Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE3.1.2.3.10
Specification of AXMEDIS P2P tools AXEPTool and AXMEDIA Tools update of DE3.1.2.2.10

Version: 3.5 (public version)
Date: 18-07-2007
Responsible: DSI, D. Cenni (revised and accepted by coordinator)

Project Number: IST-2-511299 Project Title: AXMEDIS Deliverable Type: report Visible to User Groups: yes Visible to Affiliated: yes Visible to the Public: yes
Deliverable Number: DE3.1.2.3.10 Contractual Date of Delivery: M34 Actual Date of Delivery: 18/07/2007 Title of Deliverable: Document Work-Package contributing to the Deliverable: WP3.1 Task contributing to the Deliverable: WP3, WP2 Nature of the Deliverable: report Author(s): DSI, EXITECH, HEXAGLOBE

Abstract: this part includes the specification of components, formats, databases and protocol related to the AXMEDIS Framework area of P2P including AXMEDIS P2P tools, AXEPTool, AXMEDIA, AXMEDIS P2P Query Service Servers, etc.

Keyword List: AXMEDIS P2P network, AXEPTool, AXMEDIA, bittorrent

AXMEDIS Copyright Notice

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

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

1. DEFINITIONS

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

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

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

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

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

2. **LICENCE**

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

3. **TERM AND TERMINATION**

1. Granted Licence shall commence on Acceptance Date.
2. Granted Licence will terminate automatically if licensee fails to comply with any of the terms and conditions of this Licence.
3. Termination of this Licence does not affect either party's accrued rights and obligations as at the date of termination.
4. Upon termination of this Licence for whatever reason, licensee shall cease to make any use of the accessed Copyright.
5. All provisions of this Licence, which are necessary for the interpretation or enforcement of a party's rights or obligations, shall survive termination of this Licence and shall continue in full force and effect.
6. Notwithstanding License termination, confidentiality clauses related to any content, document or part of it as stated in the document itself will remain in force for a period of 5 years after license issue date or the period stated in the document whichever is the longer.

4. **USE**

1. Licensee shall not breach or denigrate the integrity of the Copyright Notice and in particular shall not:
   i. remove this Copyright Notice on a Document or any of its reproduction in any form in which those may be achieved;
   ii. change or remove the title of a Document;
   iii. use all or any part of a Document as part of a specification or standard not emanating from the Licensor without the prior written consent of the Licensor; or
   iv. do or permit others to do any act or omission in relation to a Document which is contrary to the rights and obligations as stated in the present license and agreed with the Licensor

1. **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.

2. **WARRANTY**

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

3. **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.

4. **GOVERNING LAW AND JURISDICTION**

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

Please note that:

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

Table of Content

1 Executive Summary and Report Scope

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

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

1.1 This document concerns

Specification of AXMEDIS P2P tools, AXEPTool and AXMEDIS, first update of part of DE3.1.2

1.2 List of Modules or Executable Tools Specified in this document

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

Module/tool Name	Module/Tool Description and purpose, state also in which other AXMEDIS area is used	Standards exploited if any
AXMEDIS P2P Query Service	A system server with a set of features to provide suppot for quering into the the P2P network	SQL, WSDL
AXMEDIS Query Service WEB Portal	An interface for business and final users to make queries on the P2P network	SQL
AXMEDIS Query Service Web Service	A WEB Service to receive queries from remote other tools to pose them on the P2P network.	SQL, WSDL
Publishing Objects Interface	A Web Service interface of the AXEPTool to allow publishing AXMEDIS objects into the P2P network	
Monitoring Web Service	A Web Service interface of the AXEPTool to allow downaloding and monitoring AXMEDIS objects from the P2P network	
query (results) interface on P2P tools	Query interface integrated into the AXEPTool and AXMEDIA tools to make queries to the AXMEDIS P2P Query Service via the AXMEDIS Query Service Web Service	
AXEPTool	P2P client peer for B2B file Sharing	BitTorrent protocol, AXMEDIS
AXMEDIA	P2P client peer for C2C file sharing	BitTorrent protocol, AXMEDIS

1.3 List of Formats Specified in this document

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

Format Name	Format Description and purpose, state also in which other modules is used	Standards exploited if any

1.4 List of Databases Specified in this document

Database Name	database Description and purpose, state also in which other AXMEDIS area is using	Standards exploited if any

