



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

[www.AXMEDIS.org](http://www.AXMEDIS.org)

### DE4.4.1 Content Sharing and Production on P2P

**Version:** 1.12

**Date:** 05/10/2005

**Responsible:** CRS4 (revised and approved by DSI)

Project Number: IST-2-511299

Project Title: AXMEDIS

Deliverable Type: Report and Prototype

Visible to User Groups: Yes

Visible to Affiliated: Yes

Visible to Public: Yes

Deliverable Number: DE4.4.1

Contractual Date of Delivery: M13 (end of September 2005)

Actual Date of Delivery: End of September 2005

Work-Package contributing to the Deliverable: (WP4)

Task contributing to the Deliverable: WP4.4

Nature of the Deliverable: Report and Prototype

Author(s): CRS4

**Abstract:**

This document reports on research activities and demonstrator implementation related to WP4.4 Content Sharing and Production on P2P

**Keyword List:**

Peer-to-peer, content sharing, content search, content download, automated activities, query user interfaces.

# Table of Contents

<b>1</b>	<b>EXECUTIVE SUMMARY AND REPORT SCOPE .....</b>	<b>6</b>
<b>2</b>	<b>INTRODUCTION (MANDATORY).....</b>	<b>8</b>
2.1	SPECIFICATION OF T4.4.1 AXEPTOOL, P2P TOOL FOR B2B TRANSACTION AND CONTENT DISTRIBUTION (CRS4) 8	
2.1.1	Research and development plan .....	8
2.1.2	Planned schedule.....	9
2.2	SPECIFICATION OF T4.4.2: THE QUERY SUPPORT INTO THE AXEPTOOL DISTRIBUTED DATABASE (CRS4) .....	9
2.2.1	Research and development plan .....	9
2.2.2	Planned schedule.....	10
2.3	SPECIFICATION OF T4.4.4/5 ANALYSIS OF PUBLICATION AND LOADING MODULES (CRS4) .....	10
2.3.1	Research and development plan .....	10
2.3.2	Planned schedule.....	10
<b>3</b>	<b>REPORT ON WP4.4 ACTIVITIES.....</b>	<b>11</b>
3.1	NOTES ON TASK 4.4.3 WORKFLOW SUPPORT IN THE AXEPTOOL.....	11
3.2	SPECIFICATION OF T4.4.1 AXEPTOOL, P2P TOOL FOR B2B TRANSACTION AND CONTENT DISTRIBUTION (CRS4) 11	
3.2.1	Analysis and specification of support for P2P tools, including query, publication and workflow support (CRS4 ) 11	
3.2.1.1	Introduction.....	11
3.2.1.2	Discussion on Requirements .....	12
3.2.1.3	Architecture .....	12
3.2.2	Results concerning the protocol and architecture suitable for AXMEDIS P2P sharing (CRS4 ) .....	14
3.2.2.1	Analysis of algorithms and architectures for searching/indexing content in P2P networks.....	15
3.2.2.2	Algorithms and architectures for content delivery.....	20
3.2.2.3	First draft of architecture suitable for AXMEDIS P2P sharing.....	23
3.2.2.4	Keeping peers informed about new objects or new versions .....	33
3.3	SPECIFICATION OF T4.4.2: THE QUERY SUPPORT INTO THE AXEPTOOL DISTRIBUTED DATABASE (CRS4) ....	34
3.3.1	Analysis of the specific user interface for technical queries (CRS4) .....	34
3.4	SPECIFICATION OF T4.4.4/5 ANALYSIS OF PUBLICATION AND LOADING MODULES (CRS4) .....	35
3.4.1	Analysis and specification of Publication and Loading Module (CRS4).....	35
3.4.1.1	Introduction.....	35
3.4.1.2	Architecture .....	37
3.4.2	Results on technical solution/architecture for exchange among databases, active selection for loading and publishing. (CRS4).....	40
3.4.2.1	Publication Module .....	40
3.4.2.2	Loading Module .....	41
3.4.2.3	Downloading Module .....	42
<b>4</b>	<b>PUBLICATION MODULE OF THE AXEPTOOL .....</b>	<b>44</b>
4.1	TECHNICAL DETAILS .....	44
4.2	DESCRIPTION OF THE PURPOSE .....	44
4.3	FUNCTIONALITIES .....	44
4.4	SNAPSHOTS .....	44
4.4.1	Receives a Publication Rule actualized by the Rule Engine.....	44
4.4.2	Publish one or more new AXMEDIS Objects or new versions of already published AXMEDIS Objects 45	
4.4.3	Copy the published objects from AXDB to ADBOUT resources folders.....	45
<b>5</b>	<b>LOADING MODULE OF THE AXEPTOOL.....</b>	<b>45</b>
5.1	TECHNICAL DETAILS .....	45

<b>6</b>	<b>AXEPTOOL ACTIVE SELECTION FOR P2P MODULE .....</b>	<b>45</b>
6.1	TECHNICAL DETAILS .....	46
<b>7</b>	<b>AXMEDIS QUERY GUI.....</b>	<b>46</b>
7.1	TECHNICAL DETAILS .....	46
7.2	LISTS OF FUNCTIONALITIES .....	47
7.3	SNAPSHOTS AND EXPLANATION .....	47
7.3.1	Process the form's data in order to compose a query compliant to the AXMEDIS Query model. ....	48
7.3.2	Forward the AXMEDIS Query to the Query Support Web Service .....	49
7.3.3	Process the XML Result and send it back to the browser.....	49
7.3.4	Query Results.....	50
<b>8</b>	<b>AXEPTOOL CORE COMPONENTS.....</b>	<b>51</b>
8.1	AXEPTOOL CORE FEATURES .....	51
8.1.1	Virtual Database Features .....	51
8.1.2	Monitor Features .....	52
8.1.3	Publishing and Monitoring Objects Features .....	52
8.2	AXEPTOOL CONSOLE SCREENSHOTS .....	52
8.2.1	Connecting to the P2P network and tracing connections and other connected peers .....	52
8.2.2	AXMEDIS Content publication .....	52
8.2.3	Querying the network .....	53
8.2.4	Contents Downloading and downloads monitoring .....	53
8.2.5	Monitor the network for updates of downloaded objects. ....	54
<b>9</b>	<b>AXEPTOOL P2P WEBCACHE.....</b>	<b>55</b>
9.1	AXEPTOOL P2P WEBCACHE FEATURES .....	55
9.2	AXEPTOOL P2P WEBCACHE SCREENSHOTS .....	55
<b>10</b>	<b>BIBLIOGRAPHY .....</b>	<b>56</b>
<b>11</b>	<b>GLOSSARY .....</b>	<b>57</b>
<b>12</b>	<b>TECHNICAL APPENDIX .....</b>	<b>58</b>
12.1	THE CORE PROTOCOL AND ITS EXTENSIONS .....	58
12.2	AXEPTOOL CONSOLE.....	65
12.2.1.1	COMPILING THE CONSOLE .....	65
12.2.1.2	RUNNING THE CONSOLE .....	65
12.2.1.3	HELP COMMAND .....	66
12.2.1.4	WHO COMMAND.....	66
12.2.1.5	LISTEN COMMAND.....	66
12.2.1.6	CONNECT COMMAND .....	66
12.2.1.7	CONNECTIONS COMMAND .....	67
12.2.1.8	GET COMMAND.....	67
12.2.1.9	MONITOR COMMAND.....	67
12.2.1.10	PUBLISH COMMAND.....	68
12.2.1.11	QUERY COMMAND.....	68
12.2.1.12	INDEX COMMAND.....	68

### 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](http://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](mailto: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](mailto:nesi@dsi.unifi.it)
- You can attend AXMEDIS meetings that are open to public, for additional information see [WWW.axmedis.org](http://WWW.axmedis.org) or contact P. Nesi at [nesi@dsi.unifi.it](mailto:nesi@dsi.unifi.it)

## 1 Executive Summary and Report Scope

Market and end-users are pressing content industry to reduce prices. This is presently the only solution to setup viable and sustainable business activities with e-content. Production costs have to be drastically reduced while maintaining product quality. Content providers, aggregators and distributors need innovative instruments to increase efficiency. A solution is automating, accelerating and restructuring the production process to make it faster and cheaper. The goals will be reached by: (i) accelerating and reducing costs for content production with artificial intelligence algorithms for content composition, formatting and workflow, (ii) reducing distribution and aggregation costs, increasing accessibility, with a P2P platform at B2B level integrating content management systems and workflows, (iii) providing algorithms and tools for innovative and flexible Digital Rights Management, exploiting MPEG-21 and overcoming its limits, supporting several business and transactions models. AXMEDIS consortium (producers, aggregators, distributors and researcher) will create the AXMEDIS framework with innovative methods and tools to speed up and optimise content production and distribution, for *production-on-demand*. The content model and manipulation will exploit and expand MPEG-4, MPEG-7 and MPEG-21 and others real and de-facto standards. AXMEDIS will realize demonstrators, validated by means of real activities with end-user by leading distributor partners: (i) tools for content production and B2B distribution; (ii) content production and distribution for i-TV-PC, PC, kiosks, mobiles, PDAs. The most relevant result will be to transform the demonstrators into sustainable business models for products and services during the last project year. Additional demonstrators will be 2-3 associated projects launched as take up actions. The project will be supported by activities of training, management, assessment and evaluation, dissemination and demonstration at conference and fairs.

This deliverable is devoted to the description of content protection and supervision done inside WP4.4.

This activity is by no means finished with the completion of this deliverable, but it has to be revised during the development of the project.

Main deliverables in WP4 are:

- DE4.1.1 – Content Modelling and managing (M13), report and prototype;
- DE4.2.1 – Content indexing, monitoring and querying (M13), report and prototype;
- DE4.3.1 – Content Composition and formatting (M13), report and prototype. It also includes the details about the integration of AXMEDIS with workflow management tools;
- DE4.4.1 – Content sharing and production on P2P (M13), report and prototype;
- DE4.5.1 – Content Protection and Supervision (M13), report and prototype;
- DE4.6.1 – Content Distribution via Internet (M13), report and prototype;
- DE4.7.1 – Content Distribution toward mobiles (M13), report and prototype;
- DE4.8.1 – Content Distribution via satellite data broadcast, the push optimisation and the on demand problem (M13), report and prototype;
- DE4.9.1 – The Usability issues for the AXMEDIS production tools (M13), report.

The main activities that have supported the production of this deliverable are related to:

WP4.4 – Content sharing and production on P2P.

In this deliverable we cover three main aspects of research:

- how to share contents in a network of B2B participants and how this content can be searched and indexed.
- How to deliver large objects with an efficient solution.

- How to provide a technical user interface for editing queries.

The B2B environment is a fundamental aspect which drives technical choices. In fact, participants are expected to be known and do not need any kind of anonymity and hosts are supposed to run with a long up-time because they are supposed to be active 7/24. Moreover, one of the aspect is the automation of content distribution over the P2P network. In fact, differently from P2P for consumer-to-consumer, in B2B we need integration with a workflow system and the possibility to make content publication and content loading as part of a pipelined process which involves content production, content searching, and content distribution.

## 2 Introduction (mandatory)

AXMEDIS will pursue an integrated solution to content distribution, providing flexibility and scalability to support any kind of content over any kind of network, and configurable to support terrestrial, satellite and mobile transport protocols.

The activities that have supported the production of this deliverable are related to the analysis of content sharing and distribution on P2P. The main topics presented and analyzed in this document are about:

- Searching contents in a network of business peers
- Downloading content with state-of-the-art P2P protocols
- Automation of publication and loading of contents
- Query User Interfaces.

### 2.1 Specification of T4.4.1 AXEPTool, P2P tool for B2B transaction and content distribution (CRS4)

#### 2.1.1 Research and development plan

The R&D plan aims at the following research objectives:

- A protocol/architecture for automatic discovery of participants
- A protocol/architecture for indexing and query distribution over the net
- A protocol/architecture for content delivery across the net

The above goals will be pursued by testing some existing protocols and eventually developing new ad-hoc protocols. Given the low degree of decentralization required by AXMEDIS some protocols/architectures like Napster/Gnutella2/Edonkey2K will be taken into account for the indexing and query distribution. This analysis will probably lead to the choice of one ad-hoc protocol for AXMEDIS.

Regarding the protocol/architecture for content delivery the architecture of BitTorrent will be taken as reference model and it is likely to be developed an AXMEDIS client for BitTorrent or in alternative a client for AXMEDIS torrent-like protocol.

The work on this area will be decomposed in small iterations cycles of spike solution/test definition/proof-of-concept/simulation where a spike solution consists in exploring a technology running mock-ups and developing on existing libraries, while the simulation is performed with conventional network analyzers or ad-hoc models in order to establish the scalability and the performances.

The second objective of the R&D plan is the integration of P2P architecture in a content factory. This objective requires some sub-objectives to be reached:

- A technical solution/architecture to realize exchange databases (AXIN and AXOUT DB) meeting major requirements of AXMEDIS users
- A technical solution/architecture to realize a system of active selection of objects for publishing in the P2P network
- A technical solution to realize a system of active selection of objects for loading third-party AXMEDIS objects in the factory



- A complete integration of distribution operations on P2P with fingerprinting systems in order to check correctness of exchanged objects
- A complete integration of tools and engines designed with an external workflow system in order to automate publishing and download according to pre-definite activity models.

The components committed to perform have already been identified in the specifications of the AXMEDIS architecture and such components must be designed and implemented for a first prototyping of the AXEPTool area.

### 2.1.2 Planned schedule

The main R&D objectives will be produced in a short release – continuous integration process.

So the work will be divided into several cycles of spike solution/test definition/proof-of-concept/test acceptance with a period of about four weeks. So we will divide the work into several cycles of spike solution/test definition/proof-of-concept/test acceptance with a period of about four weeks. We foresee to release results at:

- M8: Analysis and specification of support for P2P tools, including query, publication and workflow support
- M10: Results concerning the protocol and architecture suitable for AXMEDIS P2P sharing
- M12: Preliminary prototype of P2P architecture at the basis of the AXEPTool
- M13: First version of the query support for the AXEPTool
- M14: First version of the Publication Tool Engine
- M14: First version of the Loading Tool Engine
- M16: First prototype of P2P architecture at the basis of the AXEPTool
- M18: First prototype of AXEPTool including Query Support, Publication and Loading Tool Engine.
- M18: First prototype of integration of Workflow and AXEPTool
- M24: Intermediate version of AXEPTool including Query Support, Publication and Loading Tool Engine, Workflow support, taking into account users feedback
- M36: Final version of AXEPTool including Query Support, Publication and Loading Tool Engine, Workflow support

## 2.2 Specification of T4.4.2: The query support into the AXEPTool distributed database (CRS4)

### 2.2.1 Research and development plan

The research in this task will consist in the experimentation of GUI, prototypes, and in the integration of meta-data in the P2P message based architecture. We expect to work jointly with the Query Support in the AXMEDIS Database (section 4.2 above). The Query Support in the AXEPTool will allow the specification of technical/professional query including metadata, technical information, business and licensing aspects, content based, DRM rules, etc. Studying a specific user interface integrated into the AXEPTool for defining technical queries and it is integrated into WP4.4 Complex technical queries as described in the WP4.2, including query for complex objects and single components, reusable components. Technical queries have to include:

- details related to the description of components (see below for the formatting)
- costs and DRM rules, for each action a price, play, excerpts, redistribution, resizing, distribution on a different area, validity of the DRM rule and copyright coverage, etc.
- available languages if there is speech or text
- range of age suggested
- business model suggested
- time of delivery and availability in terms of first delivering, if not ready
- type of delivery: on-line, offline, etc.

- if on-line time of downloads or acquisition
- cultural level
- subject, description of content with simple metadata
- Textual description of subject and evolution

### 2.2.2 Planned schedule

The R&D objective will be pursued in a short release – continuous integration process.

So we will divide the work into several cycles of spike solution/test definition/proof-of-concept/test acceptance with a period of about four weeks. We foresee to release results at:

- M8: Analysis and specification of query support for P2P tools
- M10: Results on analysis on query support for AXEPTool
- M10: Analysis of the specific user interface for technical queries
- M12: Preliminary version of user interface for technical queries
- M13: First version of the query support for the AXEPTool
- M18: Integration of query support interface into AXEPTool
- M24: Refined version of the query support interface for AXEPTool
- M36: Final version of the query support interface for AXEPTool

## 2.3 Specification of T4.4.4/5 Analysis of Publication and Loading Modules (CRS4)

### 2.3.1 Research and development plan

The research plan consists of two main objectives:

- Analysis and first realization of the Loading Module as depicted in the General Architecture. The Loading Module will be capable of automatically defining rules (selections) for loading content from the AXEPTool framework and make it available for the distributors according to specific activation of Formatting
- Analysis and first implementation of the Publication Module is depicted in the General Architecture. The Publication Tool Module will permit the automatic publishing of AXMEDIS content into the Output database of the AXEPTool according to specific rules and selections. The Publication Tool may automatically invoke the protection tool to protect unprotected content coming from the AXMEDIS database

To reach such objectives it will be necessary to identify:

- A technical solution/architecture to realize exchange databases (AXIN and AXOUT DB) meeting major requirements of AXMEDIS users
- A technical solution/architecture to realize a system of active selection of objects for publishing in the P2P network
- A technical solution to realize a system of active selection of objects for loading third-party AXMEDIS objects in the factory
- A complete integration of distribution operations on P2P with fingerprinting systems in order to check correctness of exchanged objects

### 2.3.2 Planned schedule

- M8: Analysis and specification of Publication and Loading Modules
- M10: Results on technical solution/architecture for exchange among databases, active selection for loading and publishing, integration of fingerprinting system
- M12: Intermediate version of the Publication Module with automatic publishing
- M12: Intermediate version of the Loading Module with automatic rules definition
- M14: First version of the Publication Module

- M14: First version of the Loading Module
- M16: First integration of fingerprinting system
- M16 First integration of Protection Tool with Publication Module
- M18: First prototype of AXEPTool including Publication and Loading Module
- M24: Intermediate version of AXEPTool including Publication and Loading Module, taking into account user's feedback
- M30: First version of AXEPTool including Publication and Loading Module
- M36: Final version of AXEPTool including Query Support, Publication and Loading Module

### 3 Report on WP4.4 Activities

#### 3.1 Notes on Task 4.4.3 Workflow support in the AXEPTool

*The report on the Task 4.4.3 has been removed from this deliverable as requested by the Project coordinator. Information on workflow support are reported in WP4.3 deliverables.*

#### 3.2 Specification of T4.4.1 AXEPTool, P2P tool for B2B transaction and content distribution (CRS4)

##### 3.2.1 Analysis and specification of support for P2P tools, including query, publication and workflow support (CRS4 )

###### 3.2.1.1 Introduction

Peer-to-Peer technologies have drawn a lot of attention in the last years because of the large adoption by end users in file-sharing communities. The P2P can be considered an “old” novelty and not a revolutionary idea and often the distinction between P2P and traditional distributed systems is not so evident. Old Unix tools like “*talk*” or the Web itself were conceived to make Personal Computers directly interconnected and able to exchange data. Although the P2P paradigm was potentially available from the beginning of the Internet era, only with Napster this technology gained the worldwide popularity. As everybody knows, file sharing is mainly used for illegal distribution of copyrighted contents, so the peer-to-peer technology is often referred as the technology of “pirates”. The social and legal implications of the misuse of the technology are beyond the scope of this paper, but it is important to avoid that this misuse casts a bad shadow on a technology which has the potential to leverage the cooperation in community-based processes, to reduce costs in delivery of data and services by exploiting the resources at the edge of the Internet, and to make the net a place with more freedom for the users.

