listEventsFromDate__Request_Soap'/>
            <output
message='tns:PublishingMonitoringInterface_listEventsFromDate_Response_Soap'/>
        </operation>
        <operation name='producePublicationEvent' parameterOrder='p0 p1'>
            <input
message='tns:PublishingMonitoringInterface_producePublicationEvent__Request_Soap'/>
            <output
message='tns:PublishingMonitoringInterface_producePublicationEvent_Response_Soap'/>
```

```
        </operation>
    </portType>
    <binding name='PublishingMonitoringInterface'
type='tns:PublishingMonitoringInterface'>
        <soap:binding transport='http://schemas.xmlsoap.org/soap/http' style='document'/>
        <operation name='listEventsFromDate'>
            <map:java-operation name='listEventsFromDate'
signature='KExqYXZhL3V0aWwvRGF0ZTspW0xheGVwdG9vbC9wdWJsaXNoaW5nX2FuZF9tb25pdG9yaW5nX29iam
VjdHMvUHVibGljYXRpb25FdmVudDs='/>
            <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/Publishing
MonitoringInterface#listEventsFromDate?KExqYXZhL3V0aWwvRGF0ZTspW0xheGVwdG9vbC9wdWJsaXNoaW
5nX2FuZF9tb25pdG9yaW5nX29iamVjdHMvUHVibGljYXRpb25FdmVudDs=' style='document'/>
            <input>
                <soap:body parts='p0' use='literal'/>
            </input>
            <output>
                <soap:body parts='response' use='literal'/>
            </output>
        </operation>
        <operation name='producePublicationEvent'>
            <map:java-operation name='producePublicationEvent'
signature='KFtMYXhtZWRpcy9BWE9CSUQ7TGphdmEvbmV0L1VSSTspSQ=='/>
            <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/Publishing
MonitoringInterface#producePublicationEvent?KFtMYXhtZWRpcy9BWE9CSUQ7TGphdmEvbmV0L1VSSTspS
Q==' style='document'/>
            <input>
                <soap:body parts='p0 p1' use='literal'/>
            </input>
            <output>
                <soap:body parts='response' use='literal'/>
            </output>
        </operation>
    </binding>
    <service name='PublishingMonitoringInterface'>
        <port name='PublishingMonitoringInterface'
binding='tns:PublishingMonitoringInterface'>
            <soap:address
location='http://percival:6060/PublishingMonitoringInterface/'/>
        </port>
    </service>
</definitions>
```

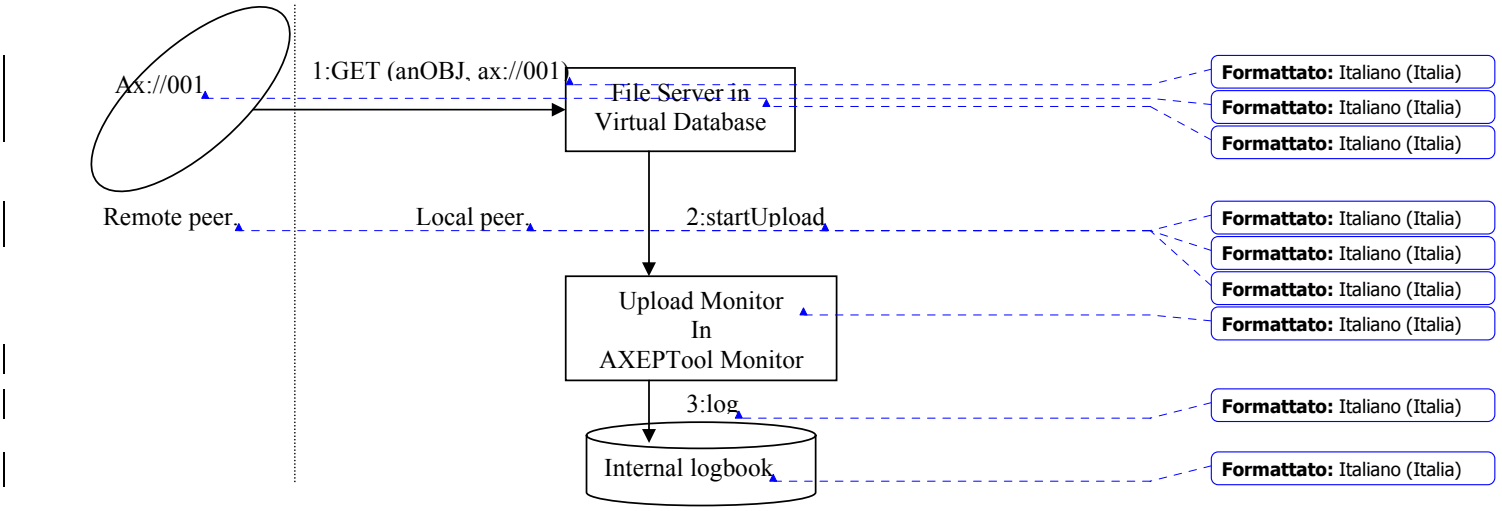
2.8.1.2 Publication Event

The Publication Event is the event sent through the P2P network and collected in this module. It contains:
Type: optional field inherited by Event
Object: optional filed that can embed an arbitrary data
Date: event date and time
peerID: the URI of the peer that generated the event in form ax://<uuid>



2.8.2 Collaboration Diagram: Registering for publishing events

1. *GET*(object ID=anOBJ, requester URI=ax://001): from the P2P network this message is processed by the File Server
2. *startUpload:* the Upload Monitor is invoked, a new upload is created and started
3. *log*: the Upload Monitor adds a record on the internal logbook when a downloader completes the download



As result, the peer ***ax://001*** is registered as downloaeder of object ***anOBJ.*** In case of publication of new versions of this object the peer ***ax://001*** will be notified (see **Collaboration Diagram: Publication of Objects in a super-peer** in Publication Module section)

2.9 P2P Low Level Virtual Database (CRS4)

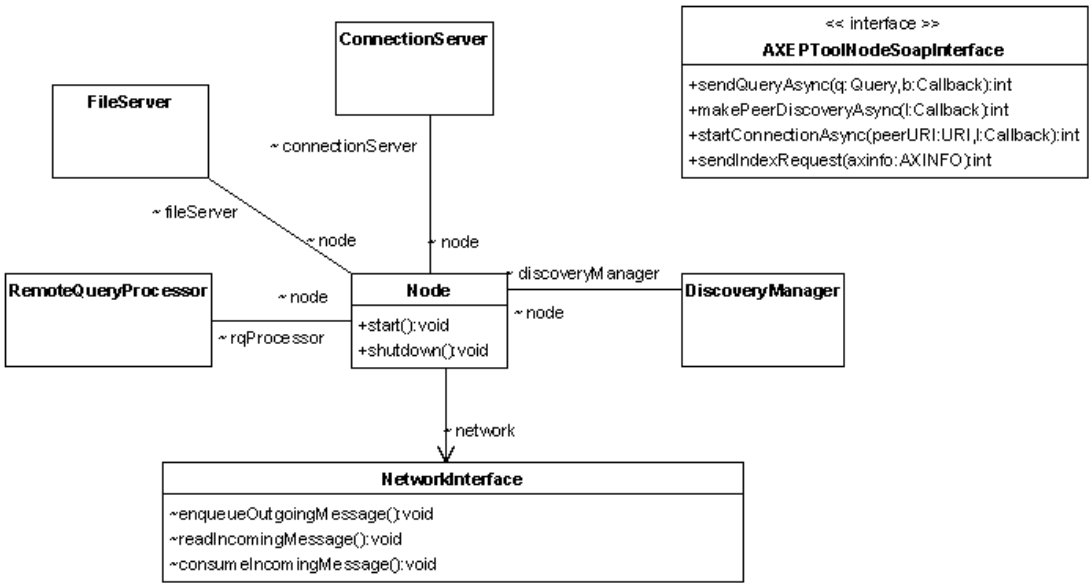
Module Profile		
P2P Virtual Database		
Executable or Library(Support)	Executable	
Single Thread or Multithread	Multithreaded	
Language of Development	Java	
Responsible Name	D. Carboni	
Responsible Partner	CRS4	
Status (proposed/approved)		
Platforms supported	All	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
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

Business users share a common environment in which AXMEDIS Objects can be published and loaded without the need of a centralized infrastructure. The AXMEDIS P2P Low Level Virtual database is an abstraction term for an underlying P2P Network architecture in phase of definition.
The AXMEDIS P2P Network will appear as a distributed database to the other areas of AXMEDIS. The AXEPTool will be a node of this Network, it has direct access to it and provides the abstraction.
AXMEDIS Objects should be published and loaded without the need of a centralized infrastructure which would be costly to maintain and a potential bottleneck in the whole production chain. Some functionalities of the network could be centralized wherever the centralization allows better performances, security, and dramatic traffic reduction.
The messages exchanged between hosts in the P2P network should be transported in a way that the presence of firewalls and NATs do not compromise the network itself. Thus, HTTP tunnelling and other appropriated solutions will be designed and deployed whenever necessary. P2P Low Level Virtual Database can be thought as the set of the architecture of the P2P network and the protocol which allows the peers do make their activities (discovering, querying, …).
At this moment of the designing it's not very clear the final shape of such architecture (centralized, decentralize, hybrid). One reasonable possibility is the protocol provided by Gnutella2 network.

From a general architectural view we can shape this module subdividing it in components shown in the next UML diagram.

Commento [c1]:

2.9.1 Class Diagram



In the previous diagram we can see a first detailed view of components taking part in the P2P Low Level Virtual Database module. In the following sections we analyse the components.

2.9.2 Network Interface

This component is an abstraction of the network. Client objects can send, read and consume messages from the network.

2.9.3 Node

The node component is the core objects which dispatches messages to all the other components in the Virtual Database.

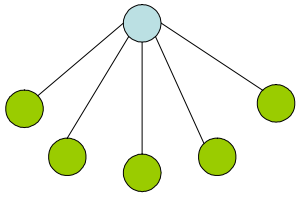
2.9.4 P2P Network Topology

Defining a P2P Network Architecture or choosing an existing one involves the important choice of the network topology. Every P2P network follow a design model. At a first sight we can distinguish some main network topologies:

2.9.4.1 Centralized Network Topology

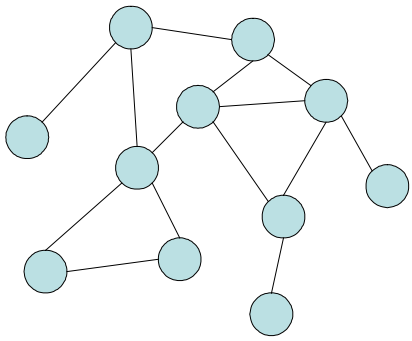
Probably, it is the most widely used topology around Internet, think about client/server models used by databases, web servers, and other simple distributed systems. All network functionalities and information is centralized and managed by a server and many clients connect directly to the server for task requesting and to send or receive data. Many P2P applications also have a centralized component. For example, original

Napster's search architecture was centralized, although the file sharing was not. Next picture shows a typical centralized topology



2.9.4.2 Decentralized Network Topology

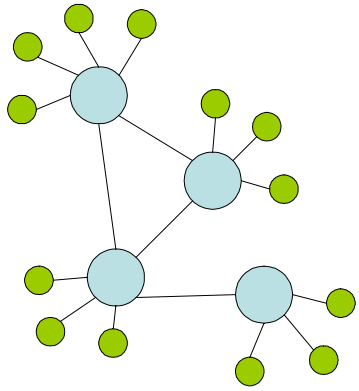
The base concept behind a decentralized network topology is a model where all peers communicate symmetrically and have equal roles. Communication can occur between any two nodes, with each node holding routing information for others. Gnutella, is an examples of decentralized P2P network, with only a small centralized function to bootstrap a new host, especially after the introduction of “super-peer” nodes. Next picture shows a node organization in a decentralized topology.



Decentralized networks, base its own maintenance on broadcast flooding. Flooding is not very efficient and entails some bandwidth overhead for the network.
They also tend to be insecure: it is easy for a node to join the network and start putting bad data into the network. On the other hand a virtue of decentralized systems is their extensibility.

2.9.4.3 Hybrid Network Topology

An hybrid topology is formed using a combination of the previous topologies.
An example can be the hybrid "centralized – decentralized" topology where some "hub" or "super-peer" nodes act like a server for several network functions, serving "client" leaf nodes, but "hubs" form a decentralized network topology. In this topology falls the Gnutella2 P2P Network.
Next picture shows a diagram for this network type:



A network combining centralized and decentralized systems benefits of some of the advantages of both: Centralized and Decentralized. Decentralization contributes to the extensibility and fault-tolerance. The partial centralization makes the network more coherent than a purely decentralized system, as there are relatively fewer hosts that are holding authoritative data. Manageability is about as difficult as a decentralized system, and the system is no more secure than any other decentralized system but a remarkable property is the scalability of this hybrid topology.

Starting from this point we will refer to a P2P Network architecture of hybrid type adoption for AXMEDIS P2P Virtual Database design scope.
In particular we introduce two types of network peers:

- A **Leaf** peer with limited responsibilities in the network
- A **Ultra-peer** (also called **Hub**) with high responsibilities in the network, as it will show

Leaf nodes are directly connected with Ultra-Peer nodes, Ultra-Peers are highly interconnected between them and form an important and active part of the network infrastructure, organising surrounding nodes, filtering and directing traffic over several media types. Ultra-Peers nodes devote substantial resources to the network, and as a result their capacity to participate in higher level network functions is limited.

2.9.5 Discovery Manager

A first, higher, level is taken by the Discovery Manager. It performs all the necessary bootstrap operations in order to connect the peer to the P2P Network.
It is an independent thread which iteratively absolves the following main tasks:

- Searching of new "connectable" peers in the network
- Management of an internal updated Node Cache: a table with discovered peers IPs

Strictly related to the P2P Network Architecture is the mechanism of discovery. For example, in this bootstrap step, making uses of WebCaches applications can be a reasonable choice.
A WebCache is a web application, published on a http server and written in any language (Java Servlet, PHP, ASP, Perl,…) which maintain an updated table of network peers and a list of other WebCaches. Peers in bootstrap phase can ask them for an updated list of other nodes IP, in order to try to establish a connection with them. Requests are performed using simple parameterized HTTP's GET requests. WebCaches tables are updated by "reliable" peers (ultra-peers or hubs, depending on terminology) on the network.
As written above, this module performs the task of discovering peers on the P2P Network. It's behaviour, tasks and protocol used are strictly related to the P2P Network implementation.

2.9.5.1 Registrar / Web Cache modules

In order to connect to the P2P Network a "potential" peer must know at least another peer IP address.

This bootstrap information needed can be obtained dynamically connecting and querying a Registrar, often called WebCache, module.
Registar or WebCache modules simply web applications running at well known URLs.
They can be contacted and queried by peers sending them simple HTTP GET request with suitable and well defined parameters and they mainly answer providing an updated list of IP addresses of other peers connected to the P2P Network.
A Registrar / WebCache application mainly absolves the following tasks:

1. Maintain an updated internal cache of IP addresses of peers in the P2P network
2. Update the internal cache using "update information" provided by "authorized" special nodes of the network
3. Answer to peers requests for an update list of IPs using the IPs stored in its internal cache
4. Maintain an updated list of URLs of other WebCaches in the Internet

In this way a new peer which wants to attempt a connection to the P2P Network can have a list of well known WebCaches and ask them for IP addresses of network peers to try to connect to.

Considering the two node types distinction, WebCaches are updated by hub peers and can be queried for IP lists by both peer types.
As will be show in the network topology section, an hybrid P2P network with "Centralized-Decentralized" topology is composed by Leaf peers connected to Ultra-peer (Hubs) peers interconnected with other Hubs in the network. Last type of peers have big responsibilities in network maintenance and in network services like queries forward; they maintain peers cache and "serve" leafs providing a reference point for their "requests".
For a new peer attempting to connect to the P2P network is necessary to know at least one "available to accept leafs" Hub node. This is made contacting a Registrar / WebCache service.

2.9.6 Peer ID

Every peer in the AXMEDIS P2P Network must be univocally identified. At a first sight a reasonable choice could be associating the peer to its IP address. Another chance is to associate a URN to every peer in the network. The URN could be generated one time when, i.e, the AXEPTool is installed on a machine.

A URN is defined as follows:

<URN> ::= "urn:" <NID> ":" <NSS>

Where <NID> is the Namespace Identifier Syntax, defined as follow:

```
<NID> ::= <let-num> [ 1,31<let-num-hyp> ]
<let-num-hyp> ::= <upper> | <lower> | <number> | "-"
<let-num>::= <upper> | <lower> | <number>
<upper> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" |  "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" |
            "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
<lower>     ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" |
            "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
<number> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |  "8" | "9"
```

And <NSS> is the Name space Specific String, defined as:

```
<NSS>::= 1*<URN chars>

  <URN chars>::= <trans> | "%" <hex> <hex>

  <trans>::= <upper> | <lower> | <number> | <other> | <reserved>

  <hex>::= <number> | "A" | "B" | "C" | "D" | "E" | "F" |
           "a" | "b" | "c" | "d" | "e" | "f"

  <other>::= "(" | ")" | "+" | "," | "-" | "." |
           ":" | "=" | "@" | ";" | "$" |
           "_" | "!" | "*" | "'"

<reserved>    ::= '%" | "/" | "?" | "#"

<excluded> ::= octets 1-32 (1-20 hex) | "\" | """ | "&" | "<"
           | ">" | "[" | "]" | "^" | "`" | "{" | "|" | "}" | "~"
           | octets 127-255 (7F-FF hex)
```

Using URNs we can abstract from IP address and identify a peer independently from its eventually temporarely connection.
On the other hand, using URNs, a peer address solving mechanism is required.

2.9.7 Connection Server.

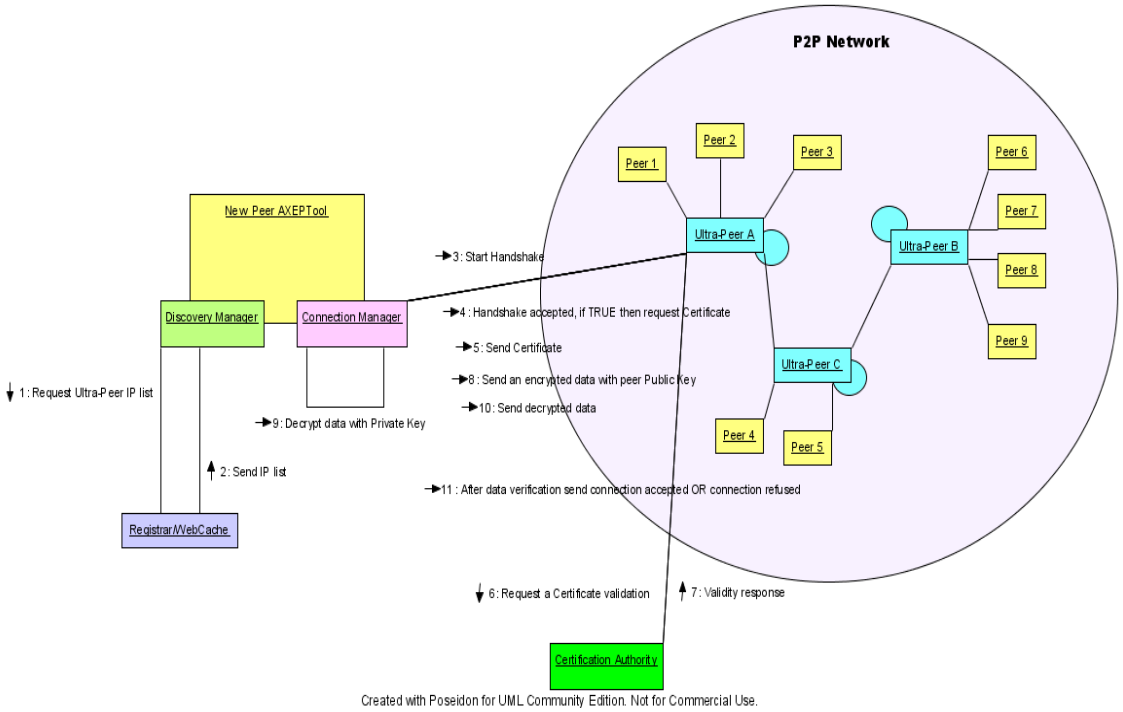
This module cares the connection phase between two peers in the P2P Network in a secure way and it's handshake related step. Handshaking step is basically performed with a well defined information messages

exchange between the peer which wants to establish a connection with another peer and the contacted peer itself. At a architectural first view we can define two types (or roles) of peers in the P2P Network: peers and ultra-peers, the first ones are normal peers of the network with low burden and capable to connect to and exchange information with other peers. The second ones act like also as "hubs" and are capable of receiving TCP/IP incoming connections from other peers taking an updated table of neighbour peers and other ultra-peers. This component is optional because some peers will not be able to receive incoming connections.

2.9.8 Collaboration Diagram: Discovery and Connecting to the network

Next diagram shows the steps performed by involved modules for a new Peer (an AXEPTool) which wants to connect to the P2P Network. Modules involved are:

- An AXEPtool representing the new peer which intend to connect to the AXMEDIS P2P Network
- One or more well known Registrar / WebCache; for "well known" is indented the URL knowledge
- The AXMEDIS P2P Network, an hybrid P2P network composed by Peers and Ultra-Peers
- A Certification Authority, for Certificates verification such as the AXMEDIS Certifier and Supervisor, as described in part H of the document.



A new AXEPtool wants to become a new peer in the AXMEDIS P2P Network, steps:

1. **Request Ultra-Peer list**: the AXEPTool has a list of AXMEDIS Registrars/WebCaches URLs; the Discovery manager tries to query one or more of these web applications in order to obtain a list of Ultra-Peers IPs to try to connect to.
2. **Send IP list**: the Registrar/WebCache queried sends its response back to the peer; if positive the response includes an Ultra-Peers IP list. If response is negative, the peer repeats actions in step 1.

3. **Start Handshake:** at this moment, the AXEPTool has a list of Ultra-Peers IP list; Now it tries to iteratively handshake one or more of these peers. An handshaking phase consists in some information exchange and negotiations between a peer already existing in the network and the new peer trying to connect. Handshaking information could include the client type or name, client version, protocol (if more than one protocol is allowed), the peer type: Peer or Ultra-Peer, etc…
4. **Handshake accepted, If TRUE then request Certificate:** If Handshaked is refused by Ultra-Peer, i.e. because it has reached the maximum number of connections it can manage, the peer use IP list received at step 1 and process restarts. If the handshaking is accepted by Ultra-Peer contacted, then it MUST assure the peer identity and reliability. This can be made using a Certificate. So, the peer which wants to connect to the AXMEDIS P2P Network must have a Certificate, released and granted by a reliable Certification Authority. The Ultra-Peer makes a request to the connecting peer for a Certificate.
5. **Send Certificate:** the peer sends its Certificate to Ultra-Peer
6. **Request a Certificate Validation:** the Ultra-Peer use Certification data to contact a Certification Authority service in order to verify the Certificate and peer identity. Probably the Ultra-Peer use an LDAP service or a Web Service maintained by the Certification Authority. For more details see AXMEDIS Certifier and Supervisor in part H of the document.
7. **Validity response:** The Certification Authority gives response about Certificate validity. If it is valid the next step is performed, else handshake stops here.
8. **Send an encrypted data with peer Public Key:** At this point, Ultra-Peer has the peer Certificate and its validity confirmation sent by the Certification Authority. A last verification is performed: the Ultra-Peer encrypts a data chunk (i.e. a unique random generated alphanumeric string) with the peer Public Key taken from the Certificate and sends it to the peer.
9. **Decrypt data with Private Key:** The peer has received the encrypted data chunk, it uses its own private key to decrypt it
10. **Send Decrypted data**: the peer sends the decrypted data chunk to the Ultra-Peer.
11. **After data verification send connection accepted OR connection refused:** The Ultra-Peer matches the last received data chunk with this sent. If they are the same then peer is secure and the connection is established; else connection is refused.

After a successfully completion of the 11th step, the AXEPTool is a new peer of the AXMEDIS P2P Network and can regularly start its operations. Starting from this moment all network communication is performed packaging data as required by the AXMEDIS P2P Network Protocol.

Other operations performed by the Connection Manager are these realated to basic network maintenance operations: ping its status to other Ultra-Peers, update its internal node caches. When a new Ultra-Peer connects to the network, it receives several data from its neighbours Ultra-Peers, i.e. node caches, so it can store this data in it internal caches and it enters in a particular subnet called “Ultra-Peer Cluster” where the subnet is composed by some neighbour Ultra-Peers.

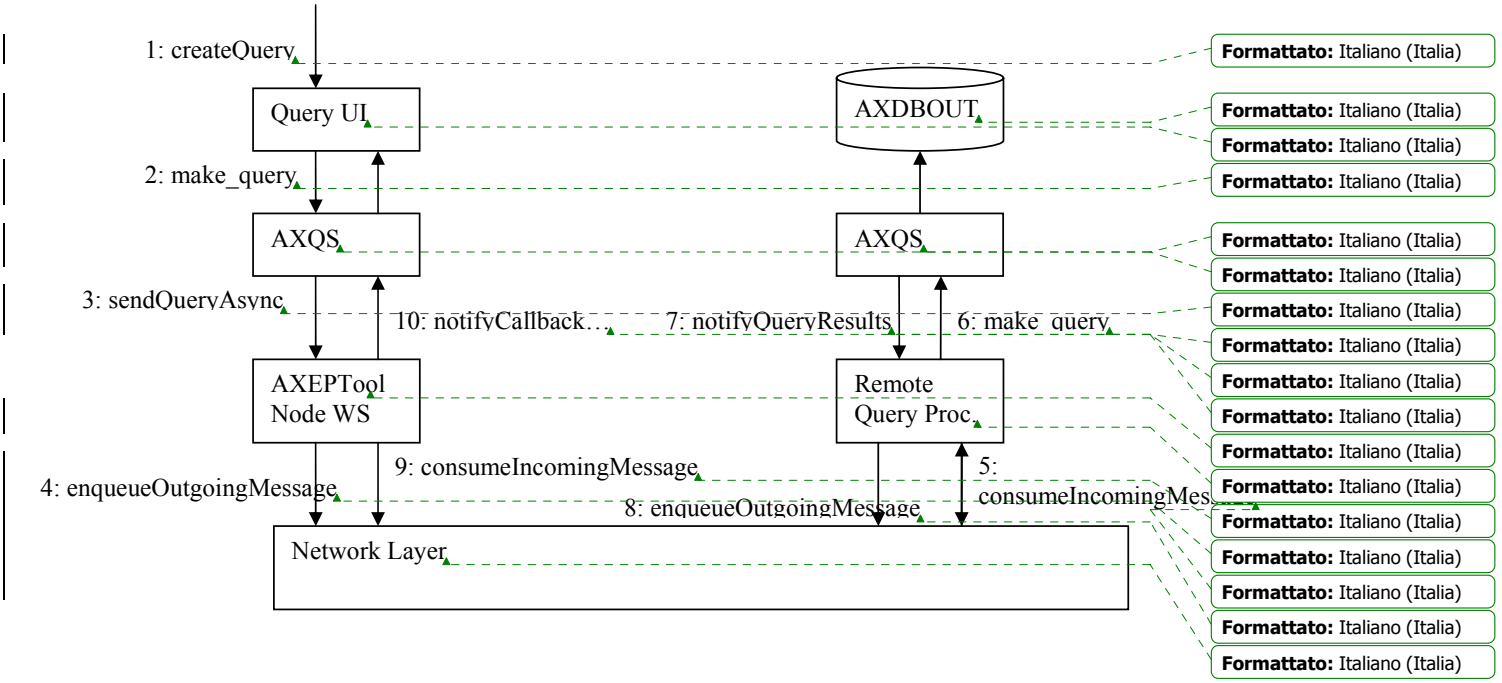
2.9.9 Remote Query Processor

It’s main task is to “collect” the queries produced by remote peers, connect to the Axmedis Query Support and obtain a list of query results to send back to the network. This component is optional depending on the P2P network topology. In fact, if the peer is a leaf it is likely to index all contents and send the index to a high-level peer (ultra-peer or hub depending on terminology) which is committed to produce query results.

2.9.10 Collaboration Diagram: Producing a query and collecting results(CRS4, EXITECH)

1. *createQuery()*: The user creates the query by the Query User Interface and clicks the “send” button or an equivalent item in a menu.

2. *make_query*(): the AXQS is invoked to make a query
3. *sendQueryAsync*():The AXQS forwards the query to the AXEPTool Node.
4. *enqueueOutgoingMessage*: The query is packed and sent through the network
5. *consumeIncomingMessage*: this method unblocks, the query is unpacked from the incoming message the RemoteQueryProcessor is committed to process the query
6. *make_query*: the AXQS in the AXDBOUT is invoked to solve the remote query
7. *notifyQueryResults*: the RemoteQueryProcessor receives results from the local AXQS in the AXDBOUT
8. *enqueueOutgoingMessage*: query results are packed and sent back
9. *consumeIncomingMessage*: this method unblocks, it returns a message containing query results.
10. *notifyCallbackEvents*: the AXEPToolNode notifies query results using the callback reference



2.9.11 File Server.

At this level software modules don't manage AXOBs but they deal with raw files (it's a lower level of abstraction). A possible way can be the adoption of a Bittorrent like technology or another ad-hoc implementation of "torrents" adapted foe AXMEDIS needs.

2.9.12 AXEPToolNodeSoapInterface

In order to be remote controlled the AXEPTool exposes a WebService Interface, with the following methods:

sendQueryAsync(Query q, Callback l)	Invoked in order to send a query to be executed by the node. Query results will be notified to the specified callback
makePeerDiscoveryAsync(Callback l)	Starts the P2P peers discovery, callback will be

	asynchronously notified
startConnectionAsync(URI peerURI, Callback l);	Starts a connection to the specified peer, callback is used as shown previous
sendIndexRequest(AXINFO axinfo);	Sends a request to a remote super-peer for indexing a local object in a remote AXDBOUT

```
<?xml version='1.0' encoding='utf-8' ?>
<definitions name='axeptool.publishing_and_monitoring_objects.PublishingMonitoringInterface'
targetNamespace='http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/'
  xmlns:tns='http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/'
  xmlns:ns0='http://axmedis.org/xsd/SchemaTypes/'
  xmlns:map='http://axmedis.org/mapping/'
  xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns='http://schemas.xmlsoap.org/wsdl/'>
  <types>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://axmedis.org/wsdl/axeptool/"
      xmlns:map="http://axmedis.org/mapping/"
      xmlns:tns="http://axmedis.org/wsdl/axeptool/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:complexType name="Event">
        <xsd:annotation>
          <xsd:appinfo>
            <map:java-type name="axeptool.Event"/>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence/>
      </xsd:complexType>
    </xsd:schema>

    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://axmedis.org/wsdl/axmedis/"
      xmlns:map="http://axmedis.org/mapping/"
      xmlns:tns="http://axmedis.org/wsdl/axmedis/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:complexType name="AXOBID">
        <xsd:annotation>
          <xsd:appinfo>
            <map:java-type name="axmedis.AXOBID"/>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence/>
      </xsd:complexType>
      <xsd:complexType name="ArrayOfAXOBID">
        <xsd:annotation>
          <xsd:appinfo>
            <map:java-type name="[Laxmedis.AXOBID;"/>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element maxOccurs="unbounded" minOccurs="0"
            name="AXOBID" nillable="true" type="tns:AXOBID"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>

    <xsd:schema elementFormDefault="qualified"
```

```
      targetNamespace="http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/"
      xmlns:map="http://axmedis.org/mapping/"
      xmlns:tns="http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/"
      xmlns:xns5="http://axmedis.org/wsdl/axeptool/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import namespace="http://axmedis.org/wsdl/axeptool/"/>
      <xsd:complexType name="PublicationEvent">
        <xsd:annotation>
          <xsd:appinfo>
            <map:java-type
            name="axeptool.publishing_and_monitoring_objects.PublicationEvent"/>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:complexContent>
          <xsd:extension base="xns5:Event">
            <xsd:sequence/>
          </xsd:extension>
        </xsd:complexContent>
      </xsd:complexType>
      <xsd:complexType name="ArrayOfPublicationEvent">
        <xsd:annotation>
          <xsd:appinfo>
            <map:java-type
            name="[Laxeptool.publishing_and_monitoring_objects.PublicationEvent;"/>
          </xsd:appinfo>
        </xsd:annotation>
        <xsd:sequence>
          <xsd:element maxOccurs="unbounded" minOccurs="0"
            name="PublicationEvent" nillable="true"
            type="tns:PublicationEvent"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>

    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://axmedis.org/xsd/SchemaTypes/"
      xmlns:tns="http://axmedis.org/xsd/SchemaTypes/"
      xmlns:xns4="http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/"
      xmlns:xns5="http://axmedis.org/wsdl/axmedis/"
      xmlns:xns6="http://axmedis.org/wsdl/java/net/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import namespace="http://axmedis.org/wsdl/java/net/"/>
      <xsd:import namespace="http://axmedis.org/wsdl/axmedis/"/>
      <xsd:import
      namespace="http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/"/>
      <xsd:element name="p0" nillable="true" type="xsd:dateTime"/>
      <xsd:element name="ArrayOfPublicationEvent_Response" nillable="true"
        type="xns4:ArrayOfPublicationEvent"/>
      <xsd:element name="p0_1" nillable="true" type="xns5:ArrayOfAXOBID"/>
      <xsd:element name="p1" nillable="true" type="xns6:URI"/>
      <xsd:element name="int_Response" type="xsd:int"/>
    </xsd:schema>