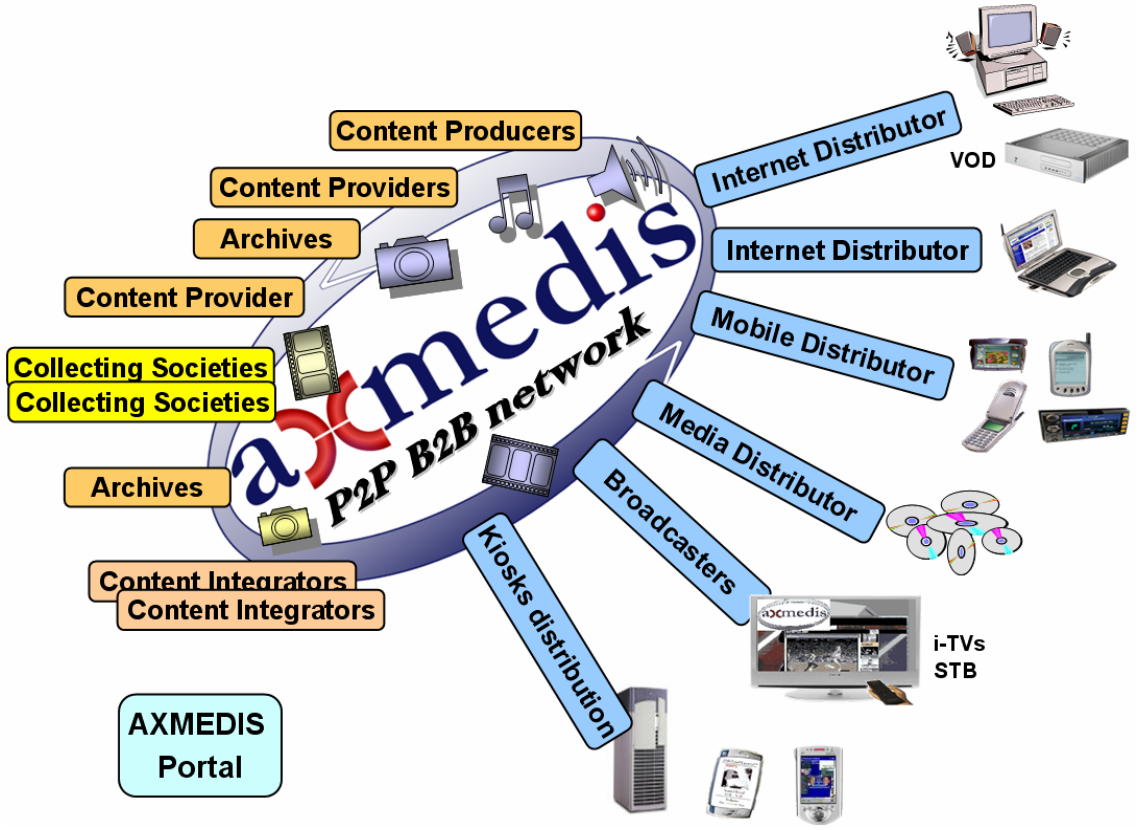
1.5 List of Protocols Specified in this document

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

Protocol Name	protocol Description and purpose, state also in which other modules is used	Who is the master and who is the slave	Standards exploited if any
Publication of Objects	A way to publish AXMEDIS objects on the B2B P2P network automatically	Master : AXCP Client AXEPTool	WSDL
Download of Objects	A way to download AXMEDIS objects from the B2B P2P network automatically, and monitoring the download	Master : AXCP Client : AXEPTool	WSDL

2 Aims of P2P in AXMEDIS

AXMEDIS framework and general solution supports DRM and Reporting about the usage of the content features, e.g., play, print, extract, etc., which are the rights These tools gather the information related to the exploitation of rights along the value chain and by the final user and reports it back to the concerned actors. These utilities are very useful to provide the evidence of the exploited rights in a transparent manner to collecting societies or other business partners.



AXMEDIS Business to Business area with some distributors

In order to ease the collaboration among the business area, AXMEDIS is providing a wide set of tools. Among these tools some of them are based on P2P concepts for content distribution, and are:

- AXEPTool for B2B content distribution
- AXMEDIA for C2C content distribution at support of B2C content distribution.