This work mainly covers the aspect of cost reduction in the production and delivery of digital contents in a B2B environments. This reduction can be achieved by a more effective use of computing resources in an overlay network of content producers, distributors, and aggregators.

The resources that can be subject to “effective” use are the CPU, the disk space, and the bandwidth. Most of the time, the CPU of our Personal Computers are “idle”. This occurs when the CPU is not performing a computation in the background and in the foreground the user is involved in an interaction where the computer reaction is one or two order of magnitude faster than the human reaction. Seti@home[1] and distributed.net[2] are success stories of CPU sharing over a network of volunteers.

The disk space is another resource that can be easily shared because the cost-per-giga is dramatically dropped and even a low-end machine is equipped with tens or even hundreds of gigabytes. This fact allows overlay networks like Chord[3] and Freenet[4] to place replicas of files according to specific needs and strategies.

The bandwidth is another critical resources that is often a bottleneck in the traditional client-server distribution. P2P systems like Bittorrent[5] implement a set of algorithms that balance the load of distribution among a network of downloaders.

P2P tools in AXMEDIS (AXEPTool) are used by factories to share their objects in an overlay network. By means of the such tools 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.

### 3.2.1.2 Discussion on Requirements

Most of the popular P2P networks were born for illegal file-sharing purpose so among their features are user-identity hiding, resistance to the censorship, scalability toward huge amount of users, etc. Most of these features are not relevant for a B2B environment like AXMEDIS, where on the contrary the main requirements are the full automation of activities, the integration with a secure environment, the enforcement of digital rights and the availability of a professional query support.

Our P2P system is not a stand-alone application, but it is part of the larger AXMEDIS framework which provides a complete business process definition, defines a digital object schema, including support for DRM and extensible metadata.

In DE2-1-1 we have identified the following general requirements for the AXEPTool for content sharing and distribution in P2P environments:

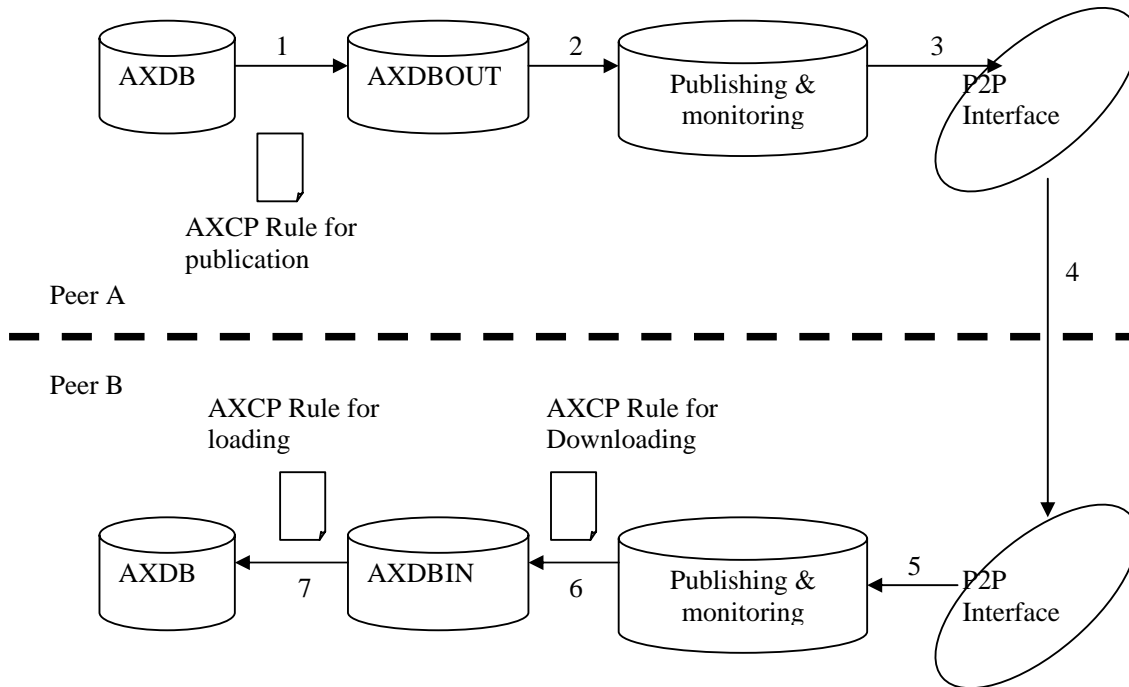
- AXMEDIS Objects should be published and loaded without the need of a centralized infrastructure which would be costly to maintain and a potential bottleneck in the whole production chain. Some functionalities of the network could be centralized wherever the centralisation allows better performance, 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.
- The P2P network should be able to scale to a number of participants as large as  $10^4 - 10^5$
- When a new object is published or an old object is updated into the AXEPTool Out database all the AXMEDIS peers that are the AXEPTool have to be informed.
- When an AXEPTool realize that a new version of an object or that a new object satisfy a P2P Active Selection it has to be downloaded and pushed into the Input Database.

### 3.2.1.3 Architecture

We designed an architecture where 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**



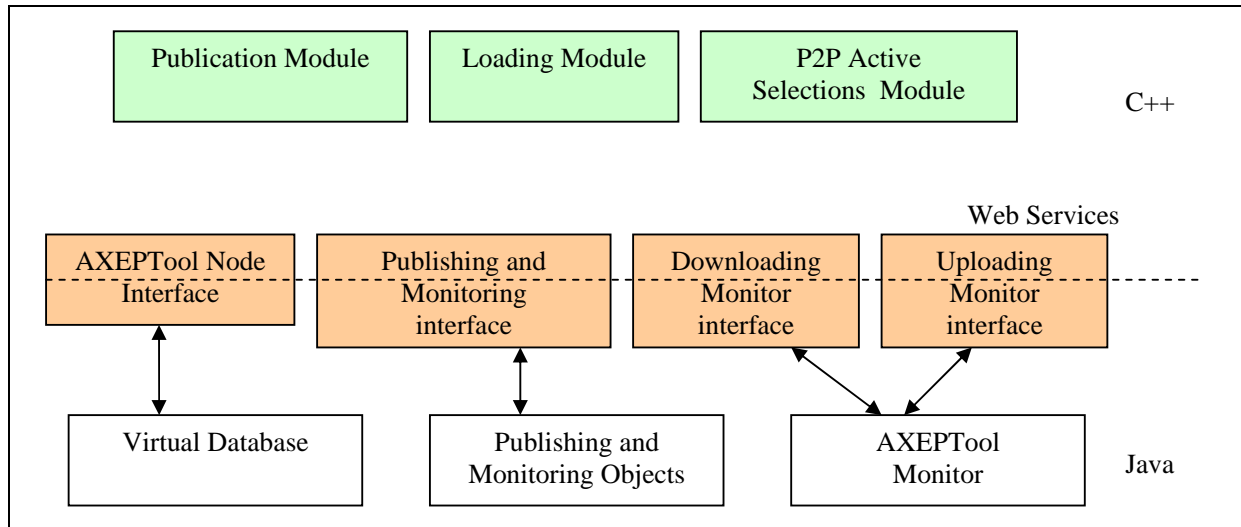
The AXEPTool area includes:

- Publication Module of AXEPTool: it provides functions to publish AXMEDIS objects from the AXMEDIS Data Base, and to move it in the AXOUTDB..
- Loading Module of AXEPTool: it load an AXMEDIS object from the AXINDB to the AXMEDIS Data Base.
- AXEPTool IN/OUT Data Base are instances of the AXMEDIS database.
- AXEPTool P2P Active Selection Module. It activates 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 AXFW/part C. Thus they are written in C++. During the execution of rules, such modules may request functionalities of the Virtual Database, Publishing and Monitoring 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 identified 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



### 3.2.2 Results concerning the protocol and architecture suitable for AXMEDIS P2P sharing (CRS4)

This section provides an overview of existing algorithms and architecture in P2P and then sketches a preliminary architecture and protocol for AXMEDIS P2P sharing.

Most P2P networks were born for illegal file-sharing so among their features are user-identity hiding, resistance to the censorship, scalability toward huge amount of users, etc. Most of these features are not relevant for AXMEDIS, where on the contrary the DRM must be enforced, workflow control during content production and aggregation is added, and less critical scalability issues are faced.

In the last years the research and the development of tools in the P2P field have been biased on the following features:

- Resistance to the censorship. It means that the architectures are able to resist to any action performed by an external authority. This feature is much more justified in those countries controlled by dictatorships due to the unfair restrictions of freedom and politic ideas. Nevertheless, also in democratic countries this feature has been developed in order to allow people to share and deliver multi-media contents infringing rules and laws regulating the copyright.
- Scalability. The possibility to have millions of hosts connected at the same time, able to share terabytes without collapsing the network
- Security. Networks are resistant to malicious nodes which aim at hinder network activities.

Taking into account this needs P2P architectures have been evolved during years reaching the following features:

- Decentralization. The network has not any central access point which could be closed by an authority. As example of centralized network we can mention Napster, in which every search action has to be done through a central point of access. In a different way, Gnutella and Freenet have decentralized every connection, indexing and delivering operation, making possible the resistance of the networks to the mere shutdown of a server.
- Anonymity. In order to guarantee the resistance to the censorship is also necessary to guarantee the anonymity of users (who share and who look for material). Although no authoritarian action is

possible for a decentralized network, single users can be accused of "copyright infringement". In USA hundreds of users have been subject to subpoena by RIAA (Recording Industry Association of America) and they have agreed to reach a settlement of some thousands US dollars.

- Reputation. Decentralized and anonymous networks can be powerful tools for users and even for copyright owners. Legal uses of P2P networks allow the sharing of original material and the delivering of not censurable news. As anonymous networks don't verify user identity, they are vulnerable to the delivering of bogus material which causes unnecessary traffic and deep distrust in the network. As example can be imagined the dissemination of wrong news. Thus, research is active to provide reputation for virtual identities.

The role of P2P networks in AXMEDIS project is very important. The aim is to take advantage of technologies for multimedia resources sharing in the process of creation and aggregation of multimedia content. In this perspective AXMEDIS requirements are different than requirements of traditional file-sharing networks, so solutions will be different. Anyway any adaptation of available architectures and protocols will be investigated. A first analysis shows that none of the existing architectures meets the whole production-to-delivery chain. It seems reasonable to join together several solutions to obtain a single P2P module.

A summary follows of the main characteristics of P2P architecture inside the AXMEDIS framework (the exact elicitation of requirements is beyond the scope of this document):

- Nature of contents. AXMEDIS contents will be compounds and more complex than simple multimedia resources. The dimension will be more or less equivalent to that of video resources in a composite content. For this reason downloading and streaming strategies have to be assessed. Among the others, Shoutcast solutions for distributed streaming in cluster and BitTorrent architecture will be investigated
- Workflow. A Workflow system will be integrated. Further information will be provided through the WP4 activity and reports.
- DRM. Although many P2P frameworks enforce decentralization and anonymity of access, in AXMEDIS we are interested in intellectual property protection and copyright. For this reason is necessary the identification for both publishers and beneficiary of content, in production phase, aggregation and final delivery. Network design has to take into account identification, digital signature, digital certificate, and connection to certification authority. DRM tools and solution in AXMEDIS will be studied in WP4.5 and integrated in the P2P architecture.
- Scalability. In the case of production, aggregation and sharing of content, scalability seems to be a problem less critic than in global file-sharing network such as Kazaa or eDonkey2k. It is reasonable that the number of actors will not catch  $10^6$ - $10^7$  but rather  $10^2$ - $10^4$ . Moreover, it seems not necessary a design of the network with a completely decentralized structure, because every activity has to be considered legal and authorized therefore not censurable. Relaxing the constraint of a completely decentralized architecture is possible to design a network able to better manage the traffic.

### **3.2.2.1 Analysis of algorithms and architectures for searching/indexing content in P2P networks**

#### **A la Napster**

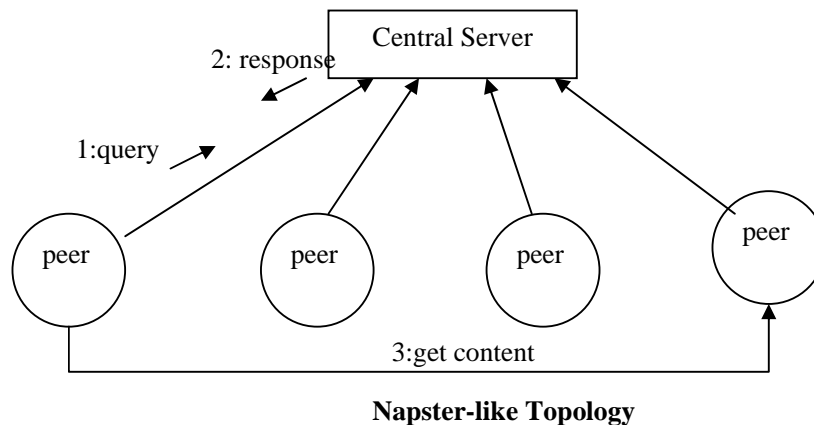
The first and probably the simpler approach to search content in a network is the centralized approach “**a la Napster**” [6]. Napster pioneered the concept of peer-to-peer file sharing. People stored files that they wanted to share (typically MP3 music files) on their hard disks and shared them directly with other people. The architecture for indexing and searching content is very simple: every peer connects with a central server (a singleton of the whole network) and send the list of all files he wants to share. Once a peer wants to search a content he sends to the central server one or more keywords which are compared against the file names

stored. Then, a list of possible matches is sent to the peer, every match contains the IP address of the peer that physically stores the file. The download of the content is performed by a direct connection without involving the central server.

The creator of Napster had two of reasons for this approach:

- There is no way a central server could have enough disk space to hold all the songs, or enough bandwidth to handle all the download requests.
- Napster was trying to take advantage of a loophole in copyright law that allows friends to share music with friends.

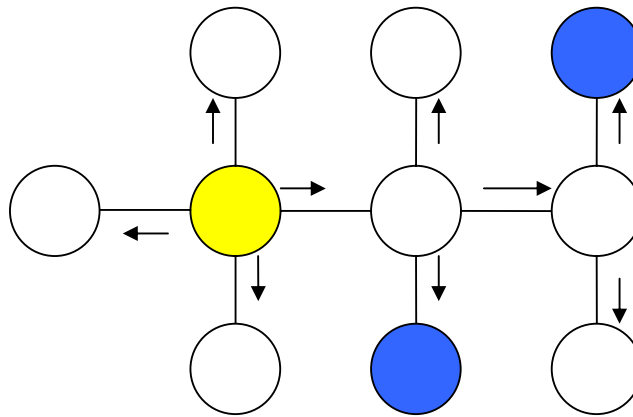
Of course, the music industry brought the case to courts and the Napster service was closed. The fast growing and the fast sinking of the Napster community were the two effects of its architecture: the centralized approach ensures a well working search/indexing mechanism but on the other hand it is a critical weakness. If the central server is shut down the whole network is down. The Napster network gained an immense popularity because for the first time people realized that by means of the Internet technology a valuable product like music can be managed like any other type of information and made free beyond the legal and technological limits of the traditional business model based on CD purchase.



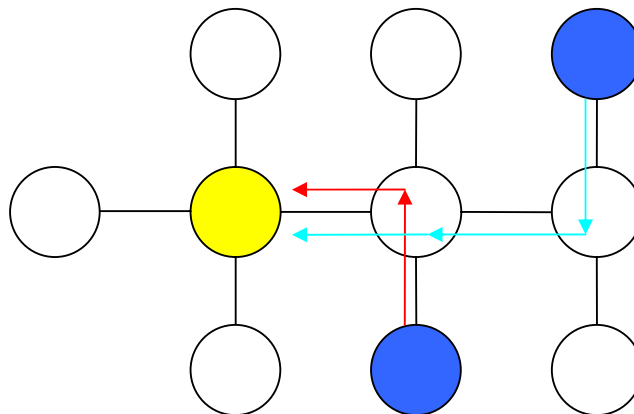
### Flooding Search

Once Napster proved the capabilities of the Internet as a mean for sharing contents, it was just a matter of time that a new architecture were deployed to fill the gap left by the Napster network. If the centralized architecture of Napster was his “Achille’s heel”, then the new architectures were born with a fully decentralized model. The first serious work in the field is the Gnutella [7] network. In a Gnutella network (if we refer to the old versions of the protocol) all peers are first-class citizen and there is not any central server. All peers can act as clients and as servers when required. The topology is highly dynamic, and every peer is connected with TCP sockets to one or more other peers. As usual, peers can produce new queries and queries are forwarded to all neighbors until all the network is covered. Any individual peer can send a response to the queries it receives, and such response (also called query-hits) are back propagated to the original requestor. Query messages in Gnutella have a special field called TTL (Time to live) which is decreased by one at every node. This limits the number of peers involved in the query. For instance, if every peer forwards to  $N$  other peers, the TTL is equal to 10, and the topology is a tree (it is not a valid assumption because generally the topology is a graph) then the number of peers reached by the message is equal to  $N^{10} + N^9 + N^8 + \dots + N^0$ .





**Flood-based search:**the yellow node produces a query which is propagated to neighbors with a flooding algorithm. One or more nodes (blue nodes) decide to send a response.



**The two distinct response messages are back-routed to the original requestor (yellow) via the red-route and cyan-route.**

The flooding algorithm described above have some disadvantages:

- Queries for files can take some time to get a complete response.
- Every machine is part of this network. It is responding to queries and passing them along, and it may be involved in the process of routing back responses as well. This make use of some amount of your bandwidth to handle requests from all the other users.
- Poor scalability, narrow band peers tend to be choked from the network for the huge amount of message passing.

### **Random walk**

To reduce the huge amount of traffic caused by flooding the flowing strategy can be deployed:

- A node sends a query to a random selection of its neighbors, the query has a TTL > 0
- A neighbor receiving the query must: reply to the query if he can, otherwise he must forward the query to a random selection of his neighbors reducing the TTL by one and so on.

### **Hub based**

A viable trade-off between good performances and decentralization is in the adoption of semi-centralized architectures based on two different types of nodes: hubs and leaves.

For instance the “Gnutella2 network is an ad-hoc, self-organising collection of interconnected nodes cooperating to enable productive distributed activities.” [8] .

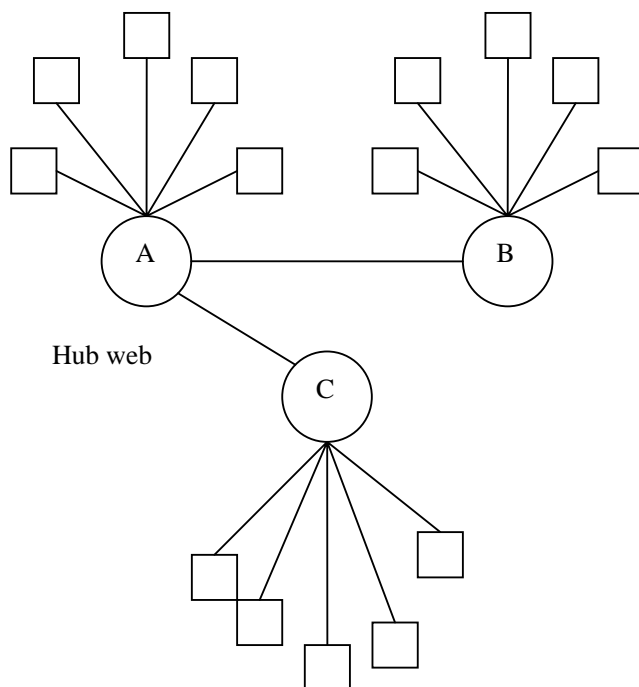
The objective of a hub-based network is to maximise the number of leaves and minimise the number of hubs and the ratio of leaves to hubs is known as the "leaf density". The network is then composed by the “hub web” which is a fully decentralized network of hubs and by the “edge” which is the set of leaves connected to hubs.