    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://axmedis.org/wsdl/java/net/"
      xmlns:map="http://axmedis.org/mapping/"
      xmlns:tns="http://axmedis.org/wsdl/java/net/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:complexType name="URI">
```

```
            <xsd:annotation>
               <xsd:appinfo>
                  <map:java-type name="java.net.URI"/>
               </xsd:appinfo>
            </xsd:annotation>
            <xsd:sequence>
               <xsd:element name="absolute" type="xsd:boolean"/>
               <xsd:element name="authority" nillable="true"
                  type="xsd:string"/>
               <xsd:element name="fragment" nillable="true"
                  type="xsd:string"/>
               <xsd:element name="host" nillable="true" type="xsd:string"/>
               <xsd:element name="opaque" type="xsd:boolean"/>
               <xsd:element name="path" nillable="true" type="xsd:string"/>
               <xsd:element name="port" type="xsd:int"/>
               <xsd:element name="query" nillable="true"
                  type="xsd:string"/>
               <xsd:element name="rawAuthority" nillable="true"
                  type="xsd:string"/>
               <xsd:element name="rawFragment" nillable="true"
                  type="xsd:string"/>
               <xsd:element name="rawPath" nillable="true"
                  type="xsd:string"/>
               <xsd:element name="rawQuery" nillable="true"
                  type="xsd:string"/>
               <xsd:element name="rawSchemeSpecificPart" nillable="true"
                  type="xsd:string"/>
               <xsd:element name="rawUserInfo" nillable="true"
                  type="xsd:string"/>
               <xsd:element name="scheme" nillable="true"
                  type="xsd:string"/>
               <xsd:element name="schemeSpecificPart" nillable="true"
                  type="xsd:string"/>
               <xsd:element name="userInfo" nillable="true"
                  type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:schema>

   </types>
   <message name='PublishingMonitoringInterface_listEventsFromDate_Response_Soap'>
      <part name='response' element='ns0:ArrayOfPublicationEvent_Response'/>
   </message>
   <message name='PublishingMonitoringInterface_listEventsFromDate__Request_Soap'>
      <part name='p0' element='ns0:p0'/>
   </message>
   <message name='PublishingMonitoringInterface_producePublicationEvent_Response_Soap'>
      <part name='response' element='ns0:int_Response'/>
   </message>
   <message name='PublishingMonitoringInterface_producePublicationEvent__Request_Soap'>
      <part name='p0' element='ns0:p0_1'/>
      <part name='p1' element='ns0:p1'/>
   </message>
   <portType name='PublishingMonitoringInterface'>
      <operation name='listEventsFromDate' parameterOrder='p0'>
         <input message='tns:PublishingMonitoringInterface_listEventsFromDate__Request_Soap'/>
         <output message='tns:PublishingMonitoringInterface_listEventsFromDate_Response_Soap'/>
      </operation>
      <operation name='producePublicationEvent' parameterOrder='p0 p1'>
```

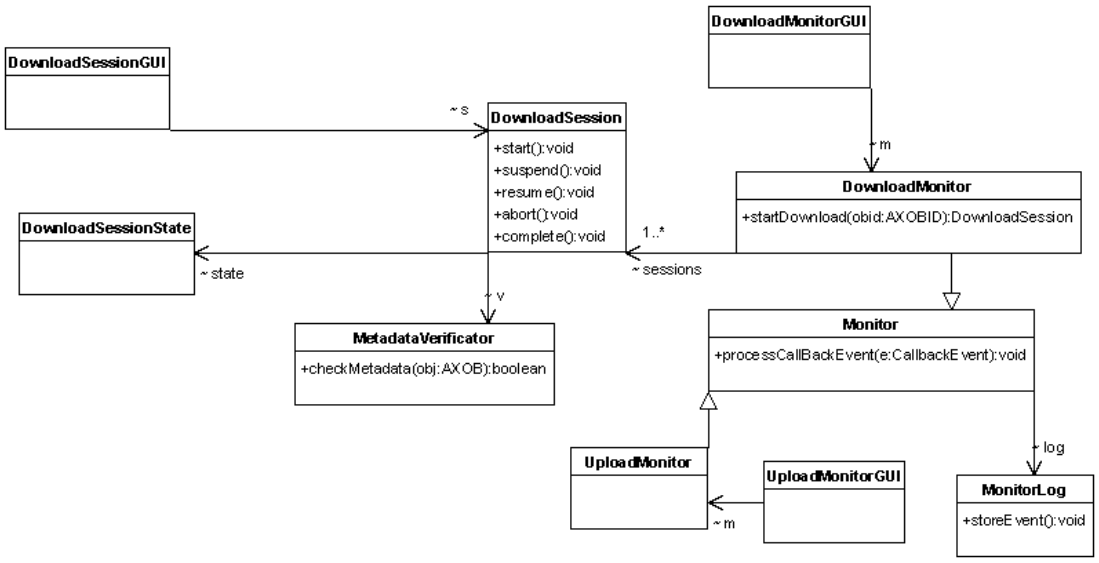
```
        <input message='tns:PublishingMonitoringInterface_producePublicationEvent__Request_Soap'/>
        <output message='tns:PublishingMonitoringInterface_producePublicationEvent_Response_Soap'/>
      </operation>
    </portType>
    <binding name='PublishingMonitoringInterface' type='tns:PublishingMonitoringInterface'>
      <soap:binding transport='http://schemas.xmlsoap.org/soap/http' style='document'/>
      <operation name='listEventsFromDate'>
        <map:java-operation name='listEventsFromDate'
signature='KExqYXZhL3V0aWwvRGF0ZTspW0xheGVwdG9vbC9wdWJsaXNoaW5nX2FuZF9tb25pdG9yaW
5nX29iamVjdHMvUHVibGljYXRpb25FdmVudDs='/>
        <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/PublishingMonitoringInterf
ace#listEventsFromDate?KExqYXZhL3V0aWwvRGF0ZTspW0xheGVwdG9vbC9wdWJsaXNoaW5nX2FuZF
9tb25pdG9yaW5nX29iamVjdHMvUHVibGljYXRpb25FdmVudDs=' style='document'/>
        <input>
          <soap:body parts='p0' use='literal'/>
        </input>
        <output>
          <soap:body parts='response' use='literal'/>
        </output>
      </operation>
      <operation name='producePublicationEvent'>
        <map:java-operation name='producePublicationEvent'
signature='KFtMYXhtZWRpcy9BWE9CSUQ7TGphdmEvbmV0L1VSSTspSQ=='/>
        <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/publishing_and_monitoring_objects/PublishingMonitoringInterf
ace#producePublicationEvent?KFtMYXhtZWRpcy9BWE9CSUQ7TGphdmEvbmV0L1VSSTspSQ=='
style='document'/>
        <input>
          <soap:body parts='p0 p1' use='literal'/>
        </input>
        <output>
          <soap:body parts='response' use='literal'/>
        </output>
      </operation>
    </binding>
    <service name='PublishingMonitoringInterface'>
      <port name='PublishingMonitoringInterface' binding='tns:PublishingMonitoringInterface'>
        <soap:address location='http://percival:6060/PublishingMonitoringInterface/'/>
      </port>
    </service>
</definitions>
```

2.10 AXEPTool Monitor (CRS4, DSI, FHGIGD)

Module Profile		
AXEPTool Monitor		
Executable or Library(Support)	Packages to be linked with the stand-alone application	
Single Thread or Multithread	Multi-threaded	
Language of Development	Java	
Responsible Name		
Responsible Partner		
Status (proposed/approved)		
Platforms supported		
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
File Formats Used	Shared with	File format name or reference to a section
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not

The AXEPTool Monitor performs the task of continuously giving real-time feedback about status of possible downloads and uploads of AXMEDIS objects from and to the P2P Low Level Virtual Database.
At a first level of detail there are two main components which absolve these tasks; The Downloads Monitor for downloads and the Uploads Monitor for uploads.
Data produced by Monitors is used by the AXEPTool Monitor GUI which gives a graphical representation of status.

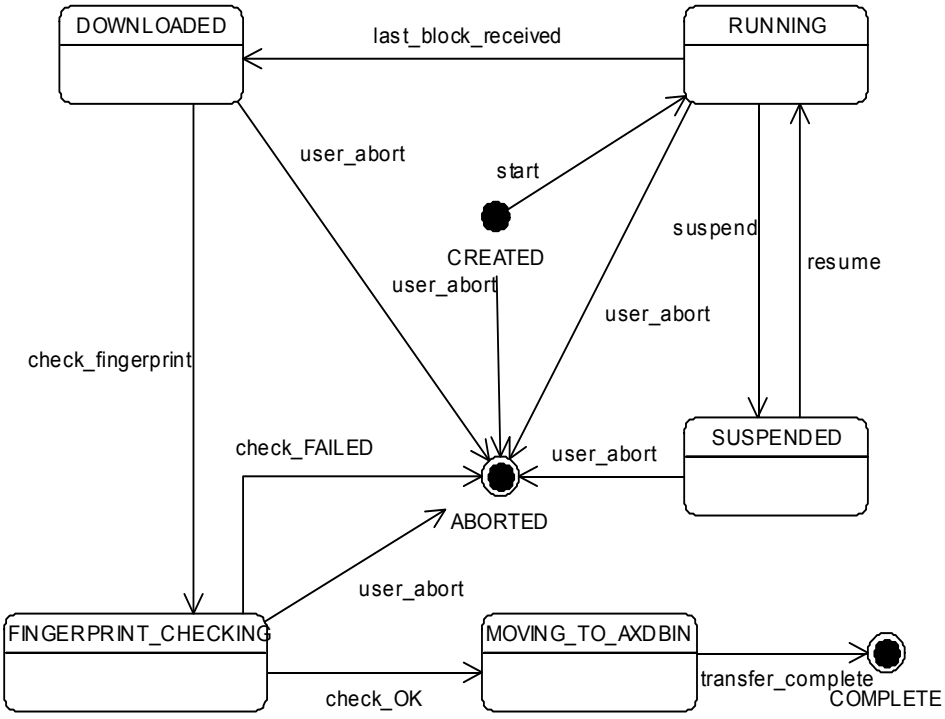
2.10.1 Class Diagram



2.10.2 Download Monitor

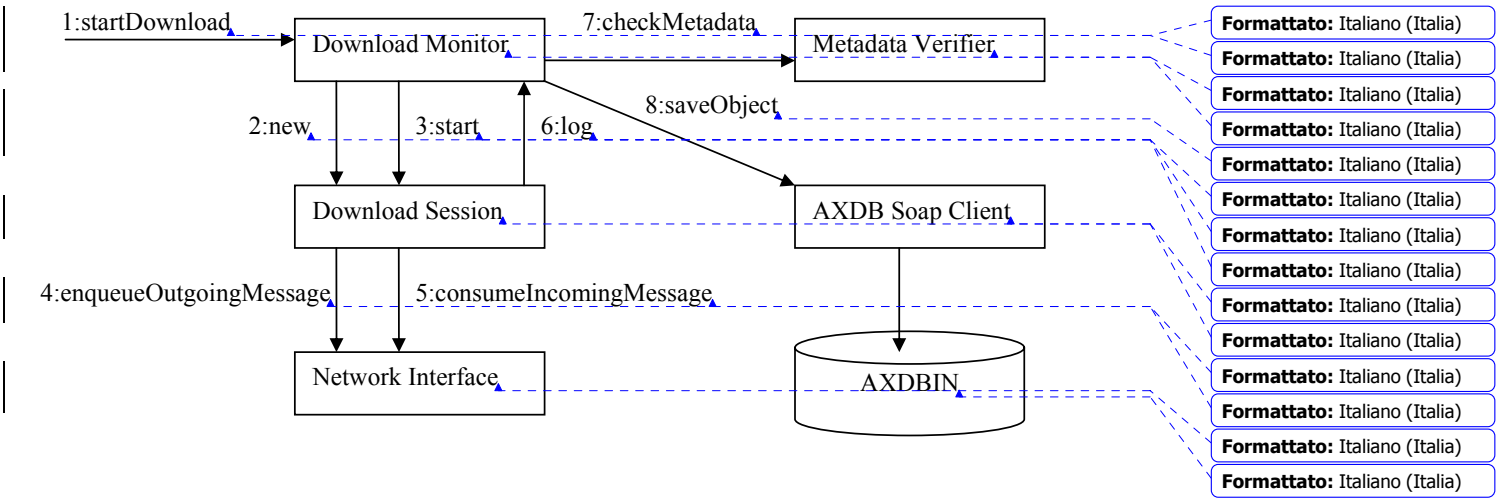
The Download Monitor can be invoked to start a download. It connects to the P2P node via Soap and register itself to receive callback messages from the P2P network. Once a download is started the DownloadMonitor creates a DownloadSession which is commited to store and manage the state of the download.
from a remote peer. The DownloadSession is a concurrent process which sends request of blocks to the underlying P2P file transmission client and keeps track of which part of a file have been received. It must allow the suspend/resume/abort and all operations cause a transition in the DownloadSessionState below.

2.10.2.1 State Diagram: DownloadSessionState



*2.10.2.2 Collaboration Diagram: Downloading an Object**

1 *startDownload*: this method asks the downloading of an object
2 *new*: a new Download Session is created.
3 *start*: starts the session
4 *enqueueOutgoingMessage*: the session sends download requests to the network interface using the appropriate P2P protocol (HTTP, TCP/IP, UDP).
5 *consumeIncomingMessage*: the session waits for incoming messages containing blocks of data addressed to the session.
6 *log*: the session saves downloading events in the Download monitor Logbook
7 *checkMetadata*: when the download is finished the Download Monitor checks the consistency of metadata using the Metadata Verificator
8 *saveObject*: the Download Monitor uses the AXDBIN client to connect to the AXDBIN and to save the new acquired object



Formattato: Italiano (Italia)
Formattato: Italiano (Italia)
Formattato: Italiano (Italia)
Formattato: Italiano (Italia)
Formattato: Italiano (Italia)
Formattato: Italiano (Italia)
Formattato: Italiano (Italia)
Formattato: Italiano (Italia)
Formattato: Italiano (Italia)
Formattato: Italiano (Italia)
Formattato: Italiano (Italia)
Formattato: Italiano (Italia)
Formattato: Italiano (Italia)

2.10.3 The Download Monitor Web Service Interface

The Download Monitor functionalities can be invoked by means of Web Services.
The WSDL Interface for DownloadMonitor is shown in the next picture

listDownloadSessionsIDs();	Returns a list of current downloads represented by their Sessions ID
startDownload(URI objectURI, URI callbackURI)	Creates and starts a new Download Session for a given object and register an optional callback
progress(String sessionID);	Returns a progress information (an integer) about the download progress given its session id
resume(String sessionID);	Resumes a suspended session given its session id
suspend(String sessionID);	Suspends a running session given its session id
abort(String sessionID);	Aborts an session given its session id

```
<?xml version='1.0' encoding='utf-8' ?>
<definitions name='axeptool.axeptool_monitor.DownloadMonitorInterface'
targetNamespace='http://axmedis.org/wsdl/axeptool/axeptool_monitor/'
  xmlns:tns='http://axmedis.org/wsdl/axeptool/axeptool_monitor/'
  xmlns:ns0='http://axmedis.org/xsd/SchemaTypes/'
  xmlns:map='http://axmedis.org/mapping/'
  xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns='http://schemas.xmlsoap.org/wsdl/'>
  <types>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="http://axmedis.org/wsdl/java/net/"
      xmlns:map="http://axmedis.org/mapping/"
      xmlns:tns="http://axmedis.org/wsdl/java/net/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
        <xsd:complexType name="URI">
          <xsd:annotation>
            <xsd:appinfo>
              <map:java-type name="java.net.URI"/>
            </xsd:appinfo>
          </xsd:annotation>
          <xsd:sequence>
            <xsd:element name="absolute" type="xsd:boolean"/>
            <xsd:element name="authority" nillable="true"
              type="xsd:string"/>
            <xsd:element name="fragment" nillable="true"
              type="xsd:string"/>
            <xsd:element name="host" nillable="true" type="xsd:string"/>
            <xsd:element name="opaque" type="xsd:boolean"/>
            <xsd:element name="path" nillable="true" type="xsd:string"/>
            <xsd:element name="port" type="xsd:int"/>
            <xsd:element name="query" nillable="true"
              type="xsd:string"/>
            <xsd:element name="rawAuthority" nillable="true"
              type="xsd:string"/>
            <xsd:element name="rawFragment" nillable="true"
              type="xsd:string"/>
            <xsd:element name="rawPath" nillable="true"
              type="xsd:string"/>
            <xsd:element name="rawQuery" nillable="true"
              type="xsd:string"/>
            <xsd:element name="rawSchemeSpecificPart" nillable="true"
              type="xsd:string"/>
            <xsd:element name="rawUserInfo" nillable="true"
              type="xsd:string"/>
            <xsd:element name="scheme" nillable="true"
              type="xsd:string"/>
            <xsd:element name="schemeSpecificPart" nillable="true"
              type="xsd:string"/>
            <xsd:element name="userInfo" nillable="true"
              type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:schema>