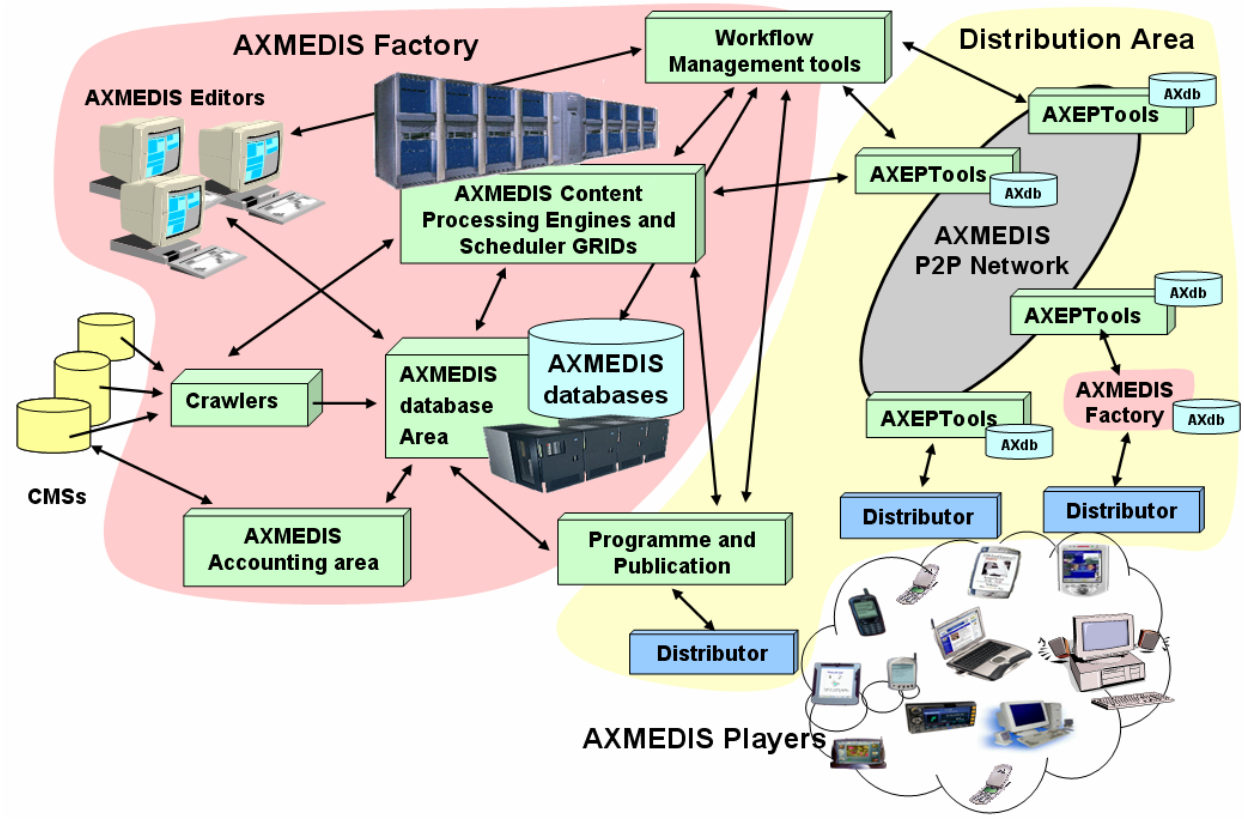
The B2B P2P Network depicted in the figure is used for sharing content among actors of the value chain by using the AXEPTool. One of more rings or area of P2P B2B may would be created to share different kind of objects or for geographical purpose.

The distribution side may present one or more single distribution paths for each type of content. In AXMEDIS, the content distributors can continue their preferred mechanisms for reaching the final users. The possible Channel Distributors have a large variety of capabilities, they are both of pull and push, and may include off-line and on-line connection from the client to the distributor.

Some Distributors may be interested in exploiting AXMEDIS technology to set up a legal P2P service for content distribution. This solution can be realized by using the so called **AXMEDIA** tools. It is a P2P tool for distributing and sharing AXMEDIS content among end users, the distributor may insert AXMEDIS content in the network of peers and this may freely navigate among them but with the supervision and control of AXMEDIS protection and monitoring models.