Leaves have not particular requirements, on the other hand hubs must:

- be able to support > 100 sockets
- have high CPU and RAM available
- have long uptime (many hours, at least two)
- have adequate bandwidth, primarily inbound bandwidth
- be able to accept inbound TCP and/or UDP

Hub responsibilities include:

- Maintaining routing tables
- Solving queries and collect index table of connected leaves in order to reduce the traffic hub-leaves
- Monitoring the status of local connections and deciding whether to downgrade to leaf mode.



A typical hub-based network is composed by an hub web which is fully decentralized and where every hub is connected to 1-20 other hubs. Moreover, to every hub there are also connected a large number of leaves (100 or more).

### **Bloom filters**

An efficient network topology should reduce the amount of traffic between nodes. The hub-based topology is a step forward but even hub-based networks like Gnutella2 use filters to restrict the propagation of queries among hubs. In other words, once a node with  $N$  neighbors receives a query, it must select  $k < N$  neighbors to forward the query. Differently from ‘random walk’ strategies, network like Gnutella2 and some implementation of Gnutella-1 [9] adopt the ‘Bloom filters’.

The **Bloom filter**, [10], is a space-efficient data structure that is used to test whether or not an element is a member of a set. False positives are possible, but false negatives are not. Elements can be added to the set, and then the filter is updated with the new element “footprint”. The probability of false positives increases with the number  $N$  of elements in the set.

For example, one might use a Bloom filter to decide whether or not propagating a query to a neighbour. In fact, inserting a bloom filter in the connection, if a node receives response for a query  $Q$  along a connection  $C$ , then the query  $Q$  is inserted in the filter and then in future propagation we can assume that the query can be repropagated along  $C$ . There is the probability of a false positive, but this is a good way to reduce the traffic.

An **empty Bloom filter** is a bit array of  $m$  bits, all set to 0. There must also be  $k$  different hash functions defined, each of which maps a key value to one of the  $m$  array positions.

To **add** an element  $x$ , compute the  $k$  hash functions for  $x$  to obtain  $k$  array positions. Set the bits at all these positions to 1.

To **test** whether an element  $y$  is in the set, compute the  $k$  hash functions for  $y$  to obtain  $k$  array positions. If any of the bits at these positions are 0, the element  $y$  is not in the set. If all are 1, then either the element is in the set, or the bits have been set to 1 during the insertion of other elements.

On the other hand, **removing** an element from a simple Bloom filter is not possible. Each element maps to  $k$  bits, and even if setting any one of these  $k$  bits to zero is a way to remove it, this has the negative effect of removing all the elements that map on that bit. This would lead to a possibility of false negatives, which are not allowed.

### **Distributed Hashtables**

One of the most useful data structures devised in computer science is the ‘hashtable’. A distributed hashtable is a system which maps a key onto a piece of data and onto a node of the network. This way, any node is responsible to store only a part of the entire hashtable: the part mapped in a given region of the ‘key’ space. The concept of distributed hashtable has been implemented with different protocols like Chord[3], Pastry[11] and Kademlia[12]. The most remarkable feature of the DHT is that any search keyword reaches the right node (the node that can effectively send a response or simply the node recipient of a message) with only  $O(\log(N))$  messages. This is a very little amount of messages if compared with other networks like Gnutella.

One of the problems with the DHT architecture is managing complex queries. In fact, DHT works well for the exact match of a keyword but it seems not the best solution to address complex queries like:

***Select all where author contains ‘Bach’ AND publication year is between 1990 and 2004***

There are some works[13] in the field that try to address this issue but at the moment there are no widely used applications that prove their efficacy.

One simple but inefficient way of doing complex searches with DHT is to search first for the exact match (author equal "foo") and then filter the results. This assumes that the query always has the format "(key==value) AND (<arbitrary boolean expression>)". An example if this strategy is exploited in Overnet, the Kademia[12] derivative used by eDonkey 1.0.

The filtering is performed "at the server side", in each peer being queried, rather than by the client that queried the DHT and may prevent a lot of unnecessary traffic.

### **3.2.2.2 Algorithms and architectures for content delivery**

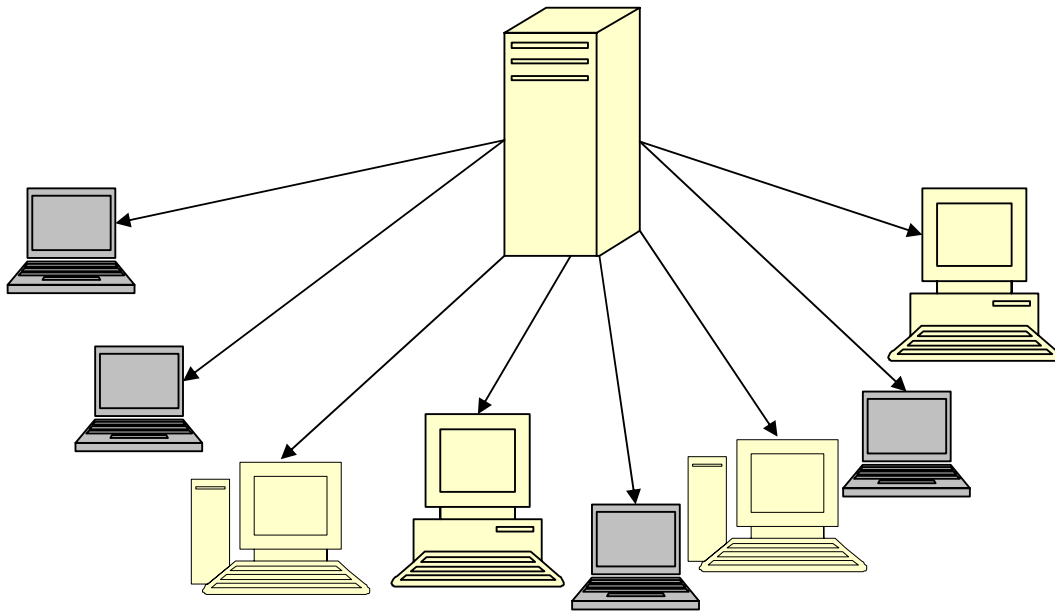
After an object has been located the second problem for a client is how to get the object itself. This problem could appear trivial if we do not take into account the size of contents and the necessary amount of resources in terms of bandwidth for uploading contents to hundreds of concurrent clients. In fact, the size of a content can range from one or more MB for a single song to some GB for a DVD file system image.

#### **HTTP/FTP Direct connections**

The simplest way to deliver content to clients is to accept incoming connections and provide the data through a standard protocol like HTTP or FTP. This solution is not only the simplest but has other advantages: it is easy to find very reliable implementations of HTTP/FTP servers, the clients can use already installed tools like the Explorer in Windows or any Web browser to get the content.

The main disadvantage of the direct HTTP/FTP protocol is in the big workload delegated to the serving machine. In fact, a single HTTP server can serve multiple clients easily, but if the size of the resource requested is large and the resource itself is a very popular file the whole bandwidth is filled almost immediately with two consequences:

- The server can accept many connections limiting the transfer rate of the single connection and thus making downloads last for a long time
- The server limits the number of connections, ensuring higher speed but serving only a subset of the clients which are enqueued for a long time.



**Direct connection HTTP,FTP**

Another disadvantage of the HTTP direct connection based on standard clients is in the absence of resumable downloads. This limitation is not in the protocol itself, in fact the HTTP protocols define the header “range” which allows a client to get only the portion of file inside a range. This header is fully exploited in clients like Wget [14] which allows to resume a download interrupted either by the user or by a system failure.

### **Download from multiple sources**

A way to manage the problems with the single-server HTTP content delivery is to mirror files in other locations.

One of the possible strategies is the automatic selection of the best mirror to download according to congestion, distance, bandwidth and all factors that can improve the download.

Another approach is to provide clients with the capability to simultaneously load pieces of data from more than one source.

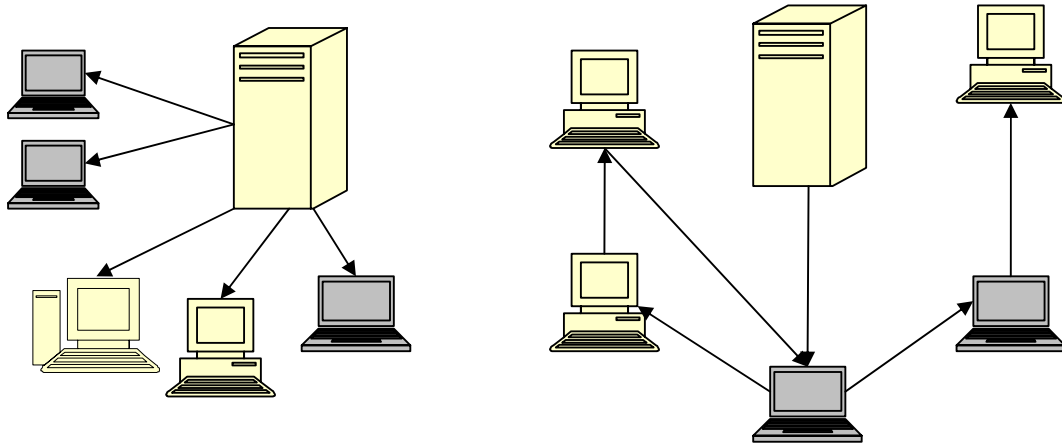
Splitting files into pieces and letting clients to download and immediately share these pieces has proved to be an effective solution already exploited in systems like e-Mule and Bittorrent. These approaches are special cases of “swarming downloads” [15]. In eMule [16] the client selects the download order of parts in order to maximize the overall network throughput and sharing. Each file is divided to 9.28 megabyte parts and each part is divided to 180 KB blocks. To optimize downloads and content sharing very rare chunks must be downloaded as quickly as possible to become a new available source.

This algorithm usually selects first the rarest parts. However, partially complete parts that are close to completion may also be selected. For common parts, the downloads are spread between the different sources.

### **Bittorrent and related algorithms.**

BitTorrent is probably the today most efficient download system based on a P2P network. The idea behind is quite simple: if you have a large popular file to deliver, instead of enqueue hundreds of clients to get that file, publish a metainfo file called “torrent” file. Inside the torrent file there is, among the other information, the Internet address of a special service called “the tracker” which is a sort of registrar. Any new downloader loads the .torrent file from a simple Web server, connects to the tracker and registers itself obtaining a list of other downloaders. Any downloader connects to one or more other downloaders, gets the map of pieces

owned by the others, and tries to get the pieces it needs. A downloader having all the pieces can shut down and disconnect or decide to stay online just to allow other peers to get more pieces. In this last case the downloader becomes a ‘seed’. To boot the network the original content provider must start a downloader in ‘seed’ state.



In a traditional single-server delivery, the server must limit the number of connections according to the bandwidth available. On the other hand, in a BitTorrent network, any downloader provides one or more upload streams contributing to the total balance  $TotalUploadRate = TotalDownloadRate$ . In other words, as the BitTorrent shares the upload rate among several peers there is no limit to the scalability of the system in terms of number of downloaders.

The system works efficiently thanks to a number of algorithms related to the piece selection. Any piece of a file is divided into subpieces and once a sub-piece has been downloaded all the remaining sub-pieces must be downloaded before to make requests for other pieces. Another important technique is called ‘**rarest first**’. In other words, the downloader must give priority to pieces that are ‘rare’ in the network in order to provide a brand-new copy available for other peers. It also make that pieces more common are left for later. An exception to rarest first is “**random first**” when a peer is just started it is important to download at least a piece. Trying to apply “rarest first” from the beginning could lead to very slow downloads, so the first piece is randomly chosen and once the piece is loaded then the downloader switches to rarest first mode.

Sometimes, a downloader may experience a slow transfer rate for a given piece if the uploader peer has a narrow bandwidth connection. This is not a big issue in the middle of a download but it could be a problem when downloading the last one or two pieces. For this reason, a downloader can enter in ‘**endgame mode**’ and broadcasts to all peers requests for all subpieces it needs. To prevent waste of bandwidth and to limit the probability to receive pieces more than once, the peer must send a ‘**cancel**’ message for every piece it receives in this phase.

### 3.2.2.3 First draft of architecture suitable for AXMEDIS P2P sharing

This section describes the issues and the solutions for indexing and searching contents in the network; for securing access to the network; for delivering of large objects and finally for keeping peers informed about news related to an object.

#### Searching contents

**Problem:** The AXEPTool must provide work with complex technical queries. Technical queries have to include:

- i. details related to the description of components.
- ii. costs and DRM rules, for each action a price, play, excerpts, redistribution, resizing, distribution on a different area, validity of the DRM rule and copyright coverage, etc.
- iii. available languages if there is speaker speech or text.
- iv. range of age suggested.
- v. business model suggested.
- vi. time of delivering delivery and availability in terms of first delivering, if not ready.
- vii. type of delivery: on-line, offline, etc....
- viii. if on-line time of downloads or acquisition.
- ix. cultural level.
- x. subject, description of content with simple metadata.
- xi. Textual description of subject and evolution.

It is evident that the use of a P2P network implies that queries produced by the user should be processed by participants in the P2P network. A query-hit is a datum containing all the information related to a remote AXMEDIS Object (a remote AXMEDIS Object is an AXMEDIS Object stored in another organization and shared in the P2P AXMEDIS Community). Among the others, the query-hit should provide :

- i. size of the content
- ii. version where applicable
- iii. identification of the owner
- iv. some form of address to get the remote object.

Technical queries are expected to be composed by query expression connected with logical operators and arbitrary levels of nesting. This requirement suggests that the query support in the AXEPTool must have the same functionalities of the general query support. For this reason, to avoid duplication of work we design the query/indexing system in the p2p network as a “remote-extension” of the local query support.

#### **Possible solutions:**

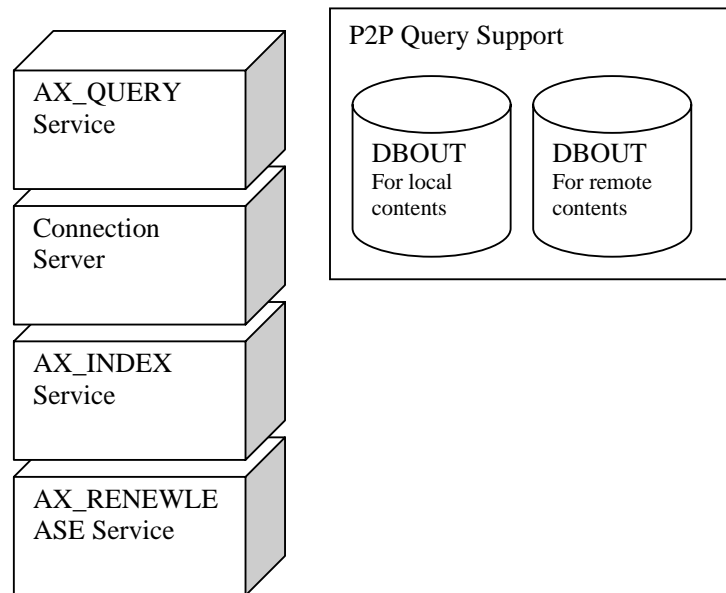
1. A DHT can be used to index and search contents. This solution has the advantage to be very scalable but has the disadvantage of being scarcely applicable to complex queries.
2. A completely decentralized network a la Gnutella where each node implements a local query support able to handle complex/technical query will solve the problem and it is very resistant to faults. Nevertheless, it has the disadvantage of being difficult to monitor and difficult to make it scale to thousands of participants.
3. A semi-centralized network where hubs store indexes and solve queries while leaves simply send queries will partially solve the problem but we need to implement a mechanism in order to object stored in a leaf L to be indexed in a hub H where H has a full AXMEDIS query support capabilities. This solution is more scalable than (2) and it is viable if we assume hubs to have broad bandwidth, large computational power, reliable presence in the network.

We decided to adopt solution (3) with hubs and leaves. Some details follow below and will be included in the next specification phase of AXMEDIS:

both types of nodes are business nodes compliant with the AXMEDIS architecture and will have:

- A database called **AXDBOUT** to index published objects
- A process called the “**AXEPTool Virtual Database (VDB)**” which is accessible both via TCP/IP from outside and can be commanded via SOAP/WebServices[part F] and via terminal console from inside the factory. The VDB implements a wire protocol for exchange data with other remote nodes. The protocol is described in Appendix (“**The core protocol and its extensions**”) and will be included in the next specification phase.
- Inside the VDB there are other multithreaded services like the Download Monitor and the Publishing and Monitoring Object (PandM) service.

In the picture below are showed the main components of the AXEPTool VDB which are ONLY devoted to the indexing and querying in a HUB NODE. The other components are not part of this view and are explained in the other sections.