      <xsd:schema elementFormDefault="qualified"
        targetNamespace="http://axmedis.org/xsd/SchemaTypes/"
        xmlns:tns="http://axmedis.org/xsd/SchemaTypes/"
        xmlns:xns4="http://axmedis.org/wsdl/java/net/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <xsd:import namespace="http://axmedis.org/wsdl/java/net/"/>
        <xsd:element name="p0" nillable="true" type="xsd:string"/>
        <xsd:element name="int_Response" type="xsd:int"/>
        <xsd:element name="p0_1" nillable="true" type="xns4:URI"/>
        <xsd:element name="p1" nillable="true" type="xns4:URI"/>
        <xsd:element name="string_Response" nillable="true"
          type="xsd:string"/>
      </xsd:schema>

    </types>
    <message name='DownloadMonitorInterface_abort_Response_Soap'>
      <part name='response' element='ns0:int_Response'/>
    </message>
    <message name='DownloadMonitorInterface_abort__Request_Soap'>
```

```
    <part name='p0' element='ns0:p0'/>
  </message>
  <message name='DownloadMonitorInterface_getState_Response_Soap'>
    <part name='response' element='ns0:int_Response'/>
  </message>
  <message name='DownloadMonitorInterface_getState__Request_Soap'>
    <part name='p0' element='ns0:p0'/>
  </message>
  <message name='DownloadMonitorInterface_progress_Response_Soap'>
    <part name='response' element='ns0:int_Response'/>
  </message>
  <message name='DownloadMonitorInterface_progress__Request_Soap'>
    <part name='p0' element='ns0:p0'/>
  </message>
  <message name='DownloadMonitorInterface_resume_Response_Soap'>
    <part name='response' element='ns0:int_Response'/>
  </message>
  <message name='DownloadMonitorInterface_resume__Request_Soap'>
    <part name='p0' element='ns0:p0'/>
  </message>
  <message name='DownloadMonitorInterface_startDownload_Response_Soap'>
    <part name='response' element='ns0:string_Response'/>
  </message>
  <message name='DownloadMonitorInterface_startDownload__Request_Soap'>
    <part name='p0' element='ns0:p0_1'/>
    <part name='p1' element='ns0:p1'/>
  </message>
  <message name='DownloadMonitorInterface_suspend_Response_Soap'>
    <part name='response' element='ns0:int_Response'/>
  </message>
  <message name='DownloadMonitorInterface_suspend__Request_Soap'>
    <part name='p0' element='ns0:p0'/>
  </message>
  <portType name='DownloadMonitorInterface'>
    <operation name='abort' parameterOrder='p0'>
      <input message='tns:DownloadMonitorInterface_abort__Request_Soap'/>
      <output message='tns:DownloadMonitorInterface_abort_Response_Soap'/>
    </operation>
    <operation name='getState' parameterOrder='p0'>
      <input message='tns:DownloadMonitorInterface_getState__Request_Soap'/>
      <output message='tns:DownloadMonitorInterface_getState_Response_Soap'/>
    </operation>
    <operation name='progress' parameterOrder='p0'>
      <input message='tns:DownloadMonitorInterface_progress__Request_Soap'/>
      <output message='tns:DownloadMonitorInterface_progress_Response_Soap'/>
    </operation>
    <operation name='resume' parameterOrder='p0'>
      <input message='tns:DownloadMonitorInterface_resume__Request_Soap'/>
      <output message='tns:DownloadMonitorInterface_resume_Response_Soap'/>
    </operation>
    <operation name='startDownload' parameterOrder='p0 p1'>
      <input message='tns:DownloadMonitorInterface_startDownload__Request_Soap'/>
      <output message='tns:DownloadMonitorInterface_startDownload_Response_Soap'/>
    </operation>
    <operation name='suspend' parameterOrder='p0'>
      <input message='tns:DownloadMonitorInterface_suspend__Request_Soap'/>
      <output message='tns:DownloadMonitorInterface_suspend_Response_Soap'/>
    </operation>
  </portType>
```

```
  <binding name='DownloadMonitorInterface' type='tns:DownloadMonitorInterface'>
    <soap:binding transport='http://schemas.xmlsoap.org/soap/http' style='document'/>
    <operation name='abort'>
      <map:java-operation name='abort' signature='KExqYXZhL2xhbmcvU3RyaW5nOylJ'/>
      <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/axeptool_monitor/DownloadMonitorInterface#abort?KExqYXZh
L2xhbmcvU3RyaW5nOylJ' style='document'/>
      <input>
        <soap:body parts='p0' use='literal'/>
      </input>
      <output>
        <soap:body parts='response' use='literal'/>
      </output>
    </operation>
    <operation name='getState'>
      <map:java-operation name='getState' signature='KExqYXZhL2xhbmcvU3RyaW5nOylJ'/>
      <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/axeptool_monitor/DownloadMonitorInterface#getState?KExqY
XZhL2xhbmcvU3RyaW5nOylJ' style='document'/>
      <input>
        <soap:body parts='p0' use='literal'/>
      </input>
      <output>
        <soap:body parts='response' use='literal'/>
      </output>
    </operation>
    <operation name='progress'>
      <map:java-operation name='progress' signature='KExqYXZhL2xhbmcvU3RyaW5nOylJ'/>
      <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/axeptool_monitor/DownloadMonitorInterface#progress?KExqY
XZhL2xhbmcvU3RyaW5nOylJ' style='document'/>
      <input>
        <soap:body parts='p0' use='literal'/>
      </input>
      <output>
        <soap:body parts='response' use='literal'/>
      </output>
    </operation>
    <operation name='resume'>
      <map:java-operation name='resume' signature='KExqYXZhL2xhbmcvU3RyaW5nOylJ'/>
      <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/axeptool_monitor/DownloadMonitorInterface#resume?KExqYX
ZhL2xhbmcvU3RyaW5nOylJ' style='document'/>
      <input>
        <soap:body parts='p0' use='literal'/>
      </input>
      <output>
        <soap:body parts='response' use='literal'/>
      </output>
    </operation>
    <operation name='startDownload'>
      <map:java-operation name='startDownload'
signature='KExqYXZhL25ldC9VUkk7TGphdmEvbmV0L1VSSTspTGphdmEvbGFuZy9TdHJpbmc7'/>
      <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/axeptool_monitor/DownloadMonitorInterface#startDownload?K
ExqYXZhL25ldC9VUkk7TGphdmEvbmV0L1VSSTspTGphdmEvbGFuZy9TdHJpbmc7' style='document'/>
      <input>
        <soap:body parts='p0 p1' use='literal'/>
      </input>
```

```
        <output>
          <soap:body parts='response' use='literal'/>
        </output>
      </operation>
      <operation name='suspend'>
        <map:java-operation name='suspend' signature='KExqYXZhL2xhbmcvU3RyaW5nOylJ'/>
        <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/axeptool_monitor/DownloadMonitorInterface#suspend?KExqY
XZhL2xhbmcvU3RyaW5nOylJ' style='document'/>
        <input>
          <soap:body parts='p0' use='literal'/>
        </input>
        <output>
          <soap:body parts='response' use='literal'/>
        </output>
      </operation>
  </binding>
  <service name='DownloadMonitorInterface'>
    <port name='DownloadMonitorInterface' binding='tns:DownloadMonitorInterface'>
      <soap:address location='http://server:6060/DownloadMonitorInterface/'/>
    </port>
  </service>
</definitions>
```

2.10.4 Metadata Verificator

This module is to verify the certification of the AXInfo in the way to avoid the listing of inconsistent objects in terms of metadata and content referred. Any object has to be certified in the phase of publication and here is neded to verify the certification.

2.10.5 Upload Monitor

As shown in the previous section and picture, this module tracks uploads status for AXMEDIS Objects, it can make use of AXMEDIS Object Manager for Object management.
Also the UploadMonitor as a WebService part which exposes some useful methods for remote controlling.
The interface is:

listUploadSessionsIDs();	Returns a list of current uploads represented by their Upload Sessions ID
progress(String sessionID);	Returns a progress information (an integer) about the upload progress given its session id
resume(String sessionID);	Resumes a suspended upload given its session id
suspend(String sessionID);	Suspends a running upload given its session id
abort(String sessionID);	Aborts an upload given its session id

```
<?xml version='1.0' encoding='utf-8' ?>
<definitions name='axeptool.axeptool_monitor.UploadMonitorInterface'
targetNamespace='http://axmedis.org/wsdl/axeptool/axeptool_monitor/'
  xmlns:tns='http://axmedis.org/wsdl/axeptool/axeptool_monitor/'
  xmlns:ns0='http://axmedis.org/xsd/SchemaTypes/'
  xmlns:map='http://axmedis.org/mapping/'
  xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns='http://schemas.xmlsoap.org/wsdl/'>
```

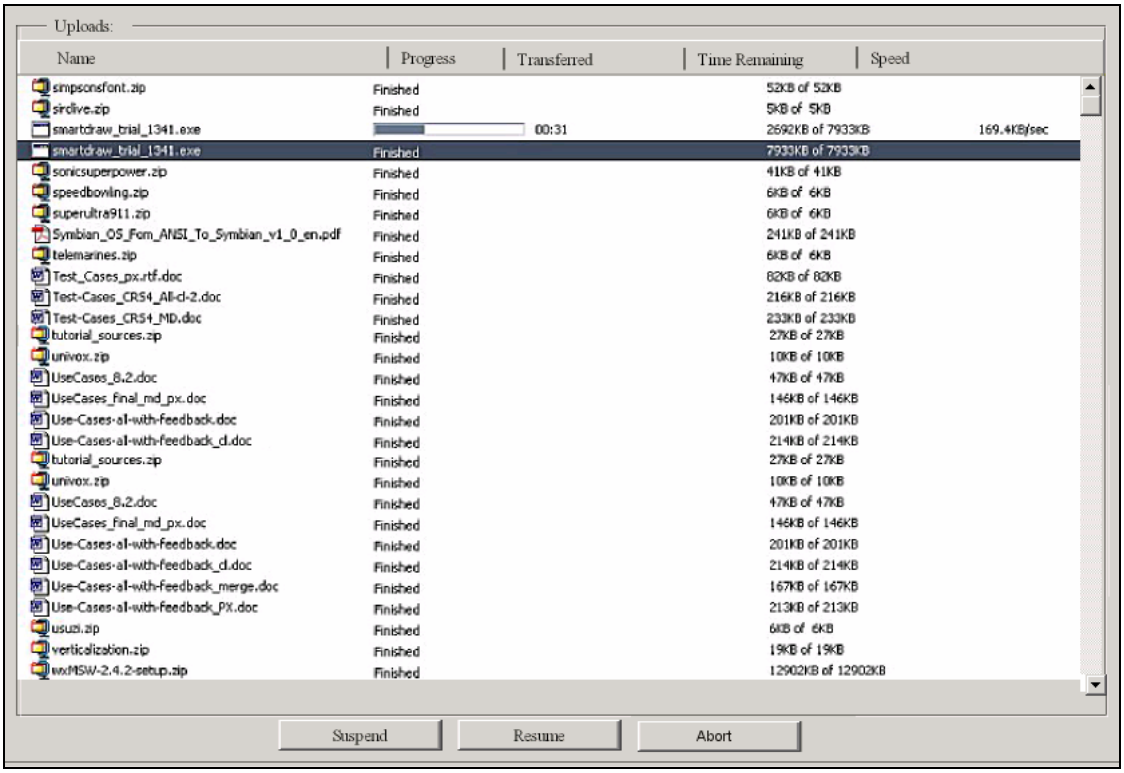
```
<types>
  <xsd:schema elementFormDefault="qualified"
    targetNamespace="http://axmedis.org/xsd/SchemaTypes/"
    xmlns:tns="http://axmedis.org/xsd/SchemaTypes/"
    xmlns:xns4="http://axmedis.org/wsdl/java/lang/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:import namespace="http://axmedis.org/wsdl/java/lang/"/>
    <xsd:element name="p0" nillable="true" type="xsd:string"/>
    <xsd:element name="int_Response" type="xsd:int"/>
    <xsd:element name="ArrayOfstring_Response" nillable="true"
      type="xns4:ArrayOfstring"/>
  </xsd:schema>

  <xsd:schema elementFormDefault="qualified"
    targetNamespace="http://axmedis.org/wsdl/java/lang/"
    xmlns:map="http://axmedis.org/mapping/"
    xmlns:tns="http://axmedis.org/wsdl/java/lang/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="ArrayOfstring">
      <xsd:annotation>
        <xsd:appinfo>
          <map:java-type name="[Ljava.lang.String;"/>
        </xsd:appinfo>
      </xsd:annotation>
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
          name="string" nillable="true" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>

</types>
<message name='UploadMonitorInterface_abort_Response_Soap'>
  <part name='response' element='ns0:int_Response'/>
</message>
<message name='UploadMonitorInterface_abort__Request_Soap'>
  <part name='p0' element='ns0:p0'/>
</message>
<message name='UploadMonitorInterface_listUploadSessionsIDs_Response_Soap'>
  <part name='response' element='ns0:ArrayOfstring_Response'/>
</message>
<message name='UploadMonitorInterface_listUploadSessionsIDs__Request_Soap'/>
<message name='UploadMonitorInterface_progress_Response_Soap'>
  <part name='response' element='ns0:int_Response'/>
</message>
<message name='UploadMonitorInterface_progress__Request_Soap'>
  <part name='p0' element='ns0:p0'/>
</message>
<message name='UploadMonitorInterface_resume_Response_Soap'>
  <part name='response' element='ns0:int_Response'/>
</message>
<message name='UploadMonitorInterface_resume__Request_Soap'>
  <part name='p0' element='ns0:p0'/>
</message>
<message name='UploadMonitorInterface_suspend_Response_Soap'>
  <part name='response' element='ns0:int_Response'/>
</message>
<message name='UploadMonitorInterface_suspend__Request_Soap'>
  <part name='p0' element='ns0:p0'/>
```

```
  </message>
  <portType name='UploadMonitorInterface'>
    <operation name='abort' parameterOrder='p0'>
      <input message='tns:UploadMonitorInterface_abort__Request_Soap'/>
      <output message='tns:UploadMonitorInterface_abort_Response_Soap'/>
    </operation>
    <operation name='listUploadSessionsIDs'>
      <input message='tns:UploadMonitorInterface_listUploadSessionsIDs__Request_Soap'/>
      <output message='tns:UploadMonitorInterface_listUploadSessionsIDs_Response_Soap'/>
    </operation>
    <operation name='progress' parameterOrder='p0'>
      <input message='tns:UploadMonitorInterface_progress__Request_Soap'/>
      <output message='tns:UploadMonitorInterface_progress_Response_Soap'/>
    </operation>
    <operation name='resume' parameterOrder='p0'>
      <input message='tns:UploadMonitorInterface_resume__Request_Soap'/>
      <output message='tns:UploadMonitorInterface_resume_Response_Soap'/>
    </operation>
    <operation name='suspend' parameterOrder='p0'>
      <input message='tns:UploadMonitorInterface_suspend__Request_Soap'/>
      <output message='tns:UploadMonitorInterface_suspend_Response_Soap'/>
    </operation>
  </portType>
  <binding name='UploadMonitorInterface' type='tns:UploadMonitorInterface'>
    <soap:binding transport='http://schemas.xmlsoap.org/soap/http' style='document'/>
    <operation name='abort'>
      <map:java-operation name='abort' signature='KExqYXZhL2xhbmcvU3RyaW5nOylJ'/>
      <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/axeptool_monitor/UploadMonitorInterface#abort?KExqYXZhL2
xhbmcvU3RyaW5nOylJ' style='document'/>
      <input>
        <soap:body parts='p0' use='literal'/>
      </input>
      <output>
        <soap:body parts='response' use='literal'/>
      </output>
    </operation>
    <operation name='listUploadSessionsIDs'>
      <map:java-operation name='listUploadSessionsIDs'
signature='KClbTGphdmEvbGFuZy9TdHJpbmc7'/>
      <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/axeptool_monitor/UploadMonitorInterface#listUploadSessionsI
Ds?KClbTGphdmEvbGFuZy9TdHJpbmc7' style='document'/>
      <input>
        <soap:body use='literal'/>
      </input>
      <output>
        <soap:body parts='response' use='literal'/>
      </output>
    </operation>
    <operation name='progress'>
      <map:java-operation name='progress' signature='KExqYXZhL2xhbmcvU3RyaW5nOylJ'/>
      <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/axeptool_monitor/UploadMonitorInterface#progress?KExqYXZ
hL2xhbmcvU3RyaW5nOylJ' style='document'/>
      <input>
        <soap:body parts='p0' use='literal'/>
      </input>
      <output>
```