2.1 AXMEDIS Content Production

The Content Production area of AXMEDIS is mainly focused on what it is called AXMEDIS Factory automating: (i) the packaging containing the digital resources (the real content), (ii) adapting and Transcoding, (iii) protecting content and producing corresponding Prot-Info, (iv) publishing and distributing the produced package, (v) producing licenses for the users, etc.



Content Object Production

- Existing contents (e.g., resources and metadata located in databases, CMSs, file systems) are crawled and collected using automated processes and rules (e.g., Crawler, rule editing);
- Crawled contents or new contents can be inserted into the AXMEDIS Database;
- Content can be automatically packaged in AXMEDIS model (which is MPEG-21 compliant);
- Content production and elements (metadata, resources, protection information and licenses) can be processed manually with authoring tools and editor or automatically with AXMEDIS Content Processing tools based on executable rules that can be hosted on single or massive GRID of computers;
- Authoring tools can be used to insert/revise metadata, define protection information, define DRM licenses models, to modify content or simply to view the objects;
- Various processing are offered (adaptation, transcoding, protection, etc.), either automatically or manually (using GUI editor);
- The various components and digital resources can be glued together by means of SMIL based templates and style that may be used to define the usage interface (format) of the whole object: karaoke, collection, menus, sliding presentation, buttons, live, animations, etc.;
- Results can be sorted in a database or on file systems or published in towards distributors or on the B2B P2P AXMEDIS network;
- They are now available on the AXMEDIS network, for further aggregation, distribution, etc., to be searched, modified, or shared;

1 Queries and P2P allow retrieving content located in all the connected AXMEDIS Factories;
2 The queries can be activated to automatically react at eventual changes in the sources, and thus to perform an automatic production/update;
3 Queries can be performed on the basis of classification and identification metadata, but also on technical features, descriptors, licensing information (PAR), etc.;

4 Once identified the objects or queries, they can be used as input parameter of processing rules for the GRID AXMEDIS Content Processing;
5 All these activities can be governed by Workflow Management Tools for defining process production flow and information of the content factory and among different factories.

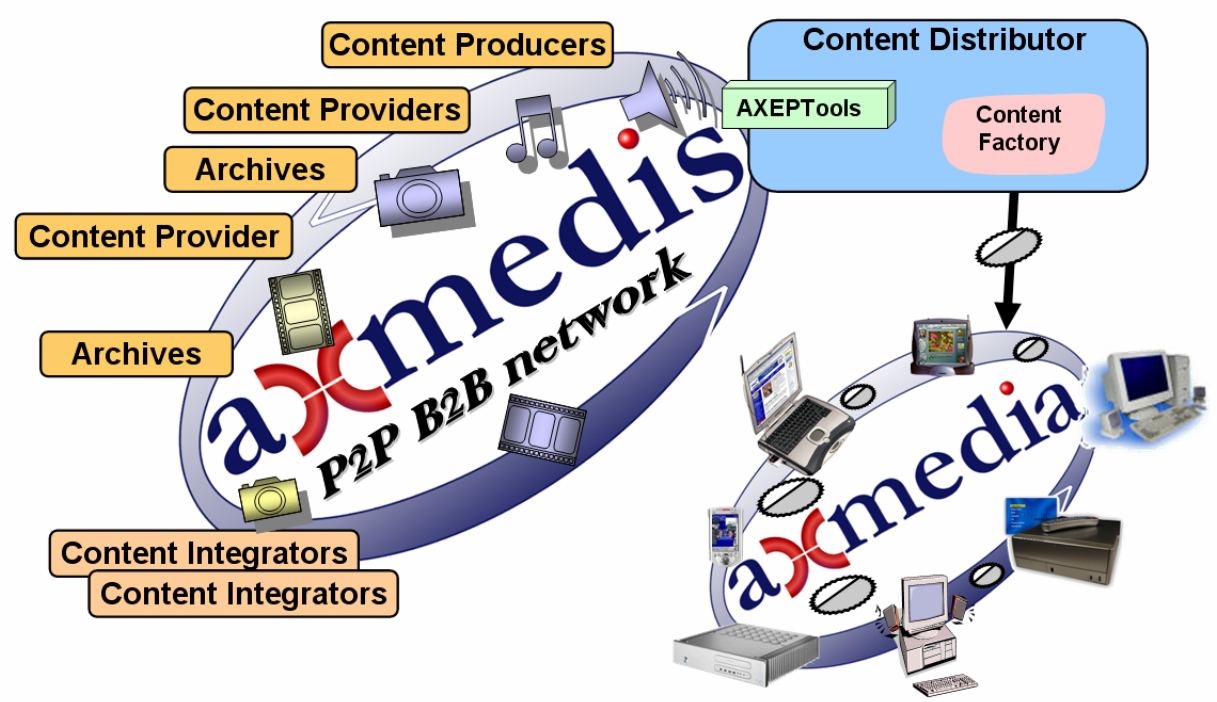
2.2 AXMEDIS P2P Tools: AXEPTool and AXMEDIA tool