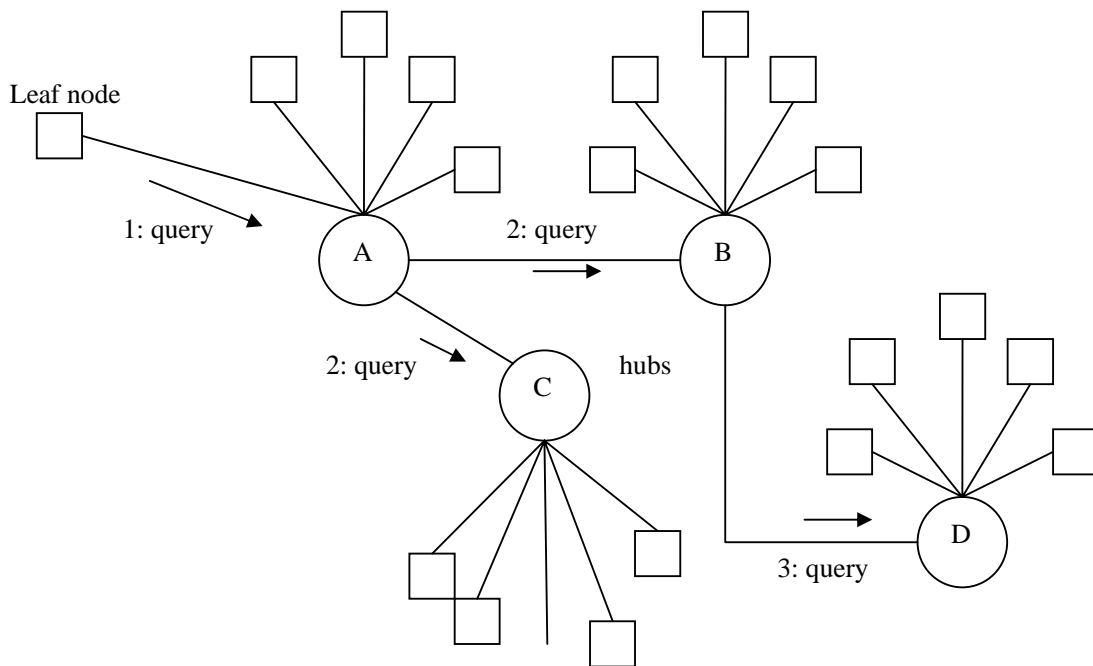
**Hub node architecture for indexing/searching contents**

The Hub node is composed by:

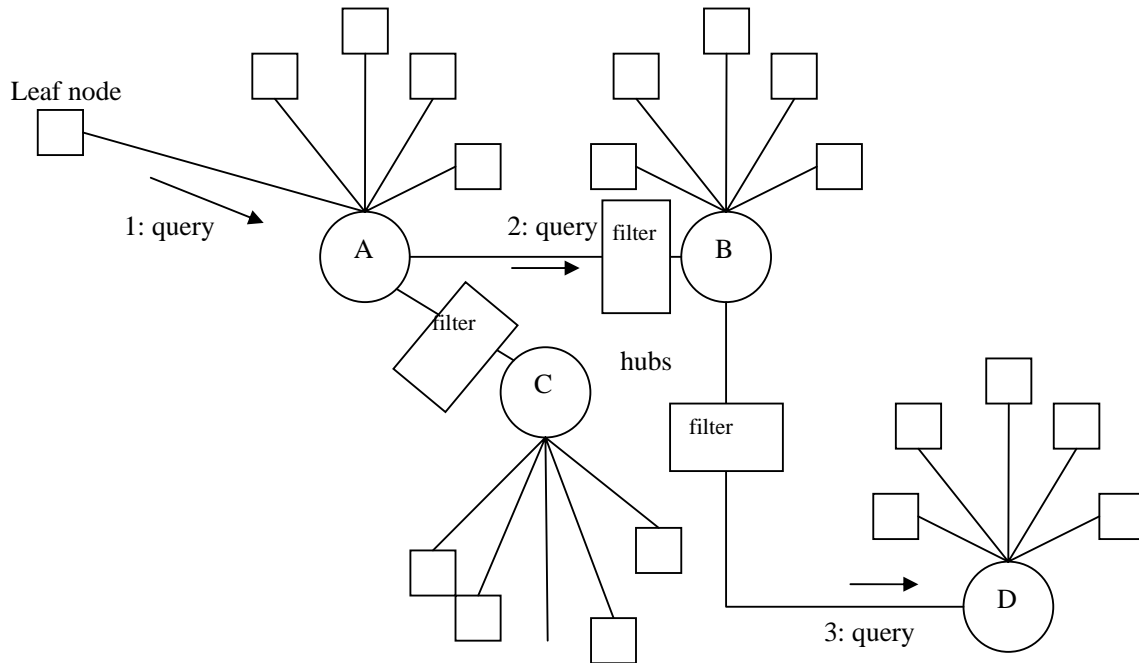
- Connection Server which allows other peers either hubs or leaves to connect with a TCP/IP connection.
- AX\_QUERY is a service in the node which collect a query from a remote peer, connects to the local P2P query support and produce zero or more results. The results are sent back to the connection and are back propagated. In all cases the query is also forwarded to other hubs with a flooding mechanism **limited** to hubs. In some cases, where the query is in the form: *field equal value AND (other complex query)* a bloom filter may be used to avoid to send the query to hubs that are probably unable to give a result. The AX\_QUERY service in a hub allows to :
  - produce a new query;



- produce a response to a remote query;
- collect results from a response received from a remote hub.
- AX\_INDEX is a service in the node which collects metadata of a objects stored in one of its leaves. It is up to leaves to send AX\_INDEX requests to the hub. The hub stores these metadata in the AXDBOUT and these metadata are used to produce query results to AX\_QUERY requests. The allocation of space for remote metadata is governed by a lease: when the lease expires the metadata of remote objects are removed. If the remote leaf renew the lease, the metadata are stored until the next lease expire time. The AX\_INDEX service in a hub node allows:
  - Receive and process a AX\_INDEX request from a leaf node
- AX\_RENEWLEASE is a service which manage the allocation of resources in the AXDBOUT. The AX\_RENEWLEASE service in a hub node performs:
  - Receive a renew lease request and assign a new expire time to the resources
  - Wipe out from the AXDBOUT all remote resources which have expired the lease time

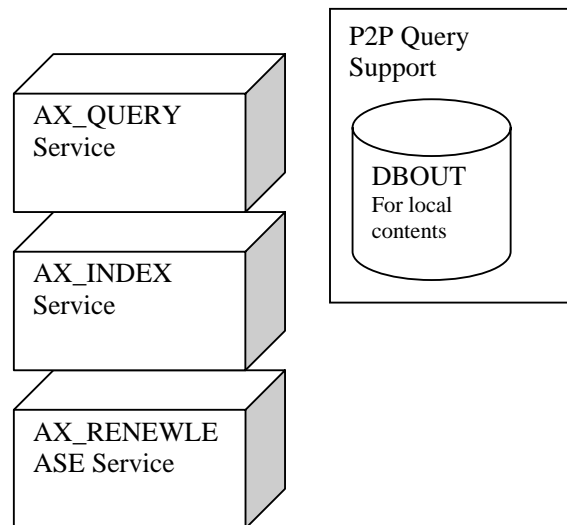


**In the picture, A,B, C and D are hubs. The query is produced by a leaf of A and forwarded to all the hubs. Queries are never forwarded to leaves, because leaves' indexes are already stored in hubs.**



**If the query can be filtered, it is only propagated to connections that transport the query to hubs that are likely to produce at least one result. In the simple example, we assume that A-B connection has a positive test on the query, while the A-C connection has a negative test. This reduces the number of messages.**

In the picture below are showed the main components of the AXEPTool VDB which are ONLY devoted to the indexing and querying in a LEAF NODE. The other components are not part of this view and are explained in the other sections.



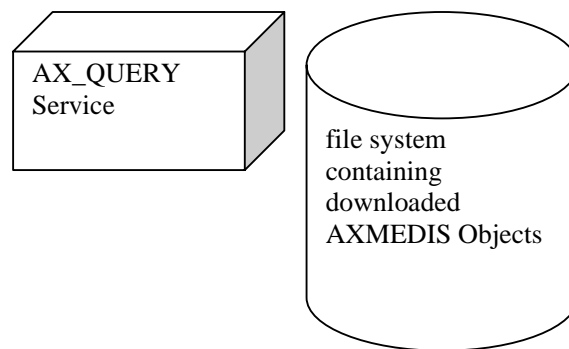
**Leaf node architecture for indexing/searching contents**

The architecture of the LEAF node is simplified because the leaf CANNOT receive incoming connections in the search/index topology (this does not mean that they cannot receive incoming connection in general. In fact, in the delivery architecture leaves can accept incoming connections for uploading contents)

The Leaf node is composed by:

- AX\_QUERY is a service in the node which PRODUCES a query in the AXMEDIS format and sends it to the hub. The results are expected from the connection and are managed locally. The AX\_QUERY service in a leaf node allows to :
  - produce a new query;
  - collect results from a response received from a remote hub.
- AX\_INDEX is a service in the node which collects metadata of a objects stored in local DBOUT and build a message to be sent to the hub. It is up to leaves to send AX\_INDEX requests to the hub. The hub stores these metadata in remote AXDBOUT and these metadata are used to produce query results to AX\_QUERY requests. The allocation of space for remote metadata is governed by a lease: when the lease expires the metadata of remote objects are removed. If the remote leaf renew the lease, the metadata are stored until the next lease expire time. The AX\_INDEX service in a leaf node allows:
  - produce and send an AX\_INDEX request to a hub node
- AX\_RENEWLEASE is a service which manages the allocation of resources in the remote AXDBOUT. The AX\_RENEWLEASE service in a leaf node performs:
  - produce and send a renew lease to the hub.
  - Check which objects in the local DBOUT need to renew the lease in the remote hub

In the picture below are showed the main components of the AXEPTool VDB which are ONLY devoted to the indexing and querying in a LEAF NODE in the CONSUMER side. The other components are not part of this view and are explained in the other sections.



#### **Leaf node architecture for indexing/searching contents in a B2C application**

The architecture of the LEAF node in the consumer side is simplified because the consumer:

- CANNOT receive incoming connections in the search/index topology (this does not mean that they cannot receive incoming connection in general. In fact, in the delivery architecture leaves can accept incoming connections for uploading contents)
- CANNOT publish/index objects.
- 

The Leaf node CONSUMER is composed by:

- AX\_QUERY is a service in the node which PRODUCES a query in the AXMEDIS format and sends it to the hub. The results are expected from the connection and are managed locally. The AX\_QUERY service in a leaf consumer node allows to :

- produce a new query;
- collect results from a response received from a remote hub.

### **Security and trust in the P2P network**

#### **Problem:**

*Identities of local and remote host must be verified by means of digital signatures. That implies the involvement of a external certification/supervisor authority. Hosts without a certified identity ARE NOT allowed to join the AXMEDIS community.*

#### **Solution:**

At the time being the security aspects in the network for searches/index is not completely addressed. In the next period we will investigate the integration with the security tools in the AXMEDIS framework. This work will be mainly driven by the FUPF partner.

Nevertheless, the current version of the prototype will be equipped with the possibility to build a network topology based on SSL sockets. This choice will allow only certified peers to join the network. **THIS IS A TEMPORARY SOLUTION** and will be replaced, refined or discarded according to a deeper analysis of the issues.

This way we don't have to build a centralized server for managing passwords and checking the identities of peers. The mechanism for authentication will work in the following way:

- At a given moment a peer P asks the CA of AXMEDIS to sign a certificate for P. This is performed only once (the certificate must be renewed after the expire time)
- P wants to join the network it tries to establish a SSL connection with another peer Q.
- P receives the Q's certificate, if this certificate is signed by CA then P can trust Q as a certified member of the AXMEDIS community
- Q receives the P's certificate, if this certificate is signed by CA then Q can trust P as a certified member of the AXMEDIS community
- The connection is up, messages are exchanged via SSL.

In the above steps there is no connection to a centralized server and the identification is performed in a decentralized and reliable way.

There are some issues to address:

- is this mechanism full compliant with the strict AXMEDIS policies about security?
- Is this mechanism too heavy in terms of computation? Every message has to be encrypted and decrypted for each hop in the network.

### **How to boot a node: mechanism to join the network**

#### **Problem:**

*One of the usual problems in a distributed system is how an entity (in our case a peer) can join the network without a previous knowledge of IP addresses of other participants?*

#### **Possible solutions:**

- a. Use multicast protocols to search other participants. Unfortunately this does not work outside LANs
- b. Knowing at least the address of another node in advance. This solution can possibly work but what happens if the known node goes down or if it changes IP address.
- c. Maintaining one or more known registrars in which special nodes (i.e. the hubs) register themselves at startup. This solution is similar to GWebCache of Gnutella2 networks and it proves to work well.

We chose the solution ‘c’ implementing a Web application able to receive HTTP request to get the list of current active hubs and to add a new hub to the list. The solution is detailed below:

In order to connect to the AXMEDIS P2P network and join it, AXMEDIS peers need a bootstrap and initial discovery mechanism.

Joining the P2P network means that a peer must connect successfully to at least one hub (super-peer) running on the network. To do that its IP and port is required.

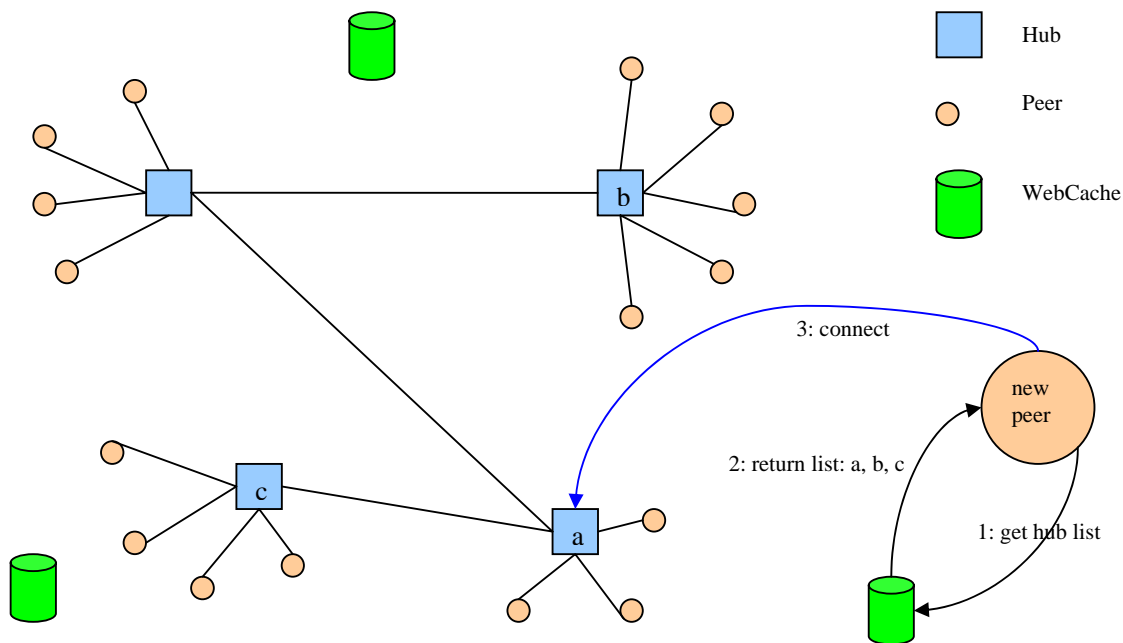
The mechanism that allows that kind of information knowledge is represented by WebCaches, which, essentially, are web applications reachable on well-know URLs. These applications maintain an updated list of hubs IP addresses (and ports) and accept HTTP requests for updating and getting them.

On connecting action, a peer, queries at least one WebCache, attempting to use the URL specified in the configuration file.

So, a WebCache performs the following main tasks:

- Maintain an updated list of hub peers in the P2P network; a list means a IP:port couple.
- Accept list requests by peers and answer with the list of updated hub peers.
- Accept update requests (only made by hubs) in order to update the maintained hubs list

In this way, the initial bootstrap and discovery of an AXMEDIS peer which is attempting to connect to the network is shown in the next diagram.



The picture shows a potential new AXMEDIS peer which attempts to connect to the P2P network to become a participant in the AXMEDIS environment.

The peer has a list of well-known WebCaches. The steps performed are:

1. The connecting peer contacts a WebCache in its list and make a request for hubs list
2. The WebCaches answers sending to the peer the hub list

3. The peer uses the hubs list and try to connect to at least one hub of the list. If a connection fails, the peer chooses another hub in the list a repeats the attempt.

### **Topology and architecture for content delivery**

#### **Problems:**

*Once a query is processed by the AXMEDIS P2P network, the user receives a list of query-hits. The user should be able to select one or more objects from the list and start the download.*

*An object should be downloaded simultaneously from more than one source if multiple sources are sharing the object.*

*The user should be able to start, suspend, cancel and resume the download of an object. Download are thus organized in sessions.*

*An object moved into the **AXMEDIS Database** becomes part of the business operations and digital rights of the object owner must be enforced.*

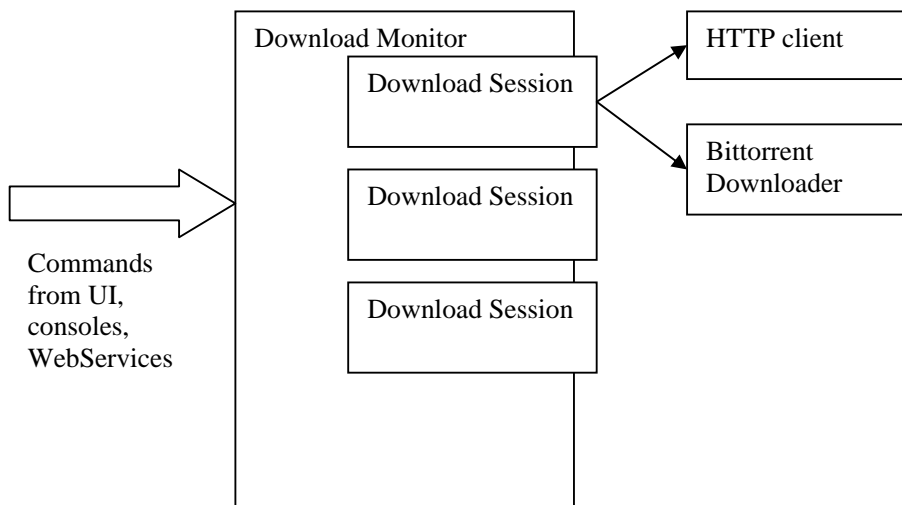
*The user should be able to configure the amount of bandwidth allocated to downloads.*

*The workload of the sources must be kept as low as possible in order to enable:*

- content providers with low bandwidth resources to participate in the network
- allow the delivery architecture to scale to thousands of simultaneous downloads for popular files.

#### **Solution:**

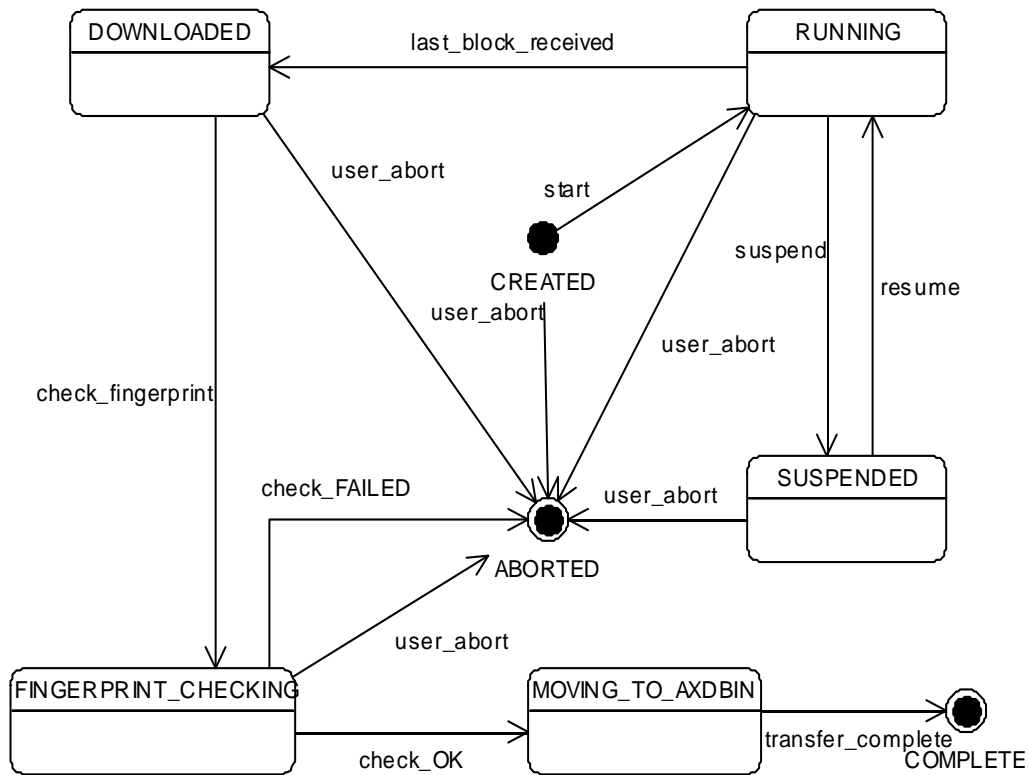
Such requirements will be addressed by the following architecture:



The Download Monitor is a component which takes care of downloading activities.

The Download Monitor can be invoked to start a download. Once a download is started the DownloadMonitor creates a DownloadSession which is committed to store and manage the state of the download.

The DownloadSession is a concurrent process which sends request of blocks to the sources in the network. It must allow the suspend/resume/abort and all operations cause a transition in the DownloadSessionState below.