```
            <soap:body parts='response' use='literal'/>
         </output>
      </operation>
      <operation name='resume'>
         <map:java-operation name='resume' signature='KExqYXZhL2xhbmcvU3RyaW5nOylJ'/>
         <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/axeptool_monitor/UploadMonitorInterface#resume?KExqYXZh
L2xhbmcvU3RyaW5nOylJ' style='document'/>
         <input>
            <soap:body parts='p0' use='literal'/>
         </input>
         <output>
            <soap:body parts='response' use='literal'/>
         </output>
      </operation>
      <operation name='suspend'>
         <map:java-operation name='suspend' signature='KExqYXZhL2xhbmcvU3RyaW5nOylJ'/>
         <soap:operation
soapAction='http://axmedis.org/wsdl/axeptool/axeptool_monitor/UploadMonitorInterface#suspend?KExqYXZ
hL2xhbmcvU3RyaW5nOylJ' style='document'/>
         <input>
            <soap:body parts='p0' use='literal'/>
         </input>
         <output>
            <soap:body parts='response' use='literal'/>
         </output>
      </operation>
   </binding>
   <service name='UploadMonitorInterface'>
      <port name='UploadMonitorInterface' binding='tns:UploadMonitorInterface'>
         <soap:address location='http://server:6060/UploadMonitorInterface/'/>
      </port>
   </service>
</definitions>
```

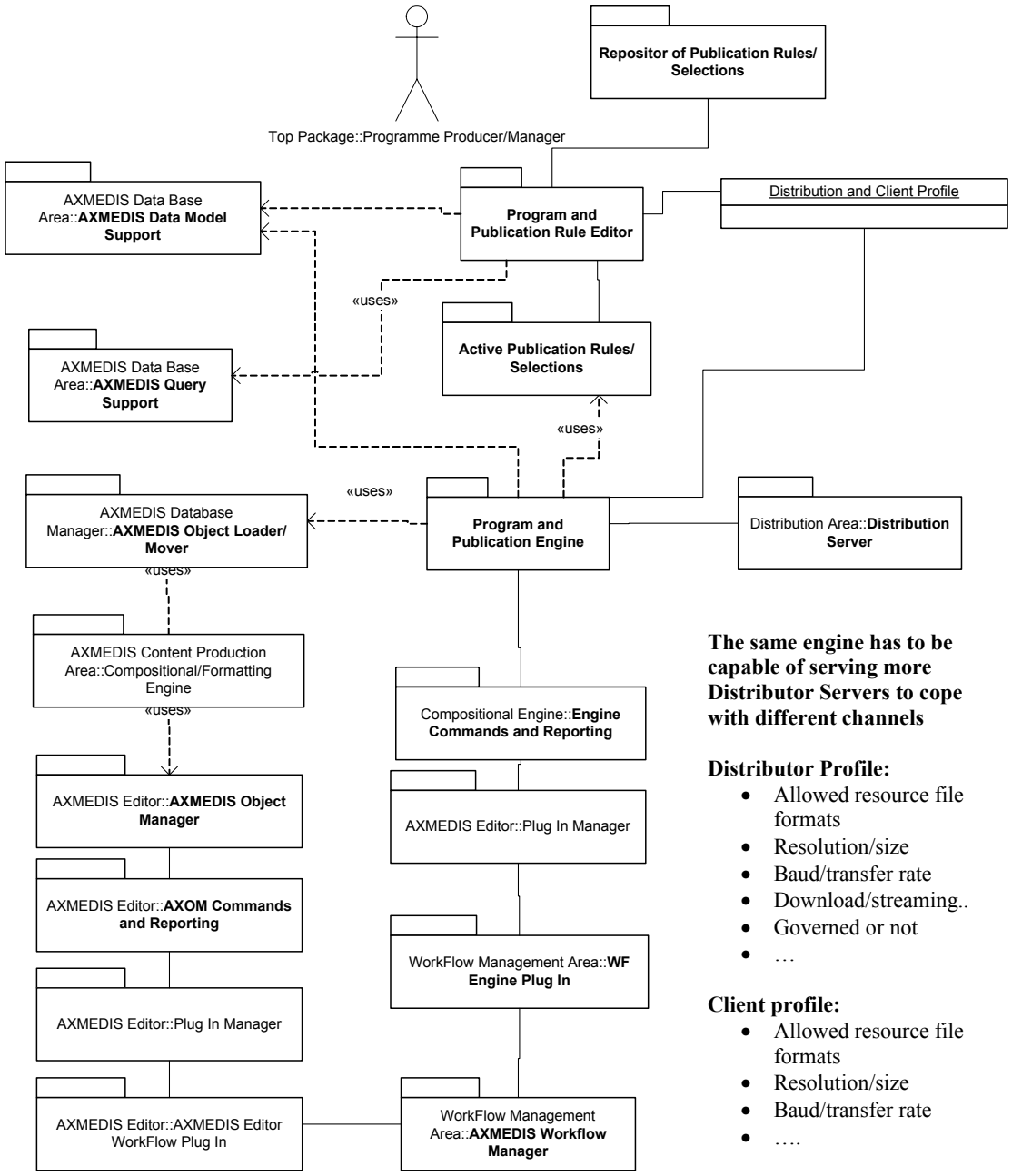
2.10.6 Upload Monitor GUI

The GUI visualizes its data in real-time modality, giving a clear status representation for the user of AXEPTool. Possible representations can be: a diagram, a percentage progress bar or other



3 Program and Publication (P&P) Area (WP5.4.5: UNIVLEEDS)

Programme and Publication Area



The same engine has to be capable of serving more Distributor Servers to cope with different channels

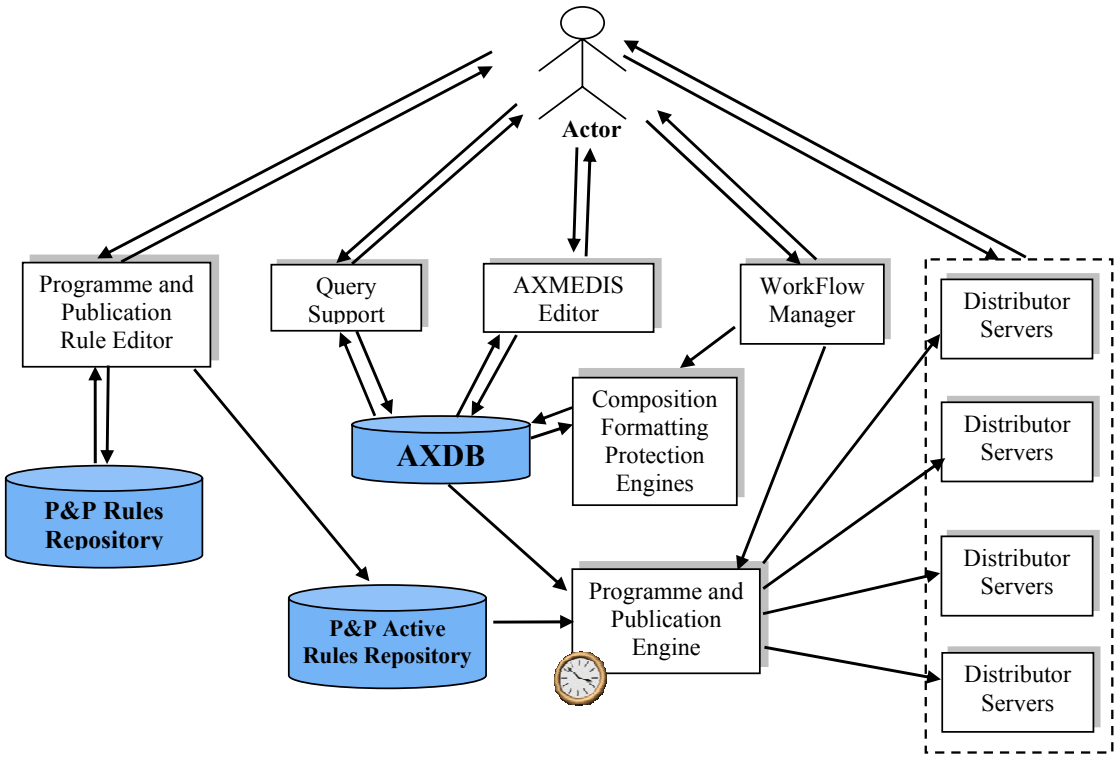
Distributor Profile:

- Allowed resource file formats
- Resolution/size
- Baud/transfer rate
- Download/streaming..
- Governed or not
- …

Client profile:

- Allowed resource file formats
- Resolution/size
- Baud/transfer rate
- ….

3.1 Programme and Publication Area Overview



3.2 Programme and Publication Programme and Rule Format

The Programme and Publication process is driven by P&P programme to carry out a set of rules as specified by the Actor using the P&P Editor, to delivery specific AXMEDIS objects to the distribution channel as specified by the Actor. The P&P process also allows on-demand requests that responds to the distribution and end user needs.
The rule has to:

- Describe the contents of the rule: AXMEDIS ObjectID or new AXMEDIS ObjectID if formatting is required
- Describe distribution channel properties etc.
- Describe time relationships
- And can handle Query Selection.

In this section the structure of a programme is described. A P&P Programme consists of a list of rules. A programme can be divided into three main sections:

1. Header – contains metadata related to the general information associated with the programme.
2. Schedule – contains the information for the activation of the programme.
3. Definition – contains the P&P rules section with detail information on the delivery of one or more AXMEDIS object, including start-time, date, duration, optional query selection, etc.

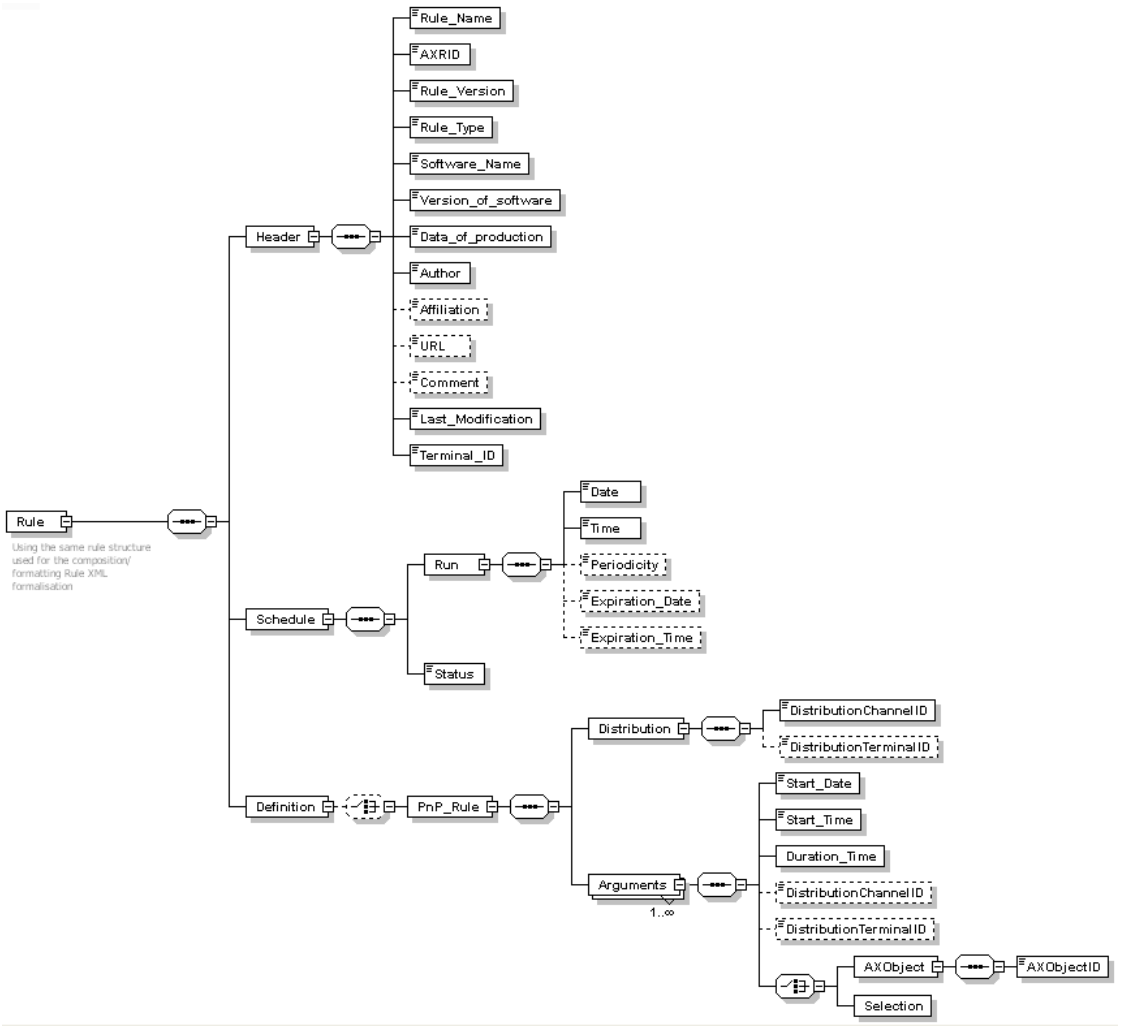
Header				
Data	**Type**	**Values/Format**	**Description**	**Issues**
Rule Name	String	e.g. "Settop Box Programme"	It defines the name of the rule	
AXRID	String		It defines the AXMEDIS Rule ID	
Rule Version	String	e.g. "1.0"	It defines the version of the rule	

Rule Type	String	"Programme",	It defines the type of rule	This field could be used with other types of rules as discussed in DE3-1-2 Content Production, "Composition/ formatting Rule General Format"
Software Name	String	"Axmedis P&P Editor"	It specifies the name of software used	
Version of software	String	e.g. "2.0"	It defines the version of software used.	
Date of production	Date	dd.mm.yy	It defines when the rule has been created	
Time of production	Time	hh.mm.ss am/pm	It defines at what time the rule has been created	
Author	String	e.g. "John Brown"	It defines the name of author who has created the rule	
Affiliation	String	e.g. "UNIVLEEDS"	It defines the name of Affiliation	
URL	String	e.g. "http://www...."	It defines the Internet address of the Affiliation	
Comment	String		It allows describing what the rule does	
Last_Modification	Date		Who is last modified	
Terminal_ID	String		Where	
Schedule				
Data	**Type**	**Values/Format**	**Description**	**Issues**
Run	Section		It defines a subsection of metadata that describe information needed for scheduling the execution of the objects within the programme.	
Status	String	"Active", "Inactive", "Trial"	It defines if a rule: • is active and can be executed • Trial • is inactive	The list of status identifiers could be extended if it is necessary.
Run				
Data	**Type**	**Values/Format**	**Description**	**Issues**
Start Date	Date	dd.mm.yy	It defines when the (programme) rule has to be executed by the engine in terms of day, month and year.	
Start Time	Time	hh.mm.ss am/pm	It defines when the rule (programme) has to be executed by the engine in terms of the time clock.	
Periodicity	String	"Monthly", "Daily", "Weekly", etc…	It defines if a rule has to be executed periodically	Optional
Expiration data	Date	dd.mm.yy	To stop the periodicity	Optional
Expiration time	Time	hh.mm.ss am/pm	To stop the periodicity	Optional
Definition – P&P_Rule – Distribution				
DistributionChannelID	String	String	Defines the distribution channel the programme rules are required to be distributed to.	
DIstributionTerminalID			Which distribution terminal the	Optional

			programme rules are to be delivered to	
Definition – P&P_Rule – Arguments				
Start Date	Date	dd.mm.yy	It defines when the P&P rule has to be executed by the engine in terms of day, month and year.	
Start Time	Time	hh.mm.ss am/pm	It defines when the P&P rule has to be executed by the engine in terms of the time clock.	
Duration Time	Time	hh.mm.ss	It defines how long the AXObject or Selection required to complete execution	
DistributionChannelID	String	String	Defines the distribution channel the object is required to be distributed to over-riding header distribution channel	Optional
DIstributionTerminalID			Which distribution terminal is the programme to be delivered to over-riding header distribution terminal	Optional

3.2.1 Programme and Publication Rule XML formalisation

The set of metadata defined previously formalised by means of the following XML Schema:



The schema for the P&P Programme uses the same schema as content production with the same Header and Schedule. The rules of the programme are specified in the Definition section under "PnP_Rule" providing the DistributionChannelID, DistributionTerminalID (to enable on-demand operation to a specific terminal), start-date, start–time and duration of AXObject or Selection. Following is the schema for the P&P programme: By using the same header, this allow better integration practising re-usability and consistency.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 sp1 U (http://www.xmlspy.com) by Royce Neagle (ICSRiM, University of Leeds) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="Rule">
    <xs:annotation>
      <xs:documentation>Same rule structure used for the composition/ formatting Rule XML formalisation</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Header">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Rule_Name" type="xs:string"/>
              <xs:element name="AXRID" type="xs:string"/>
```

```
        <xs:element name="Rule_Version" type="xs:string"/>
        <xs:element name="Rule_Type">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Compositional"/>
              <xs:enumeration value="Formatting"/>
              <xs:enumeration value="CompForml"/>
              <xs:enumeration value="ProgPub"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="Software_Name" type="xs:string"/>
        <xs:element name="Version_of_software" type="xs:string"/>
        <xs:element name="Data_of_production" type="xs:date"/>
        <xs:element name="Author" type="xs:string"/>
        <xs:element name="Affiliation" type="xs:string" minOccurs="0"/>
        <xs:element name="URL" type="xs:anyURI" minOccurs="0"/>
        <xs:element name="Comment" type="xs:string" minOccurs="0"/>
        <xs:element name="Last_Modification" type="xs:date"/>
        <xs:element name="Terminal_ID" type="xs:ID"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Schedule">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Run">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Date" type="xs:date"/>
              <xs:element name="Time" type="xs:time"/>
              <xs:element name="Periodicity" minOccurs="0">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="Daily"/>
                    <xs:enumeration value="Weekly"/>
                    <xs:enumeration value="Monthly"/>
                    <xs:enumeration value="Yearly"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="Expiration_Date" type="xs:date" minOccurs="0"/>
              <xs:element name="Expiration_Time" type="xs:time" minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Status">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Active"/>
              <xs:enumeration value="Inactive"/>
              <xs:enumeration value="Draft"/>
              <xs:enumeration value="FullTrial"/>
              <xs:enumeration value="QuickTrial"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Definition">
    <xs:complexType>
      <xs:choice minOccurs="0">
        <xs:element name="PnP_Rule">
          <xs:complexType>
            <xs:sequence>
```

```
                <xs:element name="Distribution">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="DistributionChannelID" type="xs:string"/>
                      <xs:element name="DistributionTerminalID" type="xs:string" minOccurs="0"/>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="Arguments" maxOccurs="unbounded">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="Start_Date" type="xs:date"/>
                      <xs:element name="Start_Time" type="xs:time"/>
                      <xs:element name="Duration_Time"/>
                      <xs:element name="DistributionChannelID" type="xs:string" minOccurs="0"/>
                      <xs:element name="DistributionTerminalID" type="xs:string" minOccurs="0"/>
                      <xs:choice>
                        <xs:element name="AXObject">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element name="AXObjectID" type="xs:string"/>
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                        <xs:element name="Selection">
                          <xs:annotation>
                            <xs:documentation>See the query Support Specification for more details</xs:documentation>
                          </xs:annotation>
                          <xs:complexType/>
                        </xs:element>
                      </xs:choice>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
</xs:schema>
```