In AXMEDIS, the P2P technology is used for sharing AXMEDIS Content among:

- B2B partners such as content: producers, integrators, distributors, publishers, archives, etc.; realized with AXEPTool;
- final users of a content distributor, realized with a Peer tool called AXMEDIA tool.

The **general features of any AXMEDIS P2P tools (AXEPTool and AXMEDIA)** are:

1 legal usage of P2P content distribution;
2 guarantee of consistency between the exposed metadata and the downloaded content;
3 verification and certification of content distribution and sharing;
4 monitoring of downloads and uploads in the P2P network;
5 efficiency of downloading, even for large files.



AXMEDIA usage and relationships with AXEPTool

From the point of view of B2B, the major features are:

- integration of P2P channel with the Content Factory;
  - automating the download and publishing of content;
  - separation of content to be shared from that maintained in the AXMEDIS database of the content factory;
- monitoring the content downloads and usages of published content;
- reduction of costs for content distribution and sharing:
  - automatic update of published content in the P2P network;
  - automatic download of updated content in other peers of the P2P network;
  - eventual usage of the C2C network for distributing content even among B2B users
- reduction of costs for content promotion;
  - allowing the sharing of promotional content

- reduction of costs for content search and acquisition:
    - possibility of making queries on the all connected content factories and these queries also include: classification, identification, technical descriptors of content and licensing information;
    - access to content coming from all the other connected business partners endowed by an AXEPTools.

Some Distributors are interested in exploiting AXMEDIS technology to set up a legal P2P service for content distribution among their users. This solution has to be realized by using the **AXMEDIA** tool as a Peer tool of a P2P network. It is a P2P tool for distributing and sharing AXMEDIS content among end users, the distributor may insert AXMEDIS content in the network of peers and that content may freely navigate among the peers with the supervision and control of AXMEDIS protection and monitoring model.

From the point of view of C2C, the major features are:

- easy to use
- easy to install

3 Requirements for P2P tools in AXMEDIS

The AXEPTool is the application that allows business users to share a common environment in which AXMEDIS Objects can 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.

In the following, only the most relevant and mandatory requirements have been reported. Please note that some of them area satisfied by means of the AXMEDIS Content Processing tools that actively and periodically can send queries to the P2P network and tools. These issues will be clearer observing the architecture of the AXEPTool into the AXMEDIS framework.

The AXEPTool for P2P on B2B must meet requirements divided into the following categories: system and IPR, general requirements, discovery and connection to the AXMEDIS community, query to search AXMEDIS objects, loading remote AXMEDIS objects, publishing local AXMEDIS objects.

Requirements that are not mandatory are commented in a specific manner.

3.1 System and IPR requirements for P2P tools of AXMEDIS

AXMEDIS consortium major system and IPR requirements about the P2P support and tools in AXMEDIS:

- ☐ to have of the P2P tools source code accessible and stored into the AXMEDIS Framework, located into the CVS of AXMEDIS;
- ☐ to have the possibility of manipulating the source code for the future without any restriction, and without the need to have any specific authorization;
- ☐ to have the possibility of improving the source code for integrating DRM aspects;
- ☐ to have the source code in open source license, so as to allow the usage of tools without extra costs (NOT MANDATORY).

The Consortium Agreement of AXMEDIS may clarify other aspects of IPR.