State diagram for the DownloadSession

The real client for the downloading protocol may change according to the content:

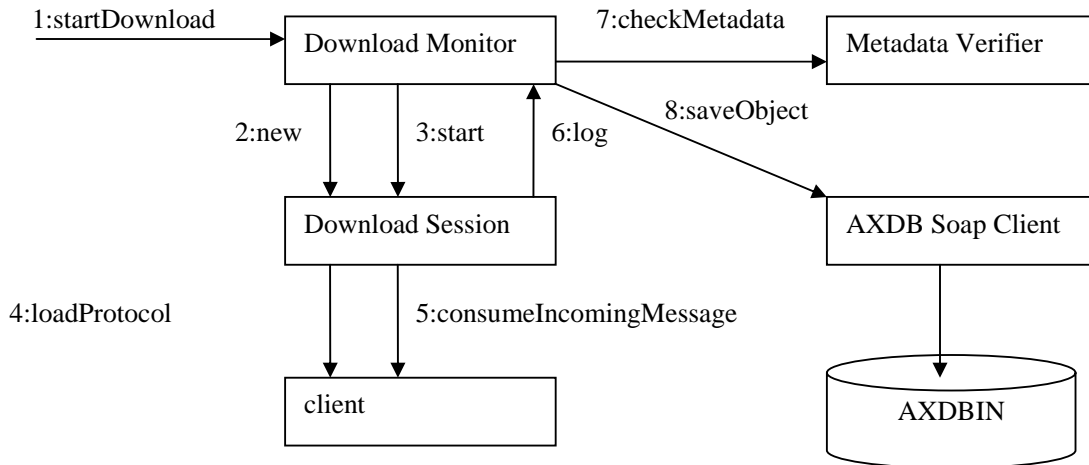
- If the content provider is distributing an object in a binary form which is unique for a given recipient (for instance, the object is encrypted and only one peer can open the object) then the protocol used for download is HTTP with resume.
- If the same binary file is available in more than one sources, then an HTTP from multiple sources will be used
- If the content provider is distributing the content via BitTorrent protocol then a BitTorrent client is used.

Below we report an updated version of the collaboration diagram already presented in DE3-1-2 part F.

#### **Collaboration Diagram: Downloading an Object**

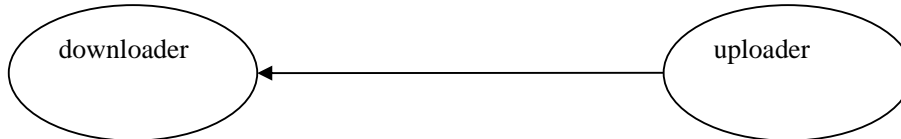
- 1 *startDownload*: this method asks the downloading of an object
- 2 *new*: a new Download Session is created.
- 3 *start*: starts the session
- 4 *loadProtocol*: the session load the right client to download the content (HTTP, Bittorrent, others).
- 5 *consumeIncomingData*: the session waits for incoming 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 Verifier
- 8 *saveObject*: the Download Monitor uses the AXDBIN client to connect to the AXDBIN and to save the new acquired object

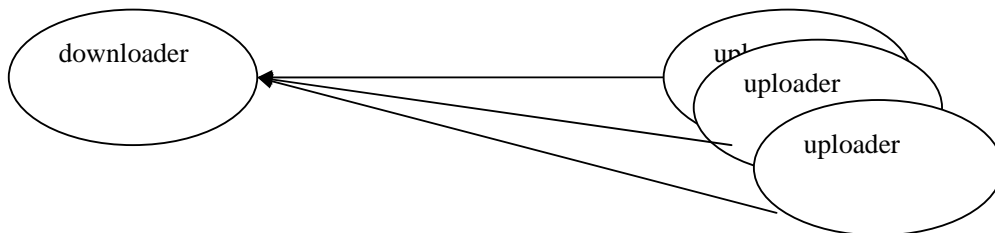


The network topology during a download session can assume the following configurations:

**One downloader to one uploader**

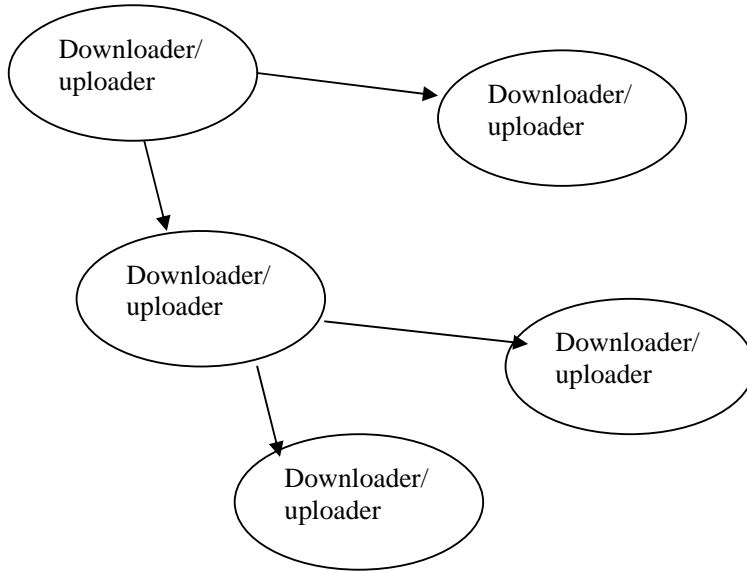


**One downloader to more uploader**





### Torrent



The advantages of a torrent have been explained in the sections regarding the **BitTorrent Protocol**

#### 3.2.2.4 Keeping peers informed about new objects or new versions

##### Problem:

*One of the peculiar characteristics of a B2B shared environment like AXMEDIS is the possibility to keep each participant up-to-date with respect to new versions of objects. In fact, if a factory A loads in its process an object O version 1 from a different factory B, it is important for A to be informed about new releases of O.*

##### Possible solutions:

Participants are connected in a P2P network so:

- 1) each peer releasing a new version (the provider) “floods” the network with an advertisement. This solution is too bandwidth consuming.
- 2) Each peer releasing a new version stores an advertisement in a one or more centralized servers. This solution is very efficient but relies on a centralized architecture and is not fault tolerant.
- 3) Each peer releasing a new version connects to other peers (the recipients) that in the past have loaded the old version. This solution has some issues: what happens if a recipient is offline for technical problems; the workload of this activity is mainly for the provider which has maintain a list of recipients and iterate over it.
- 4) Each peer releasing a new version can be periodically contacted by the other peers with a lightweight protocol in order to get news about an object. This solution is the opposite of (3). The workload of keep recipient informed is for recipients that must periodically check for updates.

We decided to adopt the solution (4) and to base the implementation of the solution on RSS news channels. In other words, once a peer A loads an object O from a peer B, A registers it self to a RSS channel hosted in B. That means that a RSS client running on A polls a RSS feed hosted on B (this is the way RSS news readers usually work). Addresses of the RSS feeds are in standard `http://hostname/feedname.xml` form because WE ARE ASSUMING THAT PARTICIPANTS OF THE B2B NETWORK HAVE FIXED HOSTNAMES SOLVABLE WITH STANDARD DNS. One could argue that a P2P overlay dynamic network cannot rely on DNS but we remind that the B2B network is composed by corporations and firms which are likely to have: long up-time, fixed IP and/or fixed domain/host names

### 3.3 Specification of T4.4.2: The query support into the AXEPTool distributed database (CRS4)

#### 3.3.1 Analysis of the specific user interface for technical queries (CRS4)

**Problem:** *The AXEPTool must provide a GUI for editing complex technical queries. Technical queries have to include:*

- xii. *details related to the description of components.*
- xiii. *costs and DRM rules, for each action a price, play, excerpts, redistribution, resizing, distribution on a different area, validity of the DRM rule and copyright coverage, etc.*
- xiv. *available languages if there is speaker speech or text.*
- xv. *range of age suggested.*
- xvi. *business model suggested.*
- xvii. *time of delivering delivery and availability in terms of first delivering, if not ready.*
- xviii. *type of delivery: on-line, offline, etc....*
- xix. *if on-line time of downloads or acquisition.*
- xx. *cultural level.*
- xxi. *subject, description of content with simple metadata.*
- xxii. *Textual description of subject and evolution.*

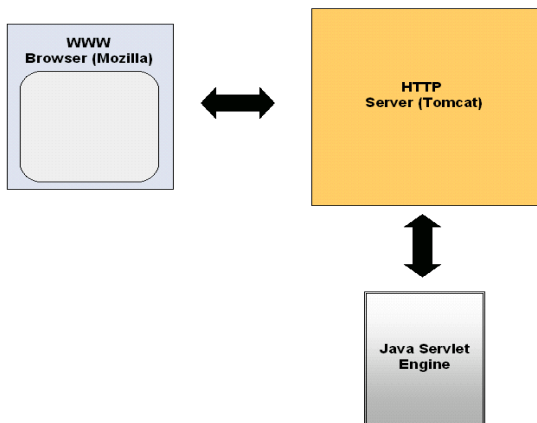
*The GUI must be used by all operators of a factory which are credited to access the AXEPTool. Thus, more user should be able to access the only P2P node in the firm.*

#### **Possible solutions:**

We envisioned two main solutions:

1. A lightweight client based on a GUI toolkit (for instance, wxWidget for C++) can be installed in each workstation and it would access to the single P2P daemon running in a dedicated server via HTTP or any other appropriate intranet protocol. This solution has the advantage to provide a good interactivity, the GUI can be embedded as a component in other C++ applications and it is quite simple to implement. The main disadvantage is that whenever a new version of the GUI client is released all installed copies must be updated with some manual or possibly automatic procedure.
2. A HTML Web based GUI is generated by an appropriate Web application running somewhere in the factory and able to send command and receive results to/from the P2P daemon. The advantage is that no dedicated software must be installed in the workstations because a simple Web browser can do the job and local updates are no longer needed. The disadvantages are that providing GUI with Javascript and HTML can be error-prone and time consuming for programmers and that interfaces cannot be re-embedded in other applications.

We decided to adopt solution (2) and the problem of reuse components in other c++/Java applications can be addressed by loading the Web GUI inside a HTML component like wxMozilla[17] for C++ or JREx[18] for Java. We have developed a prototype which is explained later in this document. Together with the prototype explanation also a short architecture overview of the system is provided.



### 3.4 Specification of T4.4.4/5 Analysis of Publication and Loading Modules (CRS4)

#### 3.4.1 Analysis and specification of Publication and Loading Module (CRS4)

##### 3.4.1.1 Introduction

The following sections reports on research activities of automatic Publication and Loading, analyzing scenarios, requirements, architecture, technical solutions and intermediate implementation. This section is mainly on automatic production of content via publishing/loading/downloading activities. Thus, the research is strictly related to the work performed in WP4.3 which involves the overall solution for content production and formatting. In this section you will find solution which will rely on a common framework of AXMEDIS Content Production (AXCP) rules and AXCP engines in which any activity like publishing an object is schedulable, scriptable and controllable by an external workflow manager. Publishing, Loading and Downloading are special cases of AXCP.

##### Automatic publication of objects

###### **Problem:**

*The actor wants to publish one or more objects in the P2P network*

###### **Solution:**

- 1 The Actor, through the Publication/Loading Rules/Selections, submits the Selection to the AXEPTool Active Publication Active Rules/Selections
  - the Selection becomes active
  - the Actor is allowed to modify the default activation period
- 2 According to activation periods the Publication Tool Engine of AXEPTool publishes each objects of the Selection on the AXEPTool OUT AXDB.

##### Automatic Download of objects

###### **Problem:**

*The user wants to download objects available on the P2P network following a given query (for instance all objects whose author is "Mister Brown" and year of publishing is "1970").*

###### **Solution:**

- 1 The Actor uses the AXCP editor and builds a new AXCP rule for downloading containing the query (such rule is also called Active Selection for P2P).

- 2 When the rule is scheduled for execution, the query is solved and a number of results are collected. If the objects are not already locally available or if are locally available but are not up-to-date one/more new download sessions are started (one for each object).
- 3 Download Monitor provides a GUI to manage file transfers.

Actually the solution does not address the integrity and correctness of the received object and metadata. This issue is going to be addressed in the next period.

The rule can contain a Javascript with some additional information like the max number of results to collect from a query.

Once a download session successfully terminates, the downloaded object is stored in the AXEPTool IN AXDB.

(Notice: we use the term *load* to denote the loading into the AXDB of a locally available object. We use the term *download* to denote the transfer of an object from the P2P network to the local peer).

### **Automatic updating of new versions**

#### **Problem:**

*In a automated content production process is likely to have downloaded an object in the past while in the network a new version of the object is available. The publishing and monitoring objects described in the sections above allows to know if a new version is available. Our problem is how to determine whether or not our AXEPTool must reload from the network the object or not.*

#### **Solution:**

- 1 The module ‘Publication and Monitoring Objects’ is informed of the new version.
- 2 Once a download rule performs its periodical execution it checks whether or not the new version is: result of the selection in the rule **andNot** a download session is already on going for that object **andNot** the object is already stored in the AXDBIN. If the above expression is true the download monitor is invoked to get the object.

Every time that an object is downloaded its integrity has to be verified in order to allow publishing its metadata and certifying that they are coherent with the description of the object.

### **Mapping different sets of metadata**

#### **Problem:**

*Objects becoming from different factories/peers may not have the same set of metadata. For this reason it may be necessary a process of “translation” from one set (the set of remote peer) to the local set of metadata before an object is loaded in the AXDB.*

#### **Solution:**

We designed a process of Metadata Mapping. 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 AXINDB they are not effectively acquired.

#### **Issue:**

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.

### 3.4.1.2 Architecture

This section gives an overview of the architecture of the prototype under development for Loading/Downloading and Publication activity inside the factory.

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

- JS\_Publisher
- Publisher Component
- Publication Rules
- AXDB Client
- AXDBOUT Client

#### Publisher Module

- 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 actualised selection
  - Unpublish() un-publishes the actualised 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.

When a publication rule is scheduled for execution the script could invoke the 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.

#### Publication Rules

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.

### **Loader Module**

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

### **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 is:

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 loader 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.

### **AXInfo Metadata Mapper**

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.

### **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 Factory 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
...	...

**Figure 1 – A mapping table.**

### **Mapping through a temporary AXOB**

As already pointed out, 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. 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.

### **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.

### **AXEPTool P2P Active Selections Module (Downloader Module)**

For simplicity this part is named “Downloader Module”.

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.

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 actualised selection of objects
    - The download rule is the Active Selection and is not a class, it is an instance of the AXCP Rule class.

### 3.4.2 Results on technical solution/architecture for exchange among databases, active selection for loading and publishing. (CRS4)

Considering architectural design and technical decisions, are hereinafter described technical solutions adopted for Publication Module and Loading Module.

Integration of fingerprinting system is not more concerning this area, but has to be considered related to Download Monitor within the DownloadSession (see FINGERPRINT\_CHECKING step).

#### 3.4.2.1 Publication Module

Publication Module (*publisher*) interacts with the following modules:

- *AXCP Rule Engine*, actualises the rule and starts the publisher through Spider mOnkey.
- *Publishing And Monitoring Objects*, to throw away a publishing event.
- *DBClient*, functionalities towards databases.
- *Database Interface*: gives AXMEDIS Object metadata information.
- *Protection Tool Engine*, the module which perform the protection of the AXMEDIS Objects.

Many of them are implemented as web services: Publishing and monitoring Objects, Database Interface and DBClient. In particular DBClient is a unified interface to a set of interfaces in DB Client Web Service interface (see further details).



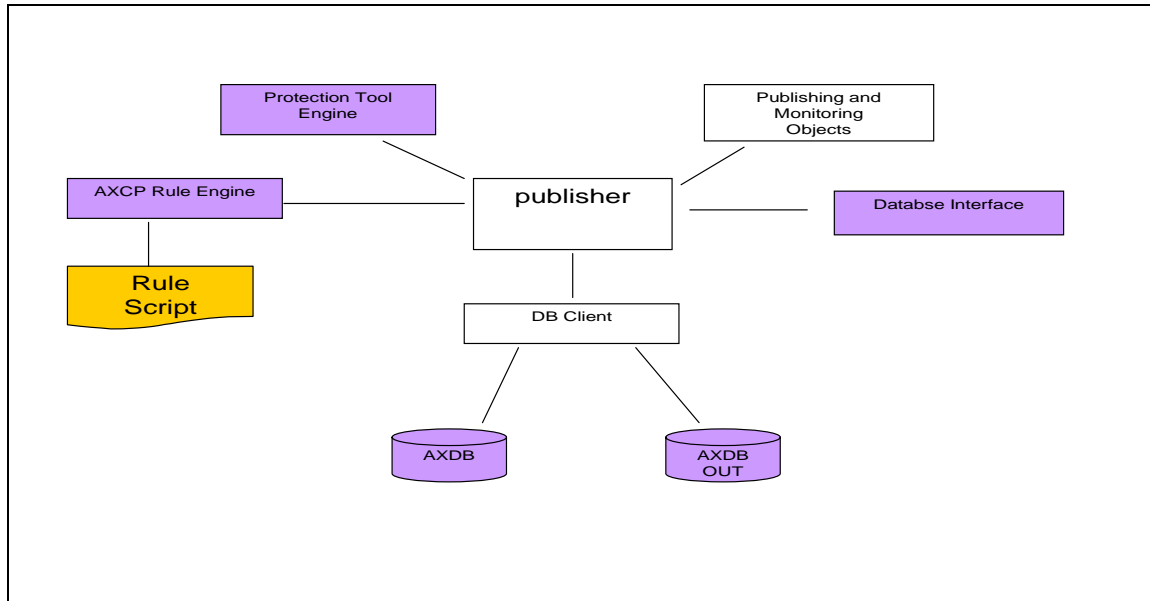


Figure 2 – Publication Module Architecture.

### 3.4.2.2 Loading Module

Loading Module (loader) interacts with the following modules:

- *AXCP Rule Engine*, actualises the rule and starts the publisher through Spider mOnkey.
- *DBClient*, functionalities towards databases.
- *Database Interface*: gives AXMEDIS Object metadata information.
- *AXMEDIS Object Manager*: acts operations on AXMEDIS Objects.
- *AXInfo Metadata Mapper*: acts the mapping of metadata.

Many of them are implemented as web services: Publishing and monitoring Objects, Database Interface and DBClient. In particular DBClient is a unified interface to a set of interfaces in DB Client Web Service interface (see further details).

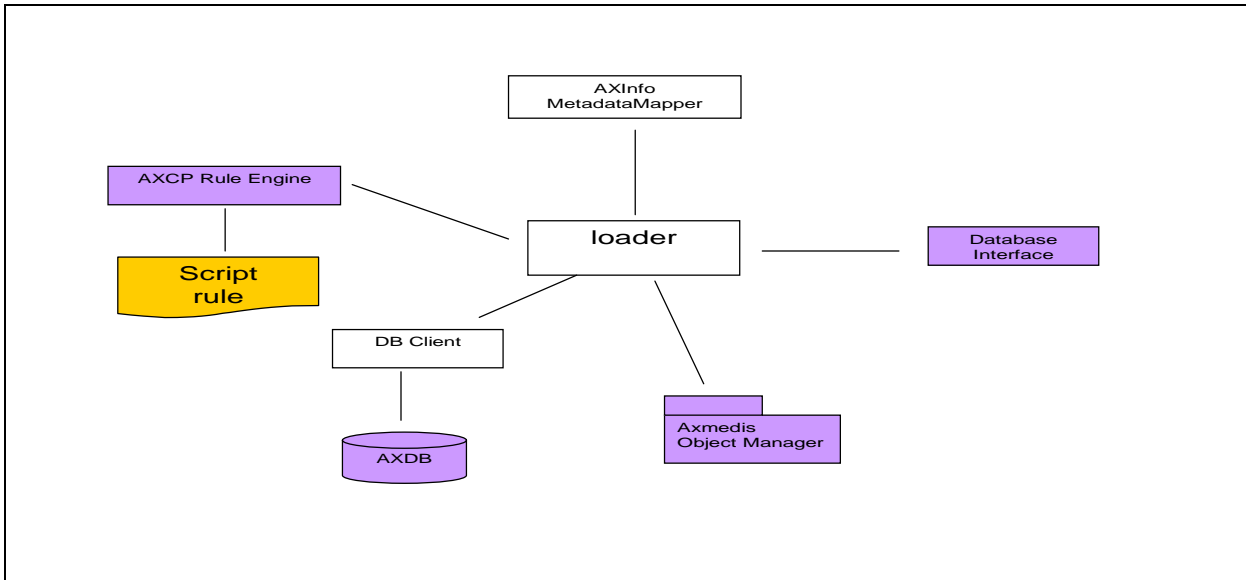


Figure 3 – Loading Module Architecture.