The following example shows the XML structure for a P&P rule:

```
<Rule>
   <Header>
      <Rule_Name> Name of Rule </Rule_Name>
      <AXRID> 0010001 </AXRID>
      <Rule_Version> 1.0 </Rule_Version>
      <Rule_Type> ProgPub </Rule_Type>
      <Software_Name> Axmedis P&P Editor </Software_Name>
      <Version_of_software> 1.0 </Version_of_software>
      <Date_of_production> 29.01.2005 </Date_of_production>
      <Time_of_production> 12:00 am </Time_of_production>
      <Author> John Brown </Author>
      <Affiliation> UNIVLEEDS </Affiliation>
      <URL> http://www.icsrim.org.uk </URL>
      <Comment> Publish selection of media for set top boxes </Comment>
      <Last_Modifications> 30.01.2005 </Last_Modifications>
      <Terminal_ID> </Terminal_ID>
   </Header>
   <Schedule>
      <Run>
```

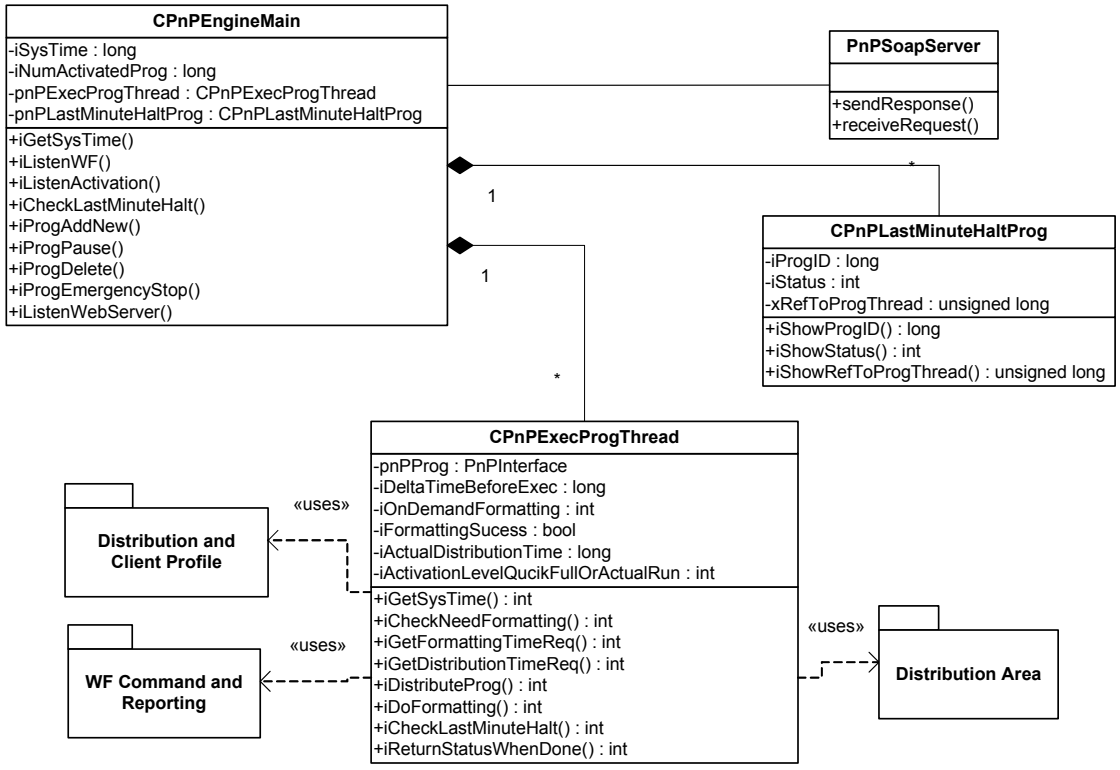
```
         <Date> 01.02.2005 <Date>
         <Time> 12:00 pm </Time>
         <Periodicity> Weekly </Periodicity>
         <Expiration_Date> 31.01.2006 </Expiration_Date>
         <Expiration_Time> 12:00 pm </Expiration_Time>
      </Run>
      <Status> Active </Status>
   </Schedule>
   <Definition>
      <PnP_Rule>
         <Distribution>
            <DistributionChannelID> Satelite12345678 </DistributionChannelID>
         </Distribution>
         <Arguments>
            <Start_Date>01.02.2005</Start_Date>
            <Start_Time>12:00pm</Start_Time>
            <Selection>
               One…..
               <query> …… </query>
               <query> …… </query>
               Object 1435325, Object 24243, Object 45670743
            </Selection>
      --- compliant with the definition of SELECTION, see part of database ---
         <Arguments>
         <Arguments>
            <Start_Date>01.02.2005</Start_Date>
            <Start_Time>15:00pm</Start_Time>
            <Duration>02:28:45</Duration>
            <DistributionChannelID> Mobile456123 </DistributionChannelID>
            <DistributionTerminalID> 129.168.0.1 </DistributionTerminalID>
            <AXObject>
               <AXObjectID>Objects1435325</AXObjectID>
            </AXObject>
         <Arguments>
      </PnP_Rule>
   </Definition>
</Rule>
```

3.3 Programme and Publication Engine (UNIVLEEDS)

Module Profile		
Programme and Publicaiton Engine		
Executable or Library(Support)	Executable	
Single Thread or Multithread	Multithreaded	
Language of Development	C++	
Responsible Name	Kia Ng and Royce Neagle	
Responsible Partner	UNIVLEEDS	
Status (proposed/approved)	Proposed	
Platforms supported	Windows XP	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
Active publication rules	Simple library	
Distribution Server	API	
Distribution and Client Profile	API	
Object Loader/Mover	API	
File Formats Used	Shared with	File format name or reference to a section

XML plus schema	Active Publication Rules	Programme and Publication Format with extension .pp
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
None	Visual Studio .NET	STL
	C++	Windows XP
		Windows Threads
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
C++	Standard C++ library including STL	
XML Parser	Xerces-c++ v-2.6.0	Apache Software Licence, v-2.0

3.3.1 Class Diagram



The Programme and Publication Engine will be developed exploiting the work performed for the Publication tool in WP4.4. This will allow the reception of specific commands (requests) for creating content produced by exploiting the capabilities of the AXMEDIS formatting engine via Workflow. In addition, the Programme and Publication Engine will also have the capabilities for producing the programme based on the specific rules.

The active engine is continuous running software accessing the system clock to process a list of programmes, which consists of "rules" to make available AXMEDIS objects to the specified destination channels at the correct time, taking into account the transfer and/or formatting (if required) time. This is achieved by the input of *activated* rules for scheduled distribution.

The active Programme and Publications Engine's main function is to continually run looking for active publication rules and make the objects in the rules available for distribution. The main points to consider:

- Access to correct system clock
- Keep a track of newly activated rules from the Active Publication Rules to add to the delivery system
- The link to the AXMEDIS formatting engine via Workflow to request the appropriate format for distribution and retrieving the object
- Submitting the objects to the AXMEDIS distribution server allowing for delivery time computed using data from the distribution channel profiles

In order to activate an on-demand operation, a request is sent from WF to the P&P engine with a valid P&P programme. On demand programmes will be processed immediately and send to the distribution server.

For the delivery of the AXMEDIS object by the P&P Engine, the P&P Engine requests the AXMEDIS object using its ID to WF, and the WF is to provide the AXMEDIS object specified by returning an URI of the object. In order to perform provide appropriate object for consumsion, the P&P requests profiles information (for Both the client and the distribution channel) so that compability issues can be reviewed and if naccessary, formatting engine can be contacted to performe formatting operations before sending the final object.

For each new item in a programme, a check has to be carried out to see if the object in question is "compatible" with the destination profile. The check could be done when new programme is "activated" by the actor so that the Engine knows this additional requirements and can allow for necessary formatting time. Non-compatible objects require the Programme and Publication Engine to interface with the Formatting engine (via workflow) to provide the appropriate processing to reformat the object for distribution. For example, the AXMEDIS object in the database could be for HDTV and the programme producer has requested it to be used on a PDA. The Programme and Publication Engine provides the Formatting engine a reference to the AXMEDIS object and the destination profile.

The Formatting engine is expected to receive

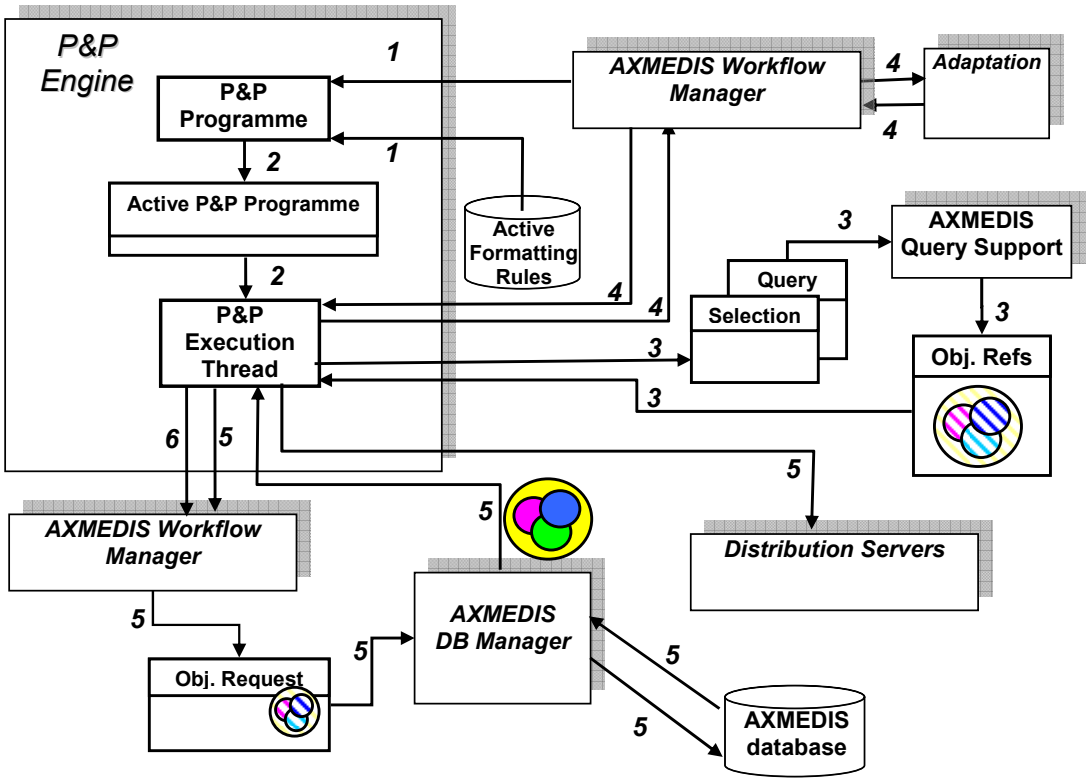
- An AXMEDIS object ID and the formatting requirements
- Process/convert/etc the object so that it is compatible to the destination profile
- Return the new formatted acmedis object ID.
- If requesting a quick trial, the formatting engine is expected to return whether formatting is possible for the P&P programme and not a new object ID

The formatting engine return the new AXOID for the formatted object to P&P via WF.

The communication between P&P and the formatting engine is via WF based on Web Services (GSOAP).
When P&P request the formatting engine, the request should be accompany by a P&P ID so that the return of the result can be identified.
Workflow is responsible for sending the formatting commandfrom P&P defined as follow:

CallFormatEng(**ProcessingFuncID**, "xml-of-the-formatting-req", WhoCalled);

Scenario on Programme and Publication Engine



Scenario on Programme and Publication – P&P Engine

Formattato: Giustificato, Regola lo spazio tra testo asiatico e in alfabeto latino, Regola lo spazio tra caratteri asiatici e numeri

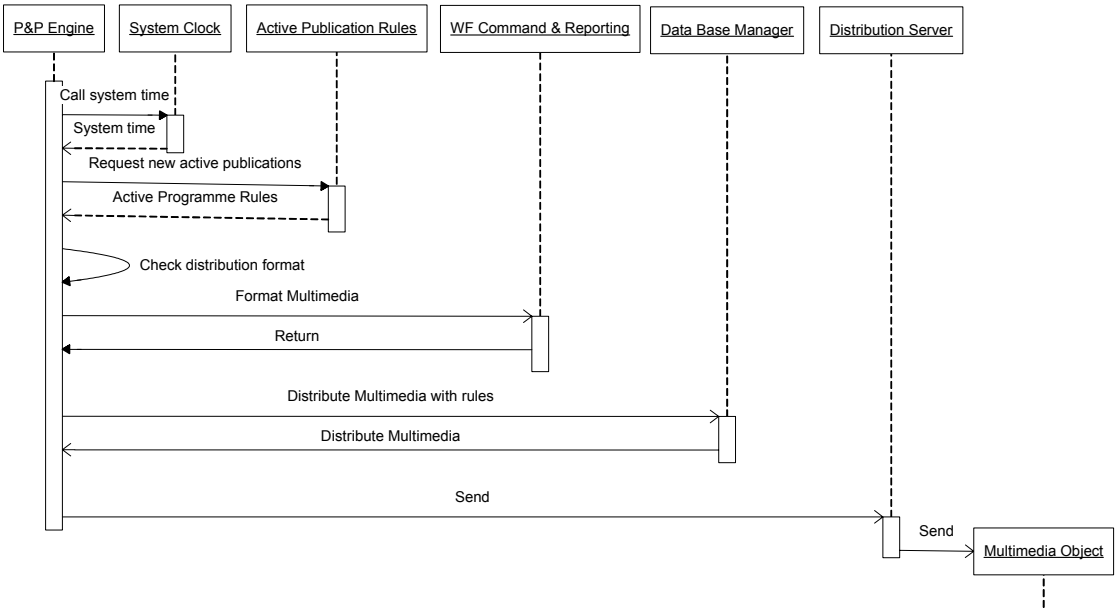
Scenario Description:

1. **Start Publication Process.** The P&P Engine receives a publication request coming from the AXMEDIS Workflow Manager or the internal scheduler activates a rule from the Active Composition Rules.
2. **Programme Execution Request.** The scheduler initiates a P&P Programme thread executing the corresponding rule.
3. **AXMEDIS Programme Objects selection**. For each selection and/or query specified in the Programme, the programme thread sends queries to the AXMEDIS Query Support to obtain references to AXMEDIS objects that match the request.
4. **Adaptation request**. If the Object format does not match the target format, a request is performed via Workflow in order to adapt objects according to the distribution profile specified in the programme and return the new Object reference.
5. **Object Distribution.** The object is requested from Workflow using URI and transferred to distribution server using FTP. For saterlite channel the notification of the transfer will be carried out by using an A.P.I. Server
6. **End process notification**. The End of the rule is notified to the AXMEDIS Workflow Manager.