3.2 General P2P Requirements valid for both AXMEDIA and AXEPTool P2Ps

The P2P infrastructure of AXMEDIS:

- ☐ Must accept at least AXMEDIS objects in protected and non protected forms.
- ☐ May accept in addition to AXMEDIS objects any files (NOT MANDATORY): not only AXMEDIS objects can be ”shared” and queried but files of any type or content.
- ☐ Must allow performing some verification of consistency of the AXMEDIS object certification, to guarantee consistency of the objects with its metadata and its integrity, signature. Specific technology is present in the AXMEDIS that can be enforced into the P2P tools to cover this requirement.
- ☐ Must allow performing the estimation of some fingerprint on the AXMEDIS objects and/or on eventual single audio and video tracks. Specific technology is present in the AXMEDIS that can be enforced into the P2P tools to cover this requirement (NOT MANDATORY if the solution already have some hash estimation to solve this)..
- ☐ Must provide an efficient monitoring tool to present in the user interface the status of the download. The P2P tools must provide a real-time monitor for downloads of files in order to give to the consumer the status of all operations under run. The related GUI should present the monitored traffic as more as possible in an intuitively view.
- ☐ Must provide support for queries in according to the AXMEDIS Query interface
- ☐ Must support multiple sources download in an efficient manner such as .bitTorrent or other solutions
- ☐ Must support the creation of multiple sources downloads where the peers are both those signed to B2B and C2C P2P networks, dependently on the content they have only.

- The IDs used into the systems should be the AXOID defined in the AXMEDIS Framework with UUID model, if not possible an additional identification/descriptor has to be added into the AXMEDIS model.

3.3 Specific AXEPTool for P2P on B2B

The AXEPTool and solution has the above mentioned system and general requirements plus the following.

- Must provide user registration and certification of clients
- Must allow to establish some SSL connection with the tracker
- Must provide integration for automatic loading and publishing of AXMEDIS content and content in general. These features have to be exported as WEB Services
- Must provide support for sophisticated queries in according to the AXMEDIS Query interface *already developed and that can be reused into it.*
- Music provide fully complete results exposing AXMEDIS AXOID and metadata, PAR, etc. as in the spirit of AXMEDIS.

3.4 Specific AXMEDIA Tool for P2P on B2C

The AXMEDIA Tool is the P2P application used in B2C via P2P. More requirements about Client/Server Distribution via PC are available in the specification of WP4 related to "**AXMEDIS for Distribution via Internet**". The AXMEDIA tool and solution has the above mentioned system and general requirements plus the following.

- Must provide a simples query support that allows simple search queries composition through a simplified GUI, providing results in a simplified format, easy to understand for the final users.
- Music provide complete results exposing AXMEDIS AXOID and metadata.
- Must be easily installable on a wide range of computers and possibly on different platforms such as Windows and MAC.

4 General Information on BitTorrent Technology and related state of the art

Largely extracted from WEB pages and from Wikipedia: www.wikipedia.org***. On the other hand, even if the following information could be accessible on the WIKIPEDIA, due to the instability/evolution of that WEB site we preferred to include and instance of that information in the document.***

This chapter contains high level informations about Bittorrent. Protocol details will be given later in this documentation.

BitTorrent is the name of a peer-to-peer (P2P) file distribution client application and also of its related file sharing protocol, both of which were created by programmer Bram Cohen. BitTorrent is designed to distribute large amounts of data widely without incurring the corresponding consumption in costly server and bandwidth resources. CacheLogic suggests that BitTorrent traffic accounts for ~35% of all traffic on the Internet while other sources are skeptical.
The original BitTorrent application was written in Python. Its source code, as of version 4.0, has been released under the BitTorrent Open Source License, which is a modified version of the Jabber Open Source License. There are numerous compatible clients, written in a variety of programming languages, and running on a variety of computing platforms.
BitTorrent clients are programs which implement the BitTorrent protocol. Each BitTorrent client is capable of preparing, requesting, and transmitting any type of computer file over a network using the BitTorrent protocol. This includes text, audio, video, encrypted content, and other types of digital information.

Creating and publishing torrents
To share a file or group of files through BitTorrent, clients first create a ".torrent" information file. This is a small file which contains meta information about the files to be shared, and about the host computer that coordinates the file distribution (http://www.bittorrent.com). The exact information contained in the tracker file deper.news.yahoo.com/27122006/185/gastro-c-est-la-treve-des-confiseurs.htmlnds on the version of the BitTorrent protocol. However, a torrent file always has the extension .torrent. Torrent files contain an "announce" section, which specifies the URL of the tracker, and an "info" section which contains (suggested) names for the files, their lengths, the piece length used, and a SHA-1 hash code for each piece, which clients should use to verify the integrity of the data they receive.