### 3.4.2.3 Downloading Module

Donloading Module (downloader) interacts with the following modules:

- *AXCP Rule Engine*, actualises the rule and starts the publisher through Spider mOnkey.
- *Database Interface*: gives AXMEDIS Object metadata information.
- *Publishing And Monitoring Objects*, to throw away a publishing event.
- *Database Interface*: gives AXMEDIS Object metadata information.
- *Download Monitor*: the engine performing downloading; gives information on downloading sessions.
- *AXEPTool Node Interface*: a web service which performs queries on the P2P network.

Many of them are implemented as web services: Publishing and monitoring Objects, Database Interface, Download Monitor and AXEPTool Node Interface.

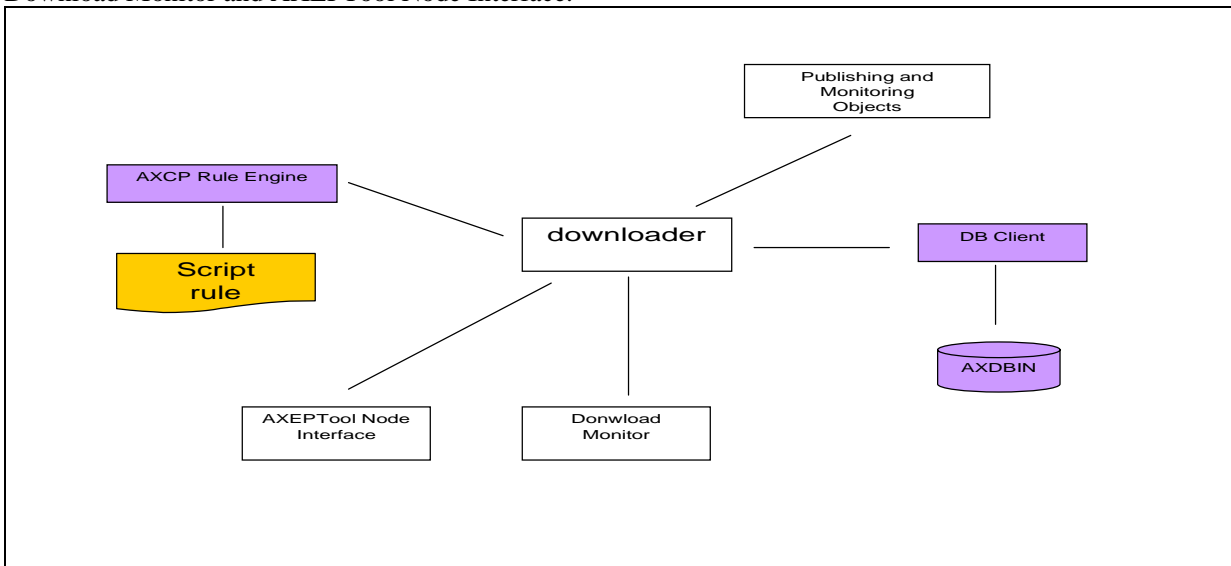
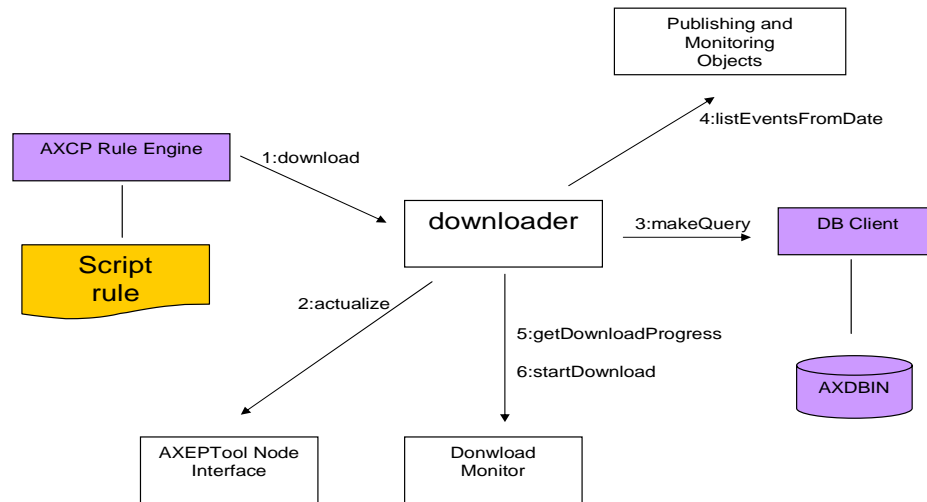


Figure 4 – Downloading Module architecture.

### Collaboration Diagram: downloading



**Figure 5 – Downloading Module collaboration diagram.**

Downloading execution involves the following steps (make attention that the AXCP Rule Engine executes the script but doesn't actualize the selection):

- *download*: the Downloader instance is commanded to download the objects.
- *actualize*: the Downloader actualizes the selection trough the AXEPTool Node Interface, which is able to make a query on the P2P network.
- *makeQuery*: the Downloader invokes an operation to the AXDBIN Soap Client in order to obtain a list of AXOB actually in AXDBIN.
- *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.
- *getDownloadProgress*: if an object is already under downloading by another rule, the Downloader doesn't have to download again.
- *startDownload*: the DownloadMonitor WebService is invoked and it starts downloading the AXOB.

## 4 Publication Module of the AXEPTool

### 4.1 Technical Details

reference to the AXFW location of the demonstrator	Private information
List of libraries used	<ul style="list-style-type: none"> <li>– Spider Monkey</li> <li>– wxWidgets</li> </ul>
References to other major components needed	
Problems not solved	<ul style="list-style-type: none"> <li>• .....</li> <li>• .....</li> </ul>
Configuration and execution context	<p>Publication Module is configurated through the <i>publisher.config</i> file:</p> <ul style="list-style-type: none"> <li>– dbclient_uri=http://server/dbclientWS</li> <li>– publishingmonitoring_uri=http://server/publishingWS</li> <li>– axdb_folder=./axdb</li> <li>– axdbout_folder=./axdbout</li> </ul> <ul style="list-style-type: none"> <li>• The testing unit can be executed by running the file:             <ul style="list-style-type: none"> <li>– JS_Publisher.exe</li> </ul> </li> </ul>
Programming language	C++

### 4.2 Description of the purpose

The purpose of this demonstrator is to show the Publication Module and its main functionalities; the aim is to allow to test the module without the integration (in a self contained unit) with the Rule Engine (so without SpiderMonkey).

### 4.3 Functionalities

- Receives a selection of objects actualized by the Rule Engine
- Publishes one or more new AXMEDIS Objects or new versions of already published AXMEDIS Objects
- Copies the published objects from AXDB to ADBOUT resources folders

### 4.4 Snapshots

The component has not any graphical user interface thus snapshots are dragged from a text-based output.

#### 4.4.1 Receives a Publication Rule actualized by the Rule Engine

Objects becoming from rule: OBJ-1,OBJ-2,OBJ-5,OBJ-6 ---

> Rule ID: 19 ---

> JS\_Publisher (createObjsFromString)---

>> Tokens ---

>>> objects: (4)

>>>> 0: OBJ-1

>>>> 1: OBJ-2

>>>> 2: OBJ-5

>>>> 3: OBJ-6

#### 4.4.2 Publish one or more new AXMEDIS Objects or new versions of already published AXMEDIS Objects

```

Managing obj(0): OBJ-1
> JS_Publisher (protect)---
> JS_Publisher (saveObjs)---
> --- PublicationSupportClient -> (instantiated)---
> --- SaverClient -> (instantiated)---

>> Saving obj: OBJ-1
> --- SaverClient -> (commitSync)--- URL: OBJ-1 user: px passwd: px-passwd
>> Results from server:
>>> return CODE = 1
>>> message = 0x22e968
>>> result = 1
>>> version = 2
> JS_Publisher (copyOBJ)---
>> copying ./axdb/OBJ-1 to ./axdbout/OBJ-1
>> copy OK
> JS_Publisher (producePublicationEvent)---
> PublishingAndMonitoringClient::init()
>> ProducePublicationEvent obj: OBJ-1
> object: (OBJ-1)
>> -----

```

#### 4.4.3 Copy the published objects from AXDB to ADBOUT resources folders

```

>JS_Publisher (copyOBJ)---
>> copying ./axdb/OBJ-1 to ./axdbout/OBJ-1
>> copy OK

```

## 5 Loading Module of the AXEPTool

A featured demonstrator for Loading Module of the AXEPTool is not available yet.

### 5.1 Technical Details

reference to the AXFW location of the demonstrator	A path in the CVS for example: <a href="https://cvs.axmedis.org/repos/Framework/">https://cvs.axmedis.org/repos/Framework/</a>
List of libraries used	
References to other major components needed	
Problems not solved	<ul style="list-style-type: none"> <li>• .....</li> <li>• .....</li> </ul>
Configuration and execution context	
Programming language	

## 6 AXEPTool Active Selection for P2P Module

A featured demonstrator for Active Selection in P2P is not available yet.

## 6.1 Technical Details

reference to the AXFW location of the demonstrator	A path in the CVS for example:
List of libraries used	
References to other major components needed	
Problems not solved	<ul style="list-style-type: none"> <li>• .....</li> <li>• .....</li> </ul>
Configuration and execution context	
Programming language	

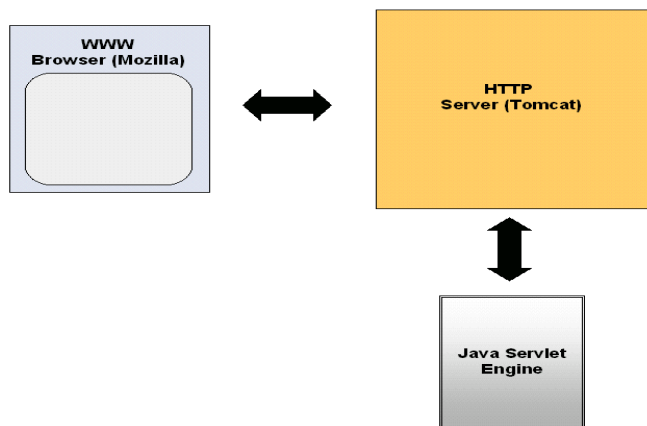
## 7 AXMEDIS Query GUI

The purpose of this demonstrator is to show the functionalities of the Query User Interface in the B2B distributed environment of AXMEDIS. **It provides a web-like user interface, based on a client server architecture composed by two different tiers.**

As it is showed in the following picture the client tier is a JSP page with HTML, CSS and Javascript elements.

The Server tier is a Java Servlet Application deployed in a Tomcat Application Server.

Responsible: Massimo Deriu



### 7.1 Technical Details

reference to the AXFW location of the demonstrator	Private information
List of libraries used	Apache Tomcat libraries for embedded web container Sun JWS DP and Apache Axis libraries for web services (total migration to Axis)
References to other major components needed	
Problems not solved	Real connection to Web Services. For simplicity a fake local proxy is used. Re-editing of existing query. Loading data from configuration file. Tree modality. Improve the functionalities of the Result GUI and of the Download monitor GUI.

Configuration and execution context	
Programming language	Java, HTML, CSS, Javascript

## 7.2 Lists of Functionalities

In this preliminary version the Query User Interface provides an appropriate GUI able to produce technical queries compliant to the AXMEDIS query model.

Note:

The QUI will be available into two different modalities: Flat and Tree:

the Flat QUI is a form-based UI which allows common “fill-the-form and send” operations

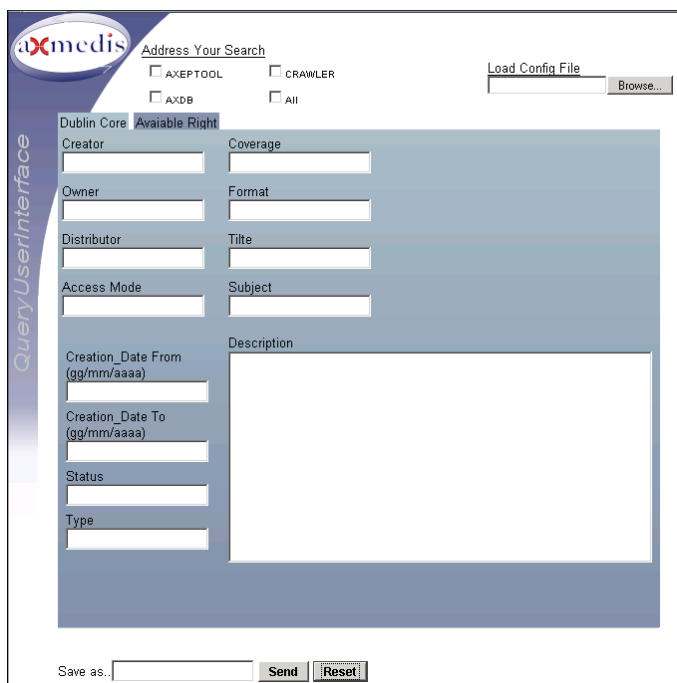
the Tree QUI is a tree-based UI which allows the composition of queries in a structured form with arbitrary complexity. The prototype is able to provide ONLY the Flat modality. The Tree modality is currently under development.

The functionalities implemented allow to:

- Compose an AXMEDIS Query
- Submit it on different locations
- Show Results
- Manage a download session

## 7.3 Snapshots and explanation

The GUI is organized into three main section: heading, form and actions:



The screenshot shows the AXMEDIS Query User Interface (QUI) with the following sections:

- Heading:** Contains a multiple choice checkbox to address the AXMEDIS query on different locations. It includes options for AXEPTOOL, CRAWLER, AXDB, and ALL, along with a 'Load Config File' button and a 'Browse...' button.
- Form:** Contains two tabs: **Dublin core** and **Available Right**. The **Dublin core** tab is active, showing fields for Creator, Owner, Distributor, Access Mode, Coverage, Format, Title, Subject, Creation Date From (gg/mm/aaaa), Creation Date To (gg/mm/aaaa), Status, and Type. The **Available Right** tab is also visible.
- Actions:** Contains the buttons to reset all the values filled in the form, or to send those values to the server tier. The buttons are labeled 'Send' and 'Reset'.

The steps to write a query are:

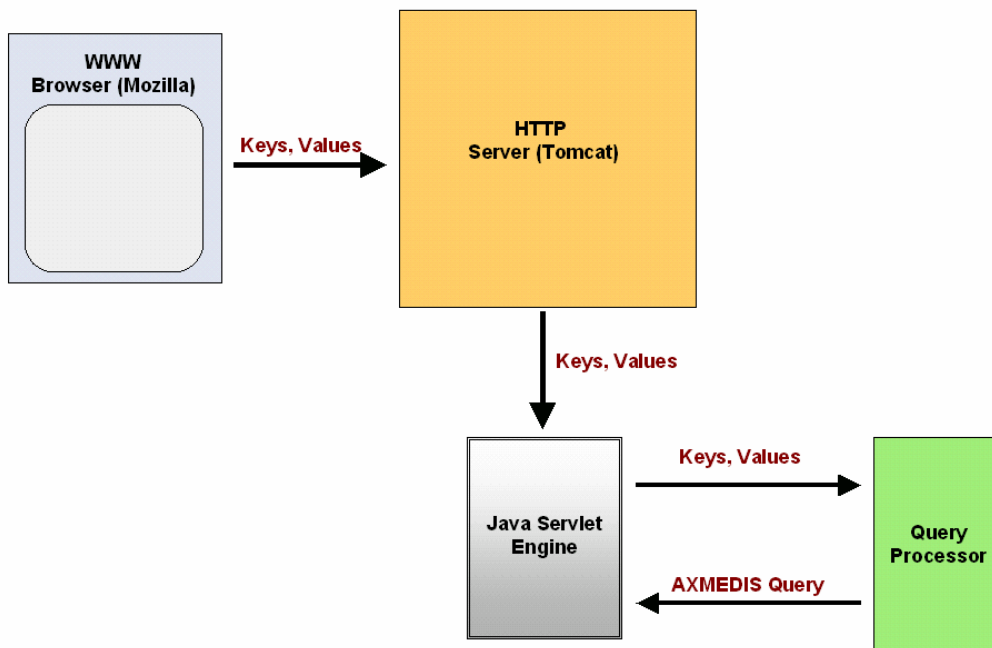
- Check one or more choice in the Heading section

- Fill the values in the Form section
- Click the button in the Actions section

All the information enclosed in the form, are than sent to the server tier. The logic of the server tier is managed by a Java Servlet that has three main task:

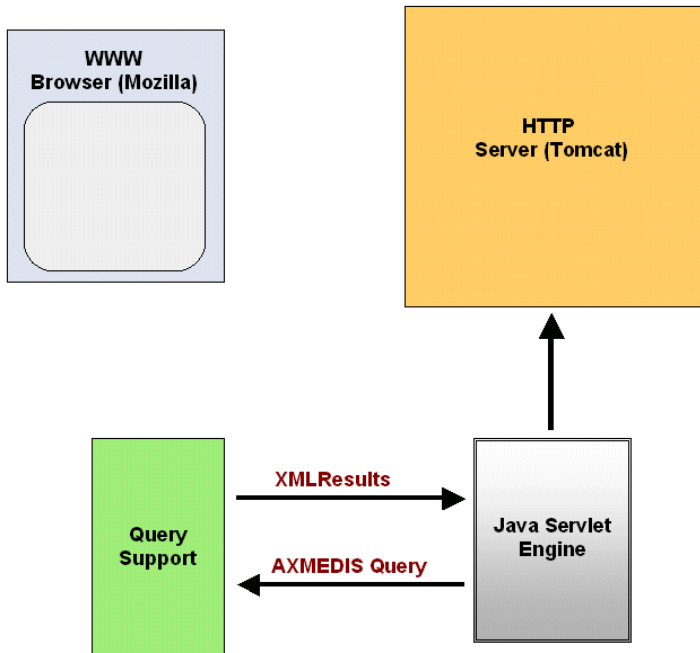
- Process the form's data in order to compose a query compliant to the AXMEDIS Query model.
- Forward the AXMEDIS Query to the Query Support Web Service
- Process the XML Result and send it back to the browser

### 7.3.1 Process the form's data in order to compose a query compliant to the AXMEDIS Query model.

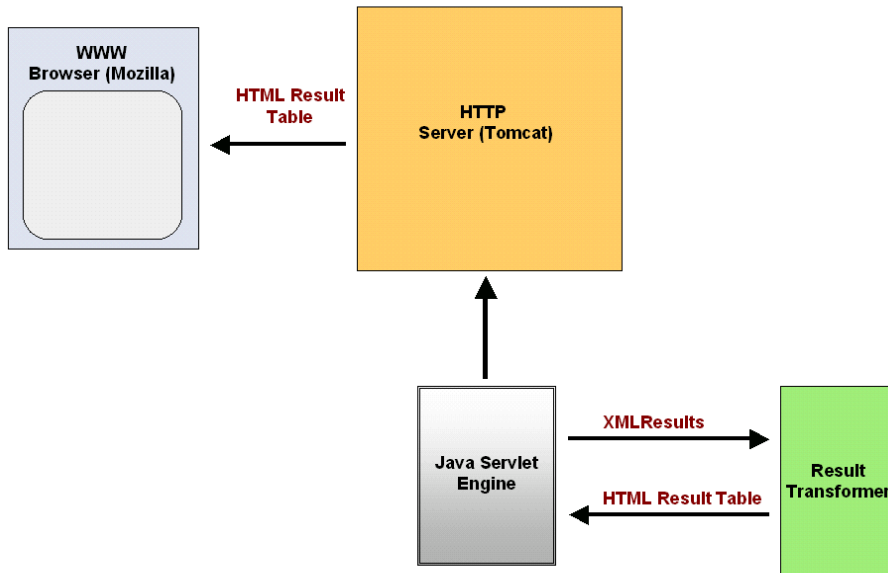




### 7.3.2 Forward the AXMEDIS Query to the Query Support Web Service



### 7.3.3 Process the XML Result and send it back to the browser



### 7.3.4 Query Results

**axmedis** Query Results

**AXEPTOOL:**

AXOID: axoid\_001  
 Name = Obj\_001  
 Type = Type\_001  
 Modification Date = 01/01/01  
 uri = http://localhost:8080/QueryUserInterface\_Integrated/Download/obj001.axm  
 Download