3.3.2 Sequence Diagram: *Publication using the Active Rules Repository*

Scenario description for activating an existing P&P programme:

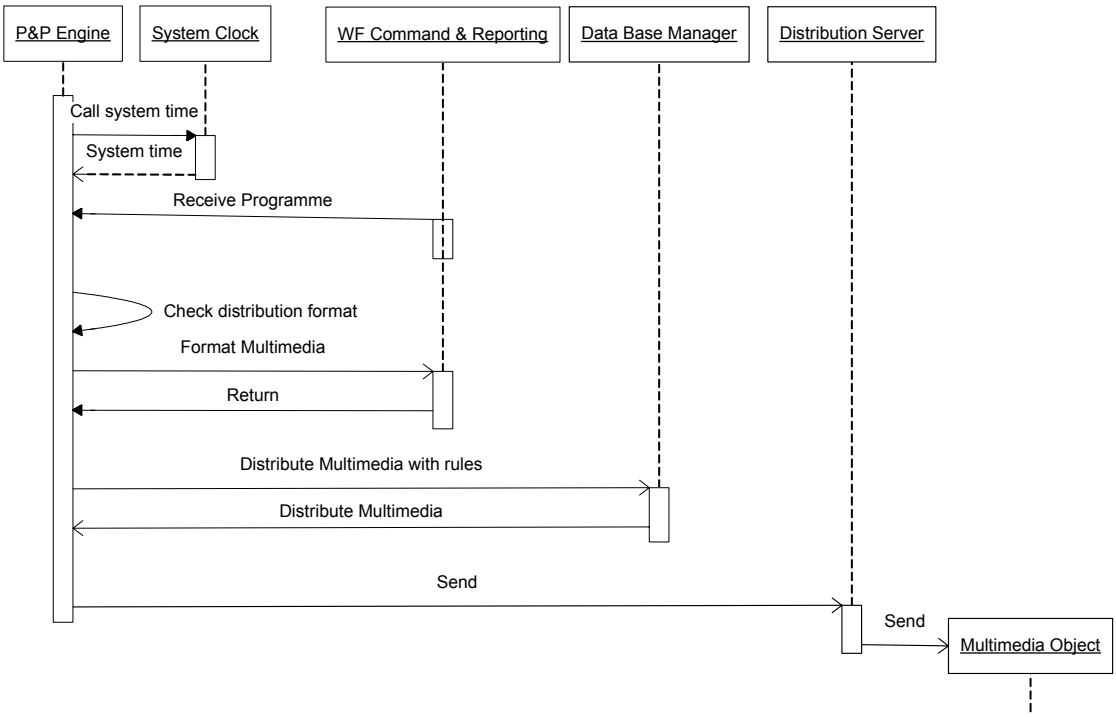
1. The Engine using the system clock requests active P&P programme
2. If format is different for object and distribution profile, using WF Command and Reporting, a request is made for the object to be formatted and a new Object reference returned
3. At correct time the objects are retrieved from the database manager and sent to the appropriate distribution server.



3.3.3 Sequence Diagram: *On-demand publication from Workflow*

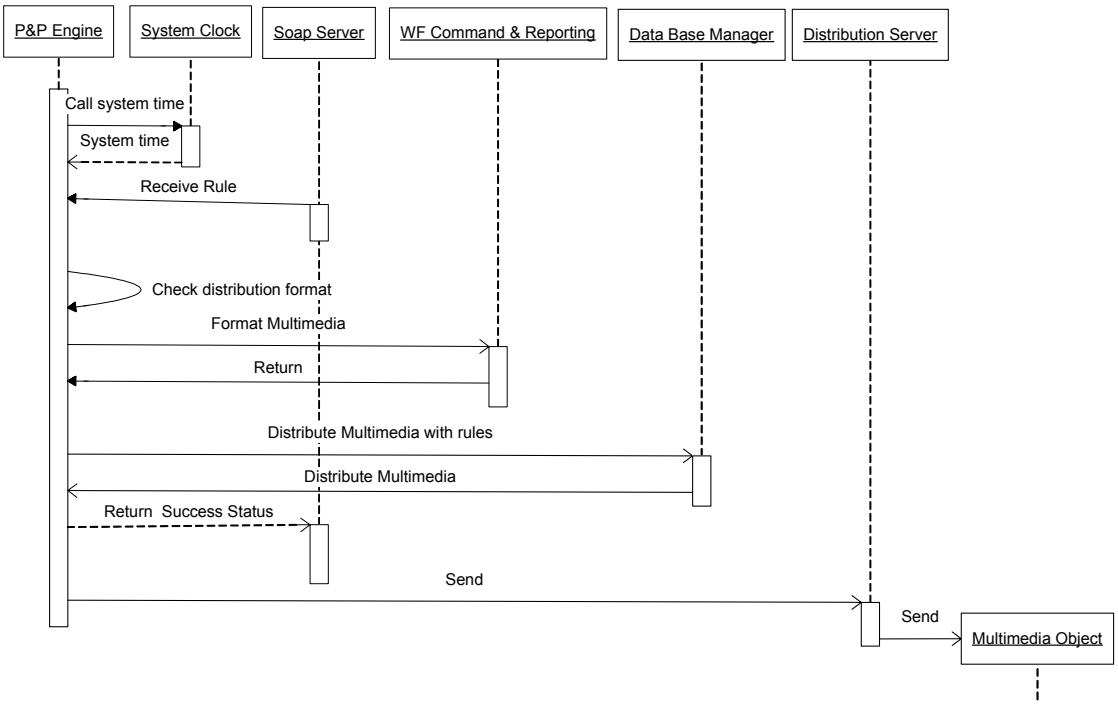
Scenario description for activating an existing P&P programme:

1. The Engine receives an on-demand programme from work flow
2. If format is different for object and distribution profile, using WF Command and Reporting, a request is made for the object to be formatted and a new Object reference returned
3. At engine using the system correct time the objects are retrieved from the database manager and sent to the appropriate distribution server.



3.3.4 Sequence Diagram: *On-demand publication from the Soap Server*

1. The Engine receives an on-demand programme from the SOAP server
2. If format is different for object and distribution profile, using WF Command and Reporting, a request is made for the object to be formatted and a new Object reference returned
3. At engine using the system correct time the objects are retrieved from the database manager and sent to the appropriate distribution server.



3.3.5 Internal Scheduler

The internal scheduler is the manager of active programme. It has to detect, fire, launch and manage the execution of a rule. During its activity, the internal scheduler has to:

1. preserve the scheduled work from interruption of service (crash of the application) giving the possibility to restore the last status of activity
2. manage and update the list of rules to be scheduled and their status
3. notify to the AXMEDIS Workflow Manager messages due to:
   - errors due to the launching phase

The functionalities provided by the programme scheduler are:

- Select from the internal scheduled programmes, the programme thread is executed. This is performed by:
  - checking the execution time and date
  - receiving an immediate run command from the AXMEDIS Workflow Manager
- Modify and set the time resolution for the control of programme thread
- Add a new submitted programme with consist of a list of rules
  - Loading the corresponding rule xml file from the P&P active repository directory
  - Extracting the metadata for scheduling
  - Generating and assigning a thread ID to the programme
- Remove a programme from the list of threads
- Check expiration conditions of a programme
- Update firing conditions of a periodic programme
- Modify the status of the programme
- Restore the last status by loading the backup copy of the list of threads

3.3.6 P&P Engine command and reporting

It provides services to the AXMEDIS Workflow Manager and P&P Rules Editor. This module provides communication functionalities and allows managing:

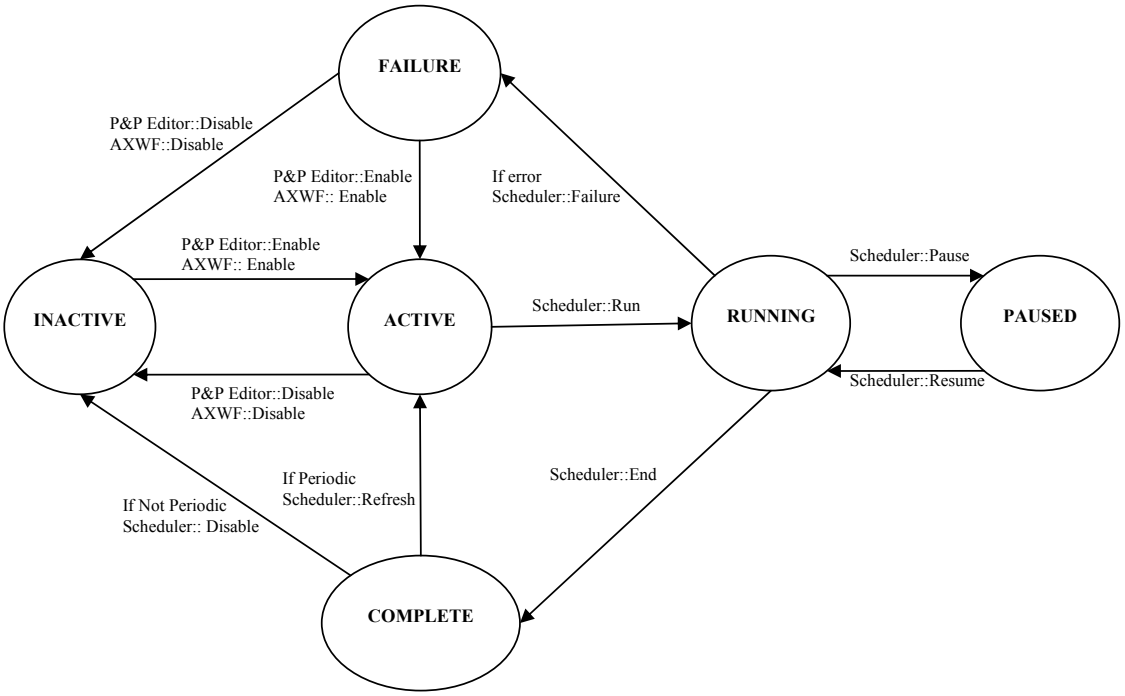
- Commands coming from the AXMDEIS Workflow Manager
  - Run programme (Programme_ID, when)
    - Activate-on-demand is the same as run programme immediately with a specified programme (in XML) which can be given via WebServices by WF
  - Enable programme (rule_type, Rule_ID, XML definitions)
  - Disable programme (Programme_ID)
    - To disable an activated by not currently executing programme
  - Remove programme (Programme_ID)
    - To remove a programme from its activated status and the repository
  - Pause programme (Programme_ID) i.e. suspend the programme
  - Resume programme (Programme_ID)
  - Get programme status (Programme_ID)
  - Kill_programme (Programme_ID) stop execution whichever status
    - To stop a currently executing programme and to remove the programme
  - Get list of programmes
- Reports to AXMEDIS Workflow Manager
  - Error notification (failure messages)
  - Complete programme notification
  - status of the current activity
  - List of scheduled programmes
- Commands to Formatting Engine using WF
  - Install_and_activate (rule_type, Rule_ID, XML definitions)
    - As an example: CallFormatEng(
      IDScaleFunc,
      "<selection>AXID</selection>
      <parameter:size_width>128</>
      <parameter:size_height>128</>
      <format>png</>...", trial_level,
      CallerID);
  - One of the parameters when calling Formatting Engine is the “trial-level” for Actual operation or full-test or quick-test.

3.3.7 P&P Engine communication with the P&P Editor

- Commands coming from the AXMDIS P&P Programme Editor
  - Get list of programmes (Either frome Programme Repository or Active Programme Repository)
  - Enable programme
  - Disable programme
  - Remove programme
  - Activate programme plus programme xml file transfer
- Report
  - Notification of activation
  - Rule Status (specifically results from trial activations)

The communication between the Programme and Publication Editor and the Enfgine will be based on the same protocol used by Workflow i.e. webservices.

3.3.8 Rule State Diagram



The life cycle of rule represents the evolution of a rule from the activation to its completion. The evolution is strictly linked to the activities performed by the scheduler, the dispatcher and the executor and it can be described by means of a status attribute. The status of a rule will assume the following values:

1. **Active** – The programme is waiting for the execution
2. **Inactive** – The programme will be not executed
3. **Failure** – An error occurred during the execution of programme and the execution is blocked
4. **Running** – The programme thread is executing
5. **Paused** – The programme thread is paused
6. **Complete** – The run is finished

The general evolution of the programme status is depicted in the State Diagram representation, whereas the description of each transition is reported in the following.

Complete to Active Transition – The transition from COMPLETE to ACTIVE status is performed if the programme has to be executed periodically. In this case, the programme is re-submitted to the scheduler and its run-conditions are updated on the basis of the specified period.

Complete to Inactive Transition – The transition from COMPLETE to INACTIVE status is performed when the programme has to be executed once. In this case, the programme is ready to be removed from the scheduler or to be modified by means the Programme and Publication Editor.

Active to Running – The transition from ACTIVE to RUNNING status is performed when the scheduler starts a thread to execute the programme.

Running to Failure – The transition from RUNNING to FAILURE status is performed if the following conditions occur during the execution of programme, the thread sends a run-time error message.

Running to Paused **–** The transition from RUNNING to PAUSED status is performed if the when the scheduler receives notification to pause the programme.

Paused to Running **–** The transition from PAUSED to RUNNING status is performed if the when the scheduler receives notification to resume the programme.

Running to Complete **–** The transition from RUNNING to COMPLETE status is performed when the thread has completed the P&P programme.

Failure to Inactive **–** The transition from FAILURE to INACTIVE status is performed when a disable programme request comes from the AXMEDIS Workflow Manager or from the Programme Editor.

Failure to Active **–** The transition from FAILURE to ACTIVE status is performed when a disable programme request comes from the AXMEDIS Workflow Manager or from the Programme Editor.