**CRAWLER:**

TAXOID: taxoid\_002  
 Name = Obj\_002  
 Type = Type\_002  
 Modification Date = 02/02/02  
 uri = http://localhost:8080/QueryUserInterface\_Integrated/Download/obj002.axm  
 Download

**AXDB:** 726x726

AXOID: axoid\_003  
 Name = Obj\_003  
 Type = Type\_003  
 Modification Date = 03/03/03  
 uri = http://localhost:8080/QueryUserInterface\_Integrated/Download/obj003.axm  
 Download

Via the results's GUI is possible to start a download session (see below)

**axmedis** DownloadMonitor for: **axoid\_001**

Start Stop Suspend Resume Abort << Back Info

Mozilla

State	Bandwidth	Peers	Progress	Session ID	Source
getState()	getBandwidth()	getPeers()	getProgress()	getSessionID()	getSource()

Close

## 8 AXEPTool Core Components

This section describes a set of demonstrators which are so strictly related that cannot be presented as separate applications but are bundled in a single Java application which contains:

- Low Level Virtual Database
- Publishing and Monitoring Objects
- AXEPTool Monitor

The AXEPTool core components are not an interactive application so to demonstrate their features we implemented an interactive application called AXEPTool console which is text-based

reference to the AXFW location of the demonstrator	Private information
List of libraries used	<ul style="list-style-type: none"> <li>♣ Apache Tomcat libraries for embedded web container</li> <li>♣ Sun JWS DP and Apache Axis libraries for web services (total migration to Axis)</li> <li>♣ Informa RSS libraries</li> </ul>
References to other major components needed	
Problems not solved	<ul style="list-style-type: none"> <li>• . P2P Network topology refinement (peers and super-peers)</li> <li>• . Integration with C++ modules (Publisher,...)</li> <li>• . Integration with Query User Interface</li> <li>• . Integration with Database Area Web Services</li> <li>• . Development of AXEPTool Web Services and GUI</li> <li>• . Adoption of a more performing download protocol (i.e. BitTorrent)</li> </ul>
Configuration and execution context	<p>AXEPTool prototype is configured through the config.properties file, editing:</p> <p>VDBNode.peerID=&lt;peerID&gt;:&lt;listening-port&gt; i.e. : VDBNode.peerID=A-Factory:4444</p> <p>EMB-TOMCAT.URL=http://&lt;ip or host name&gt;:8080 i.e. :EMB-TOMCAT.URL=http://hostA.factory.com:8080</p> <p>AXEPTool Prototype can be executed in two contexts:</p> <p>Win XP, executing the console.bat file Win 2000, executing the console.sh script under a Cygwin console</p>
Programming language	Java

### 8.1 AXEPTool Core Features

#### 8.1.1 Virtual Database Features

- connect to the P2P Network: the factory joins the network and become a participant of the B2B process

- trace connections and all connected peers: mainly implemented for network “maintenance”, allows to show all connected participants in the network
- publish a content: allows a factory to make a content public and available to other factories connected to P2P
- query the network for published contents: query facilities for published objects searching task

### 8.1.2 Monitor Features

- download a published content: using query results the factory can download the selected object from the network
- monitor downloads status: allows to monitor the progress and the status of scheduled downloads
- exposes a Soap Interface

### 8.1.3 Publishing and Monitoring Objects Features

- Manages registration and client protocol to a remote factory for RSS feeds about new versions of a downloaded object
- Push news on a local RSS feed about a local object for remote peers.

## 8.2 AXEPTool Console Screenshots

### 8.2.1 Connecting to the P2P network and tracing connections and other connected peers

```
pintux@vir /cygdrive/d/CRS4Researches/Axmedis/DEMO-meeting-firenze/apps/axeptool_console/bin
$ ./con
config.properties      console-commands.properties  console.bat            console.sh
pintux@vir /cygdrive/d/CRS4Researches/Axmedis/DEMO-meeting-firenze/apps/axeptool_console/bin
$ ./console.sh
Client socket factory is=javax.net.DefaultSocketFactory
Client socket factory is=javax.net.DefaultSocketFactory
Client socket factory is=javax.net.DefaultSocketFactory
Server socket factory is=javax.net.DefaultServerSocketFactory@60aeb0
AXEPTool Virtual DB Console
Version 0.4a
l-set-2005 12.45.41 com.sun.xml.rpc.server.http.JAXRPCContextListener contextInitialized
INFO: JAXRPCServlet12: JAX-RPC context listener initializing
l-set-2005 12.45.42 com.sun.xml.rpc.server.http.JAXRPCContextListener contextInitialized
INFO: JAXRPCServlet12: JAX-RPC context listener initializing
Processing file ./lib/jakarta-tomcat-5.5.9-embed/webapps/saver/deploy.wsdd
<Admin>Done processing</Admin>
Processing file ./lib/jakarta-tomcat-5.5.9-embed/webapps/querysupport/deploy.wsdd
<Admin>Done processing</Admin>
A-Factory:4444>
A-Factory:4444>
A-Factory:4444>connect mettiu.crs4.it:4444
Socket is=class java.net.Socket
->B-Factory:4444
A-Factory:4444>who
A-Factory:4444>received:RESPONSE:F=WHO S=0 M=5992922623094155763 from B-Factory:4444
Discovered host by Who services are:
peer:B-Factory:4444
A-Factory:4444>connections
->B-Factory:4444
A-Factory:4444>
```

**"A" Factory AXEPTool console starts**

**connect to P2P Network**

**trace of all connected peers on P2P**

**trace of all direct connections**

### 8.2.2 AXMEDIS Content publication

- A Factory named “B” has made a connection to the network
- B-Factory performs a publication of an AXMEDIS object named objA.axm

```
B-Factory:4444>
B-Factory:4444>
B-Factory:4444>
B-Factory:4444>publish file:///AXMEDIS/Axmedis-demo/files/objA.axm
publishing file:file:///AXMEDIS/Axmedis-demo/files/objA.axm
file:/AXMEDIS/Axmedis-demo/files/objA.axm committed to AXDBOUT
file:/AXMEDIS/Axmedis-demo/files/objA.axm copied to .\lib\jakarta-tomcat-5.5.9-embed\webapps\ROOT\objA.axm
B-Factory:4444>
B-Factory:4444>
```

Object publication

After publication, the object is available in the P2P network to other factories, so a query can match it and can be included in results.

### 8.2.3 Querying the network

“A” Factory queries the network for objects of any type. Query results are received from responding peers.

```
/cygdrive/d/CRS4Researches/Axmedis/DEMO-meeting-firenze/apps/axeptool_console/bin
A-Factory:4444>who
A-Factory:4444>received:RESPONSE:F=WHO S=0 M=4560226011319738854 from B-Factory:4444
Discovered host by Who services are:
peer:B-Factory:4444

A-Factory:4444>
A-Factory:4444>
A-Factory:4444>
A-Factory:4444>query query.xml
file containing the query is:query.xml
query is:<?xml version="1.0" encoding="UTF-8"?>
<query xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="QUERY-v1-6.xsd">
  <source>
    <location>AXEPTOOL</location>
  </source>
  <AXinfoQuery>
    <querycondition>
      <nesting>
        <test>
          <field>MEDIA</field>
          <operator>EQ</operator>
          <value>VIDEO</value>
        </test>
        <or/>
        <test>
          <field>MEDIA</field>
          <operator>EQ</operator>
          <value>AUDIO</value>
        </test>
        <or/>
        <test>
          <field>MEDIA</field>
          <operator>EQ</operator>
          <value>TEXT</value>
        </test>
      </nesting>
    </querycondition>
  </AXinfoQuery>
</query>
A-Factory:4444>received:RESPONSE:F=UNKNOWN S=0 M=7579121614270882479 from B-Factory:4444
sysout -- received query results:<queryresults xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="/AXQS-Result-v-1-1.xsd"><AXobject><AXEPTOOL><peers><peerID>http://mettiu.crs4.it:8080</peerID></peers><AXOID>objA.axm</AXOID></AXEPTOOL></AXobject></queryresults>
<queryresults xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="/AXQS-Result-v-1-1.xsd"><AXobject><AXEPTOOL><peers><peerID>http://mettiu.crs4.it:8080</peerID></peers><AXOID>objA.axm</AXOID></AXEPTOOL></AXobject></queryresults>
A-Factory:4444>
A-Factory:4444>
```

Querying for contents

Query Results from P2P

### 8.2.4 Contents Downloading and downloads monitoring

“A” Factory, using query results, performs an object download and monitor its status

```

</test>
<or/>
<test>
  <field>MEDIA</field>
  <operator>EQ</operator>
  <value>AUDIO</value>
</test>
</nesting>
</querycondition>
</AXinfoQuery>
</query>
A-Factory:4444>received:RESPONSE:F=UNKNOWN S=0 M=378975775166827362 from B-Factory:4444
sysout -- received query results:queryresults xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="/AXQS-Result-v-1-1.xsd"><AXobject><AXEPTOOL><peers><peerID>http://mettiu.crs4.it:8080/</peerID></peers><AXOID>objA.axm</AXOID></AXEPTOOL></AXobject></queryresults>
<queryresults xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="/AXQS-Result-v-1-1.xsd"><AXobject><AXEPTOOL><peers><peerID>http://mettiu.crs4.it:8080/</peerID></peers><AXOID>objA.axm</AXOID></AXEPTOOL></AXobject></queryresults>
A-Factory:4444>
A-Factory:4444>
A-Factory:4444>
A-Factory:4444>get http://mettiu.crs4.it:8080/objA.axm
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 13% 14488109/108006934 RUNNING
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 32% 35158189/108006934 RUNNING
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 45% 49490809/108006934 RUNNING
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 60% 65632189/108006934 RUNNING
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 77% 84013109/108006934 RUNNING
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 96% 104666649/108006934 RUNNING
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 100% 108006934/108006934 COMPLETE
A-Factory:4444>
A-Factory:4444>

```

request for an object download

Monitoring of downloads status

### 8.2.5 Monitor the network for updates of downloaded objects.

“B” Factory performs a new publication of an updated version of the AXMEDIS object named “objA.axm”, already downloaded by “A” Factory.

Result: “A” Factory is notified about the update action on “objA.axm”

```

</test>
<or/>
<test>
  <field>MEDIA</field>
  <operator>EQ</operator>
  <value>TEXT</value>
</test>
</nesting>
</querycondition>
</AXinfoQuery>
</query>
A-Factory:4444>received:RESPONSE:F=UNKNOWN S=0 M=378975775166827362 from B-Factory:4444
sysout -- received query results:queryresults xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="/AXQS-Result-v-1-1.xsd"><AXobject><AXEPTOOL><peers><peerID>http://mettiu.crs4.it:8080/</peerID></peers><AXOID>objA.axm</AXOID></AXEPTOOL></AXobject></queryresults>
<queryresults xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="/AXQS-Result-v-1-1.xsd"><AXobject><AXEPTOOL><peers><peerID>http://mettiu.crs4.it:8080/</peerID></peers><AXOID>objA.axm</AXOID></AXEPTOOL></AXobject></queryresults>
A-Factory:4444>
A-Factory:4444>
A-Factory:4444>
A-Factory:4444>
A-Factory:4444>get http://mettiu.crs4.it:8080/objA.axm
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 13% 14488109/108006934 RUNNING
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 32% 35158189/108006934 RUNNING
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 45% 49490809/108006934 RUNNING
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 60% 65632189/108006934 RUNNING
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 77% 84013109/108006934 RUNNING
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 96% 104666649/108006934 RUNNING
A-Factory:4444>monitor
Local download directory is:./downloaded
[0] http://mettiu.crs4.it:8080/objA.axm 100% 108006934/108006934 COMPLETE
A-Factory:4444>
A-Factory:4444>new item on channel B-Factory:4444 : objA.axm, version=8
A-Factory:4444>

```

Update occurred, notification performed

## 9 AXEPTool P2P WebCache

This section describes a component needed by peers to discover one or more other hub participants in the network. The component is a Web application deployable as Web Archive (war) in a Tomcat servlet engine.

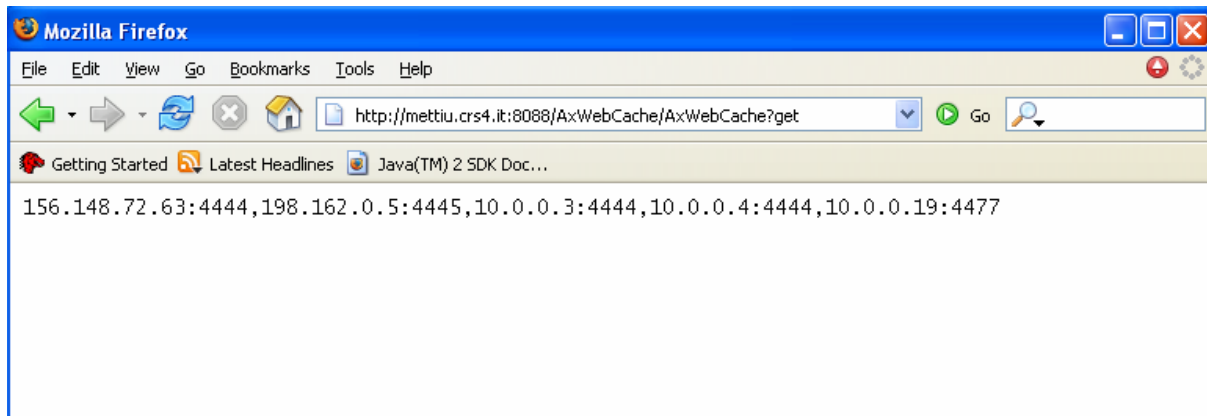
reference to the AXFW location of the demonstrator	Private information
List of libraries used	<b>Javax.servlet V2.4 only for building</b>
References to other major components needed	
Problems not solved	Should handle the age of inserted entries
Configuration and execution context	The component must be deployed as Web Application in a Tomcat engine
Programming language	Java

### 9.1 AXEPTool P2P WebCache Features

- Lists the entries (IP address and port) of hubs currently active in the network
- Receives and updates new entries from hubs.

### 9.2 AXEPTool P2P WebCache Screenshots

The screenshot shows a Web page containing entries as proof-of-concept. In the normal operating mode NO GUI is provided with this service as HTTP requests are silently managed by AXEPTools to find hubs. Entries are showd as a list of (IP:port) and are comma separated .



## 10 Bibliography

- [1] "SETI@home",  
<http://setiathome.ssl.berkeley.edu/>, University of California. Space Sciences Laboratory, 1999.
- [2] "Distributed.net", <http://www.distributed.net>
- [3] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A scalable peer-to-peer lookup service for Internet applications". Technical Report TR-819, MIT, March 2001
- [4] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. "Freenet: A Distributed Anonymous Information Storage and Retrieval System". *Lecture Notes in Computer Science*, volume 2009, year 2001.
- [5] B. Cohen, "Incentives Build Robustness in BitTorrent", *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*. 2003.
- [6] Article on Napster published in Wikipedia. <http://en.wikipedia.org/wiki/Napster>
- [7] Gnutella Protocol Development. <http://www.the-gdf.org>
- [8] [http://www.gnutella2.com/index.php/Node\\_Types\\_and\\_Responsibilities](http://www.gnutella2.com/index.php/Node_Types_and_Responsibilities)
- [9] Limewire Web site. <http://www.limewire.com/>
- [10] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM. Volume 13 , Issue 7 (July 1970), Pages: 422 - 426. Year of Publication: 1970 by ACM Press.
- [11] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001
- [12] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In Proceedings of IPTPS02, Cambridge, USA, March 2002.  
<http://www.cs.rice.edu/Conferences/IPTPS02/>.
- [13] Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch, Boon Thau Loo, Scott Shenker, and Ion Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. In Proc. of the 1st International Workshop on Peer-to-Peer Systems (IPTPS), March 2002.
- [14] GNU WGET Web page. <http://www.gnu.org/software/wget/wget.html>
- [15] Onion Networks. Swarming Technology. <http://onionnetworks.com/technology/swarming/>
- [16] eMule-Project.net - Official eMule Homepage. <http://www.emule-project.net/>
- [17] wxMozilla - Embedding Mozilla in wxWindows. <http://wxmozilla.sourceforge.net/>
- [18] JREx - The Java Browser Component. <http://jrex.mozdev.org/>



## 11 Glossary

- **AXCP:** AXMEDIS Content Production
- **AXDB:** AXMEDIS DataBase.
- **AXDBIN:** AXMEDIS DataBase IN, a DB provided with query support where objects loaded from the network are indexed
- **AXDBOUT:** AXMEDIS DataBase OUT, a DB provided with query support where local objects shared in the p2p network are indexed. In Hub peers the AXDBOUT can index objects of other peers.
- **AXEPTool:** In the AXMEDIS framework, the modules/components/tools aimed at production and distribution over a P2P network.
- **AXOB:** AXMEDIS Object
- **B2B:** Business-to-Business
- **B2C:** Business-to-Consumer
- **CA:** Certification Authority
- **CSS:** Cascading Style Sheets
- **DBIN:** see *AXDBIN*
- **DBOUT:** see *AXDBOUT*
- **DHT:** Distributed Hash Table
- **DRM:** Digital Rights Management
- **GUI:** Graphical User Interface
- **JSP:** Java Server Pages
- **JWSDP:** Java Web Services Development Pack
- **NAT:** Network Address Translator. A service which dynamically re-assign IP addresses to IP packets in order to use one IP public address with more than one local machine. It is often used in conjunction with firewalling to hinder access from outside a local network.
- **P2P:** Peer-to-Peer. An architecture in which all participants can act both as servers and as clients.

**PandM:** Publishing and Monitoring Object. A sub-module of the AXEPTool devoted to keep peers informed about new objects' versions.

**QUI:** Query User Interface. A GUI for editing queries.

**RIAA:** Recording Industry Association of America

**RSS:** Multiple definitions according to versions: Rich Site Summary (RSS 0.91); RDF Site Summary (RSS 0.9 and 1.0); Really Simple Syndication (RSS 2.0).

**SOAP:** Simple Object Access Protocol, is a computer standard defining a message format used by web services.

**TTL:** Time to live. It is a datum embedded in IP and above protocol layers which is decreased every time a packet, message, or datagram is routed. This is to prevent infinite loop-propagation of packets/messages/datagrams across the network

**UDP:** User Datagram Protocol (also known as Unreliable Datagram Protocol).

**VDB:** Virtual Database. A sub-module of the AXEPTool devoted to file sharing.

## 12 Technical Appendix

### 12.1 The core protocol and its extensions

First class entities in the network are nodes, connections, and messages.

#### Message

A message is the atomic entity of information exchange among nodes. The message has a header which is a fixed-length bunch of data, and a payload which contains the service-dependent information. The message header is structured as follows:

- **ID**, unique identifier for messages. Due to the lack of a centralized authority, message IDs are assigned randomly as 8 byte numbers. The probability that two nodes assign the same ID to two different messages is equal to  $2^{-64}$  (about  $10^{-61}$ ). The overall probability that a message ID is used more than once depends on the volume of messages produced.
- **messageType**, any message must be one of the followings: REQUEST, RESPONSE. A node behaves accordingly to the message type. A message of type RESPONSE has a field requestID!=0 which allows to discover which request the response is relevant to.
- **functionID**, any message has a functionID that numbers the protocol extension which manages that type of message payload. For instance, if a request message has functionID=SEARCH the node must implement the SEARCH extension in order to produce a response for that message. Otherwise, if the node cannot manage the functionID the message is simply routed forward or backward according to the routing algorithm.
- **length**, the field length contains the payload size in bytes.

A message header does not contain any information about the node that produced the message, nor about the nodes that routed the message, nor about the nodes that eventually produced a response. This fact leads to anonymity between clients and servers at this level of specification. Nevertheless, if an extension wants to specify the address or the Ids of sender/recipients of the message this could be done in the extended part of the protocol which is always stored in the payload of the message.

In the following of the document we adopt the notation

$REQ = REQUEST(F, P)$

to denote REQ as a message with messageType=REQUEST, functionID=F, and payload=P.

We adopt the notation  $RES = RESPONSE(REQ, F, P2)$  to denote RES as a message with messageType=RESPONSE, requestID=messageID of REQ, functionID=F and payload=P2. Moreover, we adopt the notation  $P=()$  to denote that P is an empty payload (zero-byte array).

### Connection

Unless differently specified, AXMEDIS peers will connect to exchange queries and indexes using standard TCP/IP connections. Messages are routed along the connections. Any connection is a bi-directional pipe of data: messages travel from an end to the other and vice-versa.

The logical topology of a P2P network is dynamic and non-deterministic depending on how and when nodes connect and disconnect from the network. The logical topology totally differs from the physical topology, in fact two nodes in the same LAN can be non-connected while a node running in a cellular phone can be connected to another situated in the other side of the planet.

The set of nodes connected to a given node N is the set of **neighbour** nodes to N.

### Node

A node is an object able to produce new messages, and to route or consume incoming messages. Any node has a unique nodeID which should be used by any other to establish a new connection with the node. An example of nodeID is the string ipAddress:port in the case of connections implemented as TCP/IP connections.

A node which receives an incoming REQUEST message from one of its connections tries to produce and reply a RESPONSE. **It is up to the specific application (e.g. search) to decide whether or not propagate the message to the other neighbours. The algorithm that selects the nodes to forward is beyond the scope of this specification.** Here we define all the operations that must be performed on incoming messages of type REQUEST and RESPONSE.

- BROADCAST: a node produces a new  $R = REQUEST(F, P)$  and sends it to all adjacent nodes. Eventually, it will receive one or more messages  $RESPONSE(R, F, Q)$  as responses for the original request. In the last case, it consumes the response for itself with no further routing to someone else. The BROADCAST may be filtered to avoid network flooding: **It is up to the specific application (e.g. search) to decide whether or not propagate the message to the other neighbours. The algorithm that selects the nodes to forward is beyond the scope of this specification**
- FORWARD: a node which receives an incoming  $R = REQUEST(F, P)$  from a neighbour node along a connection C, must forward the REQUEST to all other adjacent nodes. The node must store an entry of type (requestID, C), the set of such entries is the response routing table of the node. The FORWARD operation may be filtered to avoid network flooding: **It is up to the specific application (e.g. search) to decide whether or not propagate the message to all the other**

**neighbours. The algorithm that selects the nodes to forward is beyond the scope of this specification**

- **BACKWARD:** a node which receives an incoming RESPONSE(R,F,P) from an adjacent node must verify whether the RESPONSE is to be consumed by the local peer or must be routed backward to another requestor node. In the latter case, the node looks in the response-routing table for an entry of type (RID,C) where C is a connection and RID is the request ID of the original REQUEST R. The RESPONSE is then routed back along the connection C.
- **REPLY:** a node which receives an incoming R=REQUEST(F,P) from a connection C, must try to produce a corresponding RESPONSE(R,F,Q). If successful, the node sends back to C the response.

**Infinite Loop**

A message travelling along a loop in the network topology must not be forwarded forever. In some P2P protocols messages include a TTL field which counts the number of hops and the message is discarded when the TTL is equal to the maximum allowed. In our architecture we adopt a different strategy: a node never processes twice the same message. In other words, when a node produces or receives a message it stores the ID in a list, and if the same ID is received later the message is merely ignored. This is a space-consuming technique which has the advantage to never allow messages to enter in small loops (the TTL technique could allow to traverse the same nodes more than once). The amount of memory consumed is acceptable. For instance, if you consider an ID of 8 bytes length and an incoming message rate of 100 per second, with 160KB of memory it is possible to store a LRU cache of 20,000 message IDs which is equivalent to the limit of 200sec of time for the message to traverse all the network in a roundtrip.

**PROTOCOL SPECS**

Regarding the formalism describing the protocol specification we adopt the notation:

expression:  
    subexp-1  
    subexp-2

to denote that expression can assume one between the  
subexpression: subexp-1 e subexp-2.

The notation:  
expression:  
    subexp-1 subexp-2

denotes that expression is the concatenation of expressions  
subexp-1 e subexp-2.

Every expression is composed by tokens belonging to the following types:

- byte, 1 byte, from -128 to 127, inclusive
- short, 2 byte big-endian from -32768 to 32767, inclusive
- int, 4 byte big-endian from -2147483648 to 2147483647, inclusive
- long, 8 byte big-endian from -9223372036854775808 to 9223372036854775807, inclusive
- UTF-8, a string of characters with 8bit UTF encoding.

The notation `expr [N]` denotes a sequence of expressions of type `expr`, the expression `()` denotes the empty expression (a token with 0-byte).

The notation

//comment

denotes that the expression 'comment' is a one line comment.

The notation

/\*

Multi line

Comment

\*/

denotes a comment which is placed in multiple lines

### **Message Format**

message:

messageHeader messagePayload

messageHeader:

ID reqID funID sesID type length

ID:

long

reqID:

long

funID:

long

sesID://unused, here for future applications

long

type:

0 //GENERIC;

1 //REQUEST;

2 //RESPONSE;

where 0,1,2 are literals of type short.

length:

long

messagePayload:

byte[length]

### **Currently Defined FunctionIDs**

The following long values are the ones currently under use in AXMEDIS VDB. The list could increase according to future development of the system.

long WHO	=0x00000000;
long ECHO	=0x00000002;
long TEST	=0x00000008;
long AX_QUERY	=0xFFFF0000;
long AX_INDEX	=0xFFFF0001;
long AX_UNPUBLISH	=0xFFFF0002;
long AX_RENEWLEASE	=0xFFFF0003;

### **WHO**

A node can send a WHO request in the network to know which are the other participants. A WHO request should be replied by each node and propagated to the neighbors without filters. It is intended as a exceptional request and must be sent only occasionally because it provokes a message flood in the network.

A leaf node may produce and reply a WHO request.

A hub node may produce, must reply, must propagate to hubs and may propagate to leaves a WHO request.

```
R=REQUEST (WHO, payload)
```

```
payload:  
  ()
```

```
RESPONSE (R, WHO, nodeID)
```

```
nodeID:  
  UTF-8
```

### **ECHO**

This service is intended to be used only for testing a connection. A node may send a ECHO request containing a keyword in the payload and must receive as response an ECHO response with the same payload.

A leaf node may produce, must reply, must not propagate a ECHO request.

A hub node may produce, must reply, must not propagate a ECHO request.

```
R=REQUEST (ECHO, keyword)
```

```
keyword:  
  UTF-8
```

```
RESPONSE (R, SEARCH, keyword)
```

```
keyword:  
  UTF-8
```

### **TEST**

This service is intended to be used only for unit tests and acceptance tests.

```
R=REQUEST (TEST, ( ))
```

```
RESPONSE (R, TEST, ( ))
```

### **AX QUERY**

This service is intended to be used to produce, propagate and reply to AXMEDIS queries. A node may send a AX\_QUERY request containing a full AXMEDIS compliant XML query in the payload and may receive one or more AX\_QUERY responses with AXMEDIS query results in the payload.

A leaf node may produce, must not reply, must not propagate a AX\_QUERY request.

A hub node may produce, may reply, may propagate an AX\_QUERY request to other hubs, must not propagate AX\_QUERY requests to leaves.

```
R=REQUEST (AX_QUERY, AXMEDIS_QUERY)
```

```
AXMEDIS_QUERY:  
  UTF-8
```

```
//The AXMEDIS_QUERY is a string containing an XML instance specified in the
```

```
//DE3-1-2 part E

RESPONSE(R,AX_QUERY,AXMEDIS_QUERY_RESULTS)

AXMEDIS_QUERY_RESULTS:
    UTF-8
//The AXMEDIS_QUERY_RESULTS is a string containing an XML instance
//specified in the DE3-1-2 part E
```

### **AX INDEX**

This service is intended to be used by leaves to index one or more AXMEDIS objects in a remote AXDBOUT database inside a remote hub. Since the remote hub consume resources to index leaves' objects it replies with a unique lease id and a expire time for the lease. If a leaf node wants to renew the lease it must send a AX\_RENEWLEASE request otherwise its resources will be released and objects will be no longer available for searches.

A leaf node may produce, must not reply, must not propagate a AX\_INDEX request.  
A hub node may receive, may reply, must not propagate an AX\_QUERY.

```
R=REQUEST(AX_INDEX,objectArray)

objectArray:
    objectArrayLength metadata[objectArrayLength]

objectArrayLength:
    int

metadata:
    metadataLength couple[metadataLength]

metadataLength:
    int

couple:
    key value

key:
    "AXOID"
    "URL" //where the object is located or localizable for download
    AXINFO_KEY
    DublinCore_KEY
    PAR_KEY

/*
The possible 'AXINFO_KEY' are those expected as MANDATORY fields in the AXMEDIS
metadata specification and available in DE3-1-2 part A.
The possible 'PAR_KEY' are those expected as MANDATORY fields in the AXMEDIS
metadata specification and available in DE3-1-2 part A.
The possible DublinCore_KEY are those defined in the Dublin Core Specification
[ref]
*/

value:
    UTF-8
    Datetime

Datetime:
```

#### DE4.4.1 – Content Sharing and Production on P2P

```
UTF-8 //formatted as YYYYMMDDHHmmSS

/*If an hub is not able to fulfill the request then it must ignore it. The
requestor leaf must manage the absence of a reply by means of a timeout*/

RESPONSE(R,AX_ INDEX,AX_INDEX_RESULTS)

AXMEDIS_INDEX_RESULTS:
    UNIQUE_LEASE_ID EXPIRATION_DATE

UNIQUE_LEASE_ID:
    long

EXPIRATION_DATE:
    Datetime

Datetime:
    UTF-8 //formatted as YYYYMMDDHHmmSS
```

#### **AX UNPUBLISH**

This service is intended to be used by leaves to unpublish one AXMEDIS object already indexed in a remote AXDBOUT database inside a remote hub.

A leaf node may produce, must not reply, must not propagate a AX\_UNPUBLISH request.  
A hub node may receive, must reply, must not propagate an AX\_UNPUBLISH request.

```
R=REQUEST(AX_UNPUBLISH,AXOID)

AXOID:
    UTF-8
//AXOID is a 40 chars UTF-8 string specified in the DE3-1-2 part E

RESPONSE(R,AX_UNPUBLISH,())

/*
This response has not a payload because the requestor can infer from the
requestId which is the request the generates this response and thus the set of
objects that are subject to UNPUBLISH.
*/
```

#### **AX RENEWLEASE**

This service is intended to be used by leaves to renew the allocation of indexes for one or more AXMEDIS objects already indexed in a remote AXDBOUT database inside a remote hub.

A leaf node may produce, must not reply, must not propagate a AX\_RENEWLEASE request.  
A hub node may receive, must reply, may propagate an AX\_RENEWLEASE request.

```
R=REQUEST(AX_RENEWLEASE,UNIQUE_LEASE_ID)

UNIQUE_LEASE_ID:
    long

RESPONSE(R,AX_ RENEWLEASE,TIME_TO_EXPIRE)

TIME_TO_EXPIRE:
    Datetime
```

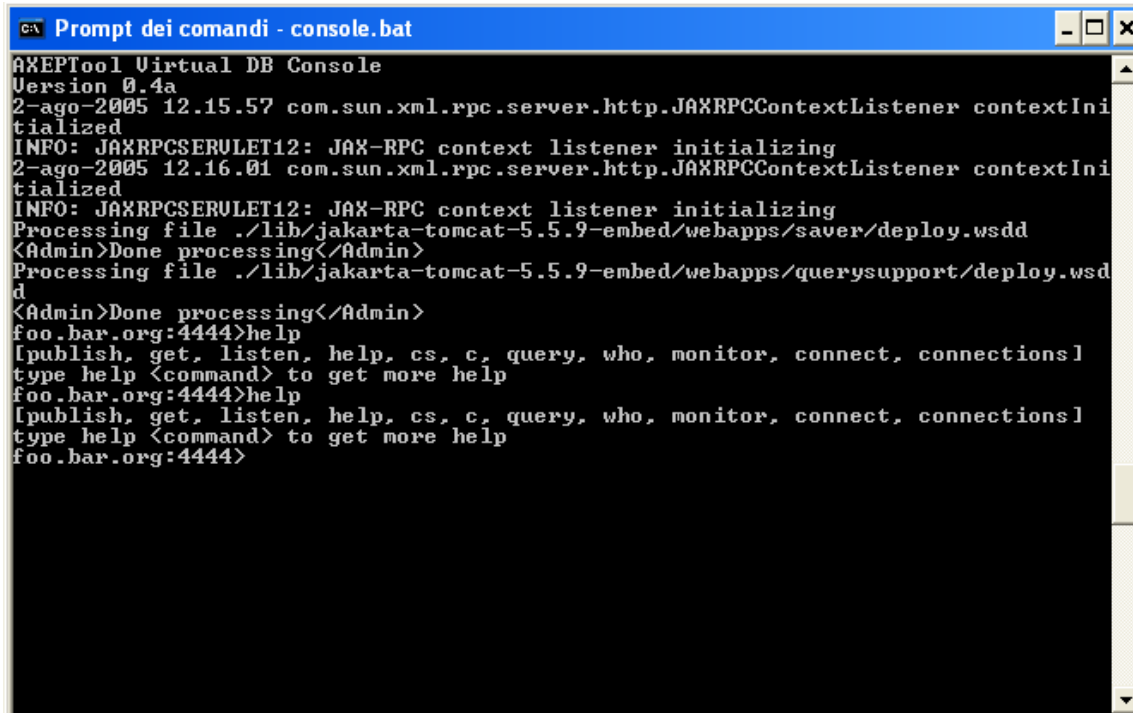


Datetime:  
UTF-8 //formatted as YYYYMMDDHHmmSS

## 12.2 AXEPTool Console

This document describes the basic functionalities of the AXEPTool Command Console (from now on "the Console").

The Console is a program written in Java which can be used to instantiate a p2p node and to invoke functionalities in interactive way. It is mainly a TESTING tool and will be used as early prototype.



```
Prompt dei comandi - console.bat
AXEPTool Virtual DB Console
Version 0.4a
2-ago-2005 12.15.57 com.sun.xml.rpc.server.http.JAXRPCContextListener contextIni
tialized
INFO: JAXRPCSEULET12: JAX-RPC context listener initializing
2-ago-2005 12.16.01 com.sun.xml.rpc.server.http.JAXRPCContextListener contextIni
tialized
INFO: JAXRPCSEULET12: JAX-RPC context listener initializing
Processing file ./lib/jakarta-tomcat-5.5.9-embed/webapps/saver/deploy.wsdd
<Admin>Done processing</Admin>
Processing file ./lib/jakarta-tomcat-5.5.9-embed/webapps/querysupport/deploy.wsd
d
<Admin>Done processing</Admin>
foo.bar.org:4444>help
[publish, get, listen, help, cs, c, query, who, monitor, connect, connections]
type help <command> to get more help
foo.bar.org:4444>help
[publish, get, listen, help, cs, c, query, who, monitor, connect, connections]
type help <command> to get more help
foo.bar.org:4444>
```

### 12.2.1.1 COMPILING THE CONSOLE

Use ANT and type:

```
>ant clean
```

```
>ant
```

```
>ant dist
```

a complete console distribution with libraries and resources will be available under the bin directory.

### 12.2.1.2 RUNNING THE CONSOLE

To run an AXEPTool node in a console move to the bin/ directory of the application and run the script:

```
>console.sh
```

in Linux or

>console.bat

in Windows.

You'll be prompted with:

<peerID>>

Type "help" for a list of commands installed and type "help <command>" for detailed help.

To change the peerID modify the bin/config.properties file.

### **12.2.1.3 HELP COMMAND**

NAME

help - list the available commands

SYNOPSIS

help [command]

DESCRIPTION

if used without command it prints the list the available commands. Otherwise it prints the usage of a specific command.

### **12.2.1.4 WHO COMMAND**

NAME

who - lists participants in the network

SYNOPSIS

who

DESCRIPTION

sends a WHO request to the P2P network, it should receives the IDs of other peers. You must be connected to at least one other peer

### **12.2.1.5 LISTEN COMMAND**

NAME

listen - accept incoming TCP/IP connections

SYNOPSIS

listen <port>

DESCRIPTION

starts listening for incoming TCP/IP connections.  
Only hubs provide this service. If the node is already listening to that port a java exception is thrown to the console.

### **12.2.1.6 CONNECT COMMAND**

NAME

connect - open a connection to a remote hub

SYNOPSIS

connect <hostname:port>

DESCRIPTION

connect to a remote hub and starts the handshaking. If the connection is well established then messages can be exchanged along the connection. According to the config.properties settings the connection may occur in plain/text or under SSL/TSL protocols.

**12.2.1.7 CONNECTIONS COMMAND**

NAME

connections - list peers currently connected to this

SYNOPSIS

connections

DESCRIPTION

list peers currently connected to this

**12.2.1.8 GET COMMAND**

NAME

get - request a remote file

SYNOPSIS

get [option] <http-uri>

DESCRIPTION

starts a download session for object in <http-uri>. Sessions can be managed via monitor command.

-bt, --bittorrent

uses BitTorrent if available as client to get a content. In this case the <http-uri> must point to the .torrent file. This feature is not implemented yet.

**12.2.1.9 MONITOR COMMAND**

NAME

monitor - manages download and upload sessions

SYNOPSIS

monitor [option sessionID]

DESCRIPTION

shows the state of download sessions and uploads (upload state not implemented yet).

-s, --suspend

suspends a session (not implemented)

-r, --resume

resumes a previous suspended session (not implemented)

-a, --abort  
cancels a session (not implemented)

#### **12.2.1.10 PUBLISH COMMAND**

NAME

publish - publishes local object in the network

SYNOPSIS

publish <object-uri>

DESCRIPTION

takes an local object for publication in the local AXDBOUT.  
A RSS feed is produced and the object becomes available for download.

#### **12.2.1.11 QUERY COMMAND**

NAME

query - produces and sends a query

SYNOPSIS

query <filename>

DESCRIPTION

Opens a file containing an AXMEDIS query and sends it to the network.  
Results are received in the console in form of XML.

#### **12.2.1.12 INDEX COMMAND**

NAME

index - index a local object in a remote hub (command not implemented)

SYNOPSIS

index <object-uri>

DESCRIPTION

Takes an local object for publication in a remote hub DBOUT.  
A RSS feed is produced and the object becomes available for download.