3.4 Programme and Publication Rule Editor (UNIVLEEDS)

Module Profile		
Programme and Publication Rule Editor		
Executable or Library(Support)	Executable	
Single Thread or Multithread	Single Threaded	
Language of Development	C++	
Responsible Name	Kia Ng and Royce Neagle	
Responsible Partner	UNIVLEEDS	
Status (proposed/approved)	Proposed	
Platforms supported	Windows XP	
Interfaces with other tools:	Name of the communicating tools	Communication model and format (protected or not, etc.)
Repositor of Publication Rules/Selections		
Query Support		
Distribution and Client Profile		
Active Publication Rules		
Active Publication Rules		
File Formats Used	Shared with	File format name or reference to a section
XML plus schema	Repository of Publication Rules/Selections and Active Publication Rules	Programme and Publication Format with extension .pp
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
wxWindows	Visual Studio .NET	STL
	Visual C++	Windows XP
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
		wxWindow: L-GPL

A Programme Manager can use a GUI to interact with the Query Engine and to make selections from the Query Engine results in order to schedule some programmes (e.g. day, week, month, year) with the following rules:

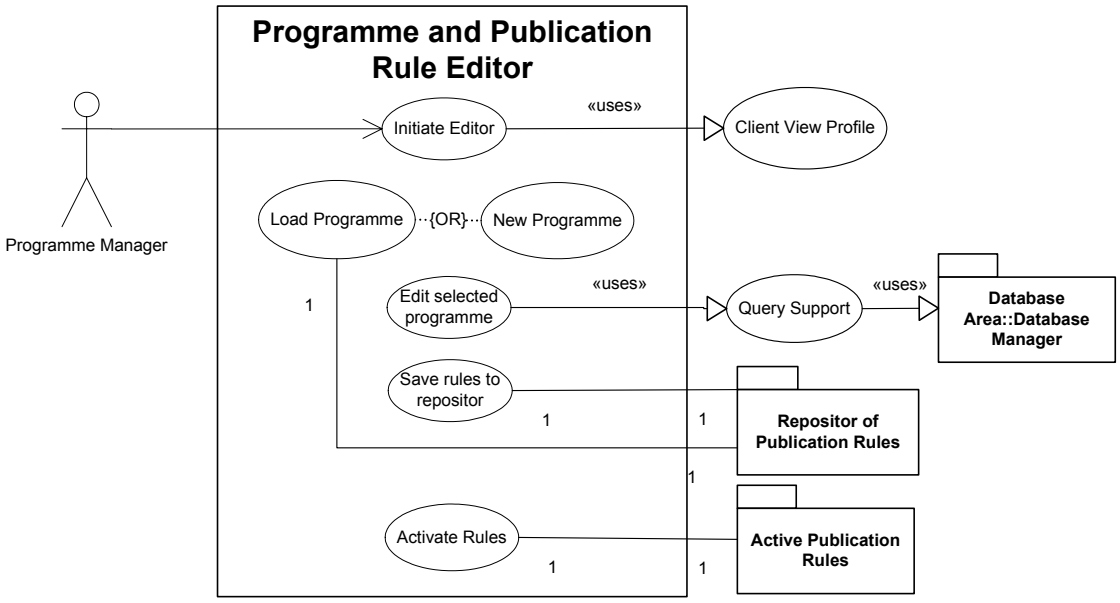
- WHAT: the AXMEDIS object of interest
- WHERE: destination channel, where to publish e.g. iTV or kiosk or other, and "where" profile
- WHEN: date, time, slot, duration
- HOW: direct transfer, reference or require formatting engine

The representation of the above rules could be represented using XML.

The Programme and Publication Rule Editor will provide the following functionality:

- Create: a programme manager uses a GUI to create some programmes using the query engine to browse the AXMEDIS database, to select, to schedule, and to return with a list of relevant objects
- Edit: a programme manager make changes to the programme rules selected from a list in the GUI read from the Programme and Publication Rule Repository. Changes can also include using the query engine to browse the AXMEDIS database to select and to return with a list of relevant objects
- Save: send the Programme and Publication Rules to the Repository for archiving
- Activate: set the Programme and Publication Rules to the Active Publication Rules

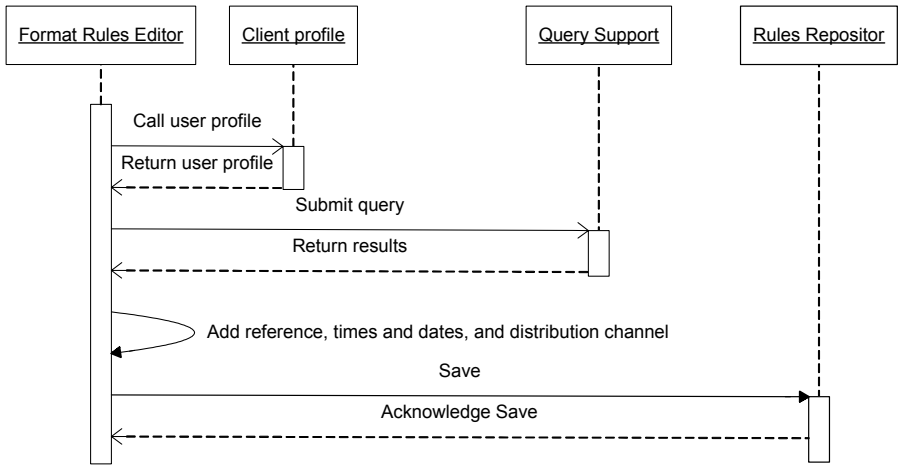
3.4.1 Use Case and Sequence Diagrams for the Programme and Publication Rule Editor



Use Case for creating, editing, saving and activating Programme and Publication rules

Scenario description for new P&P rule:

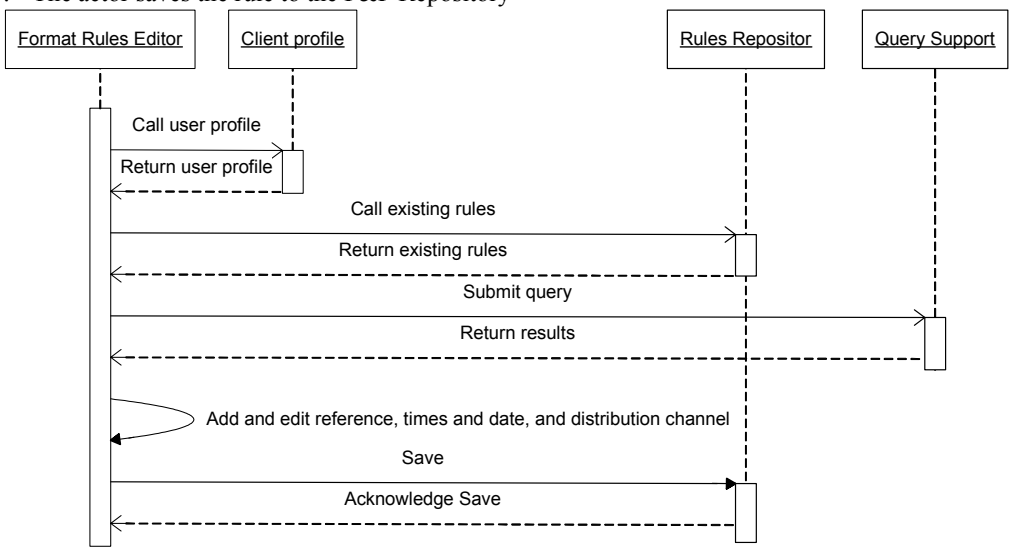
1. The actor opens the GUI (either through WF or directly) and selects a new rule
2. The actor uses query support to add AXObjects /and/or Selections to the rule
3. The actor edits rules
4. The actor saves the rule to the P&P Repository



State diagram for creating a new rule

Scenario description for editing existing P&P rule:

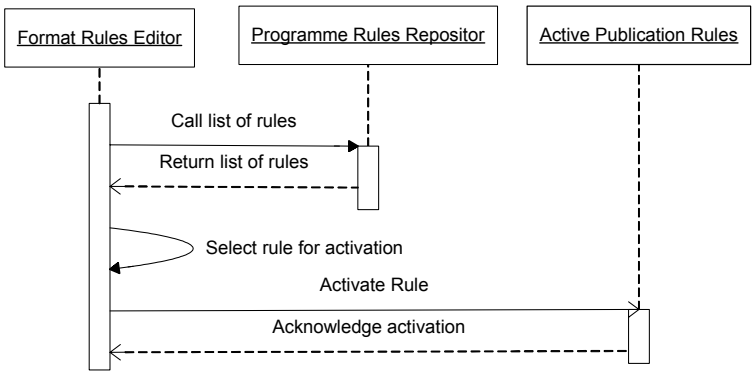
1. The opens the GUI and loads a rule from the P&P Rule Repository
2. The actor uses query support to add AXObjects /and/or Selections to edit the rule
3. The actor saves the rule to the P&P Repository



State Diagram for loading, editing and saving the rule

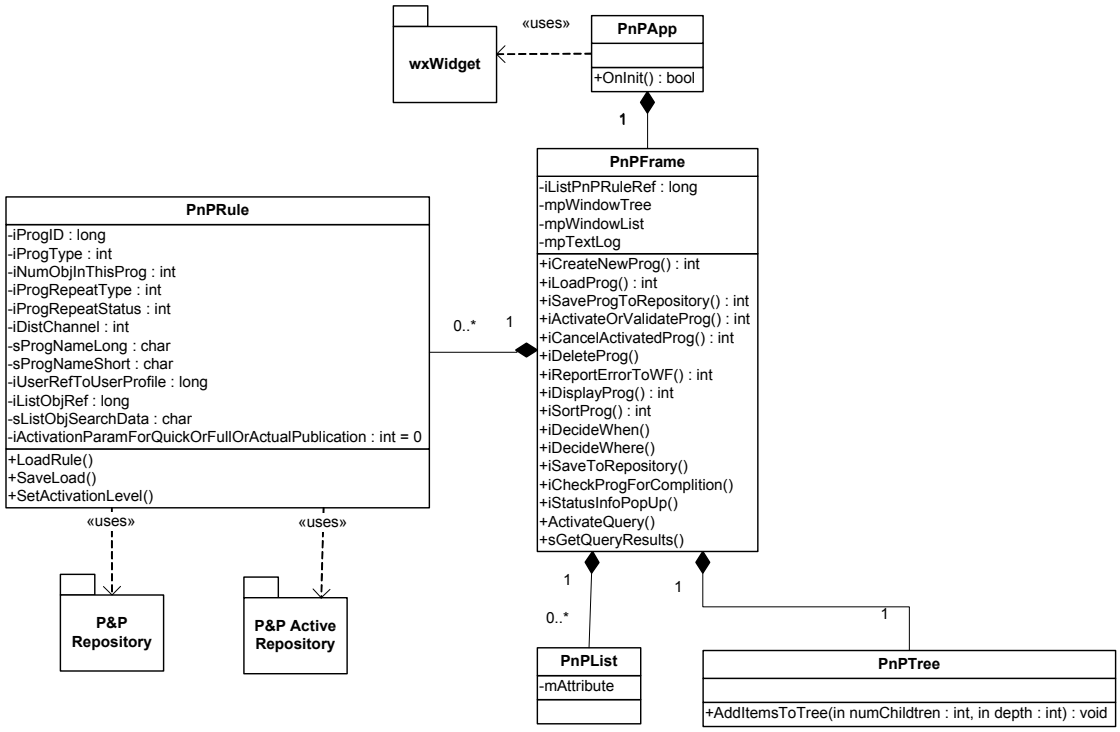
Scenario description for activating an existing P&P rule:

4. The opens the GUI and loads a rule from the P&P Rule Repository
5. The actor activates the P&P rule
6. The rule is sent to the Active P&P Rules Repository



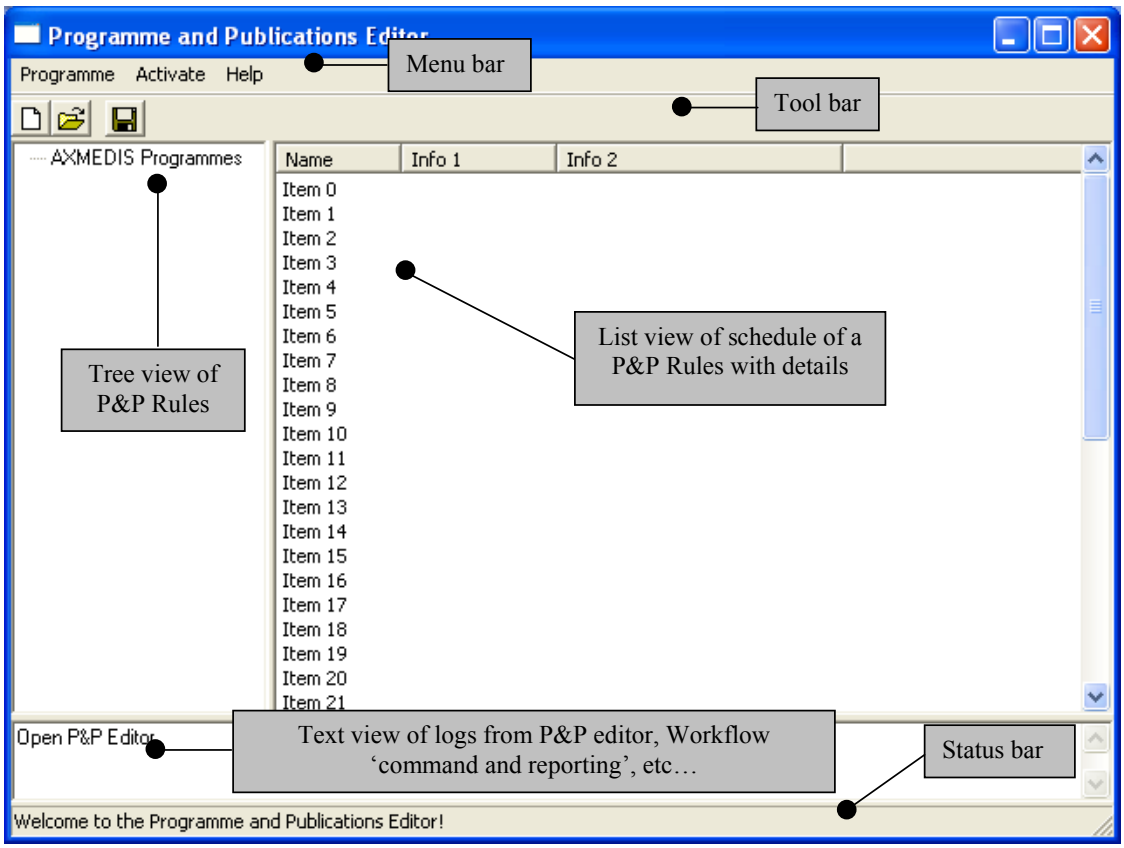
State Diagram for activation of P&P Rule

3.4.2 Programme and Publication Rule Editor Class Diagram



Class UML for the P&P Editor

3.4.3 Programme and Publication Rule Editor GUI



The Menu Bar
The menu bar will be constituted of the following entries:

File

- **New** – create a new P&P rule
- **Load** – load an existing P&P rule
- **Save** – save the current P&P rule using the current file name
- **Save as** – save the current rule by name
- **Delete –** delete the selected P&P rule
- **Close** – Close the current rule document (if we decide to have multiple views of the rule)
- **Exit** – Quit the editor

Commands

- **Activate Rule** – Send the current rule to the Scheduler of the Rule Engine
- **Quick Trial** – Tests the feasibility of the rule (tests on AXMEDIS objects)
- **Full Trial** – As Quick Trial but creating new AXMEDIS objects
- **De-Activate Rule** – Removes the selected rule from the Active P&P repository and sends terminate signal to the associated thread (if running) executing the rule in the P&P Engine.

Messages (This may be automatically set to always show messages)

- **Last message –** Displays the last message sent by the AXMEDIS Workflow Manager into the Log text window

- **Messages List -** Displays the list of messages sent by the AXMEDIS Workflow Manager

Help
- **Help** – Call the on line help
- **About** – Information about the authors, version etc.

Tree view area
In this area the structure of rule is displayed. It will be visualised using a Tree control that will permit to show and browse components according the rule XML schema.

Textual Visualisation Area
This is a text control where log messages, textual description, alert, etc… are displayed.

List view area
In this area the contents of the selected structure of the rule is displayed. It will be visualised using a List control that will permit to show and browse components according the rule XML schema.

3.4.4 User Commands and Reporting
It provides communication support from/to the AXMEDIS Workflow Manager and P&P Engine. Such services are divided in commands for the user and reporting.
- Messages coming from the AXMEDIS Workflow Manager
  - Description of the work to perform
- Messages and commands to the P&P Engine for:
  - activating and removing programmes
  - monitoring the P&P Engine by requesting the list of scheduled programmes

3.5 Repositories of Publication Programmes (UNIVLEEDS)

There are two Programme Repositories: (1) the P&P Rules Repository and (2) The P&P Active Rules Repository. Both repositories will be a file repository in a directory where the programme can be loaded and saved as XML (text) files. The P&P Rules Repository will provide the stored programmes for the Programme and Publication Rule Editor to read and list in its GUI. The file repository will be stored locally. The repository is required:
- to add new programme when the Actor selects ‘save’ in the Programme and Publication Editor GUI
- return a list of rules when the Actor requests in the P&P Editor GUI to edit existing rules, and
- (return / provide a reference / copy) the selected programme to the P&P Editor for editing

as described in the earlier state diagrams above for P&P creation of new rules and editing existing rules.

The P&P Active Rules Repository will be associated with the P&P Engine. When a programme is set to ‘active’ in the P&P Editor or by the Workflow, the programme is stored in the P&P Active Rules Repository that can be accessed by the P&P Engine and the P&P Editor. The repository is required:
- To add a programme when the Actor selects ‘activate’ in the P&P Editor GUI
- return a list of activated rules when the Actor requests in the P&P Editor GUI for checking status
- To delete a programme when the Actor selects ‘deactivate’ in the P&P Editor GUI
- Return the selected programme to the P&P Engine when requested.