Clients who have finished downloading the file may also choose to act as seeders, providing a complete copy of the file. After the torrent file is created, a link to it is placed on a website or elsewhere, and it is registered with a tracker. BitTorrent trackers maintain lists of the clients currently participating in the torrent. The computer with the initial copy of the file is referred to as the initial seeder.

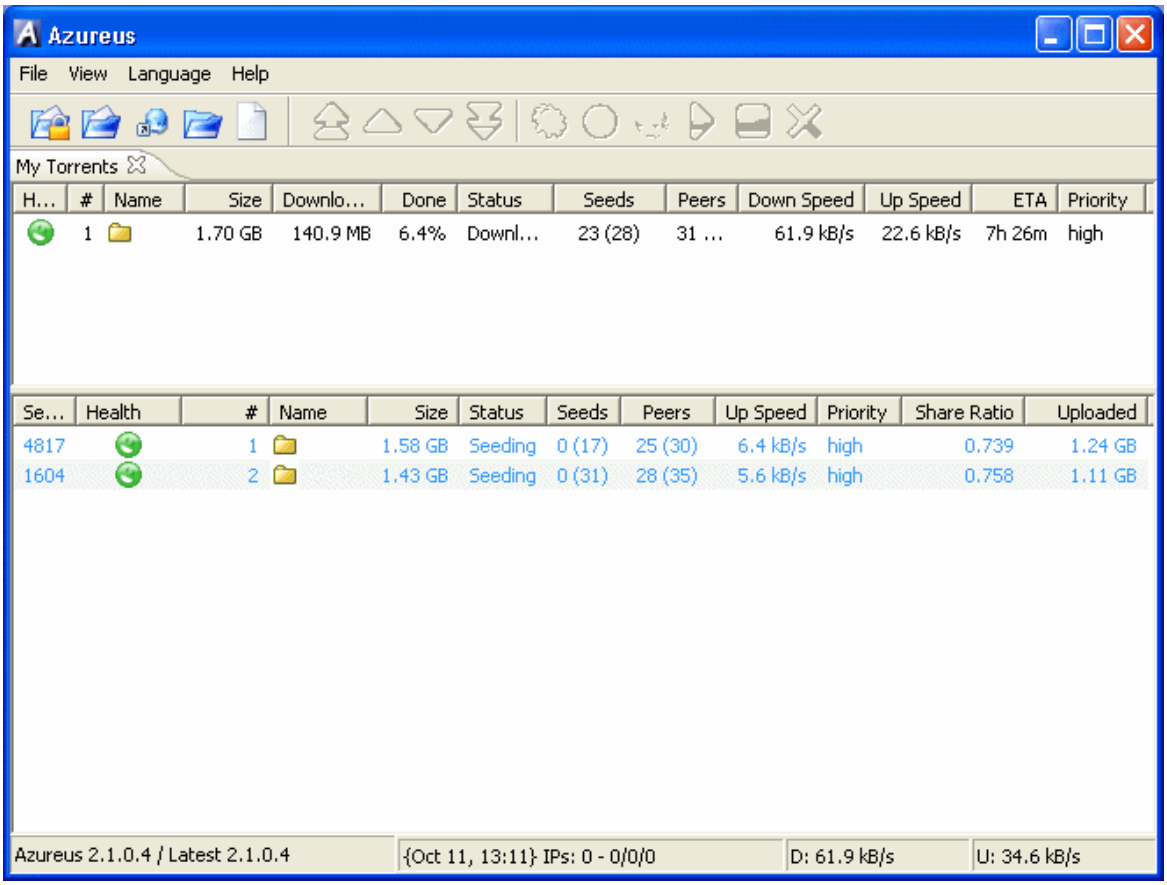
Downloading torrents and sharing files
Using a web browser, users navigate to the site listing the torrent, download it, and open it in a BitTorrent client. After opening the torrent, the BitTorrent client connects to the tracker, which provides it with a list of clients currently downloading the file or files. A group of peers on a BitTorrent or P2P connected with each other to share a particular torrent is generally referred to as a swarm.

Initially, there may be no other peers in the swarm, in which case the client connects directly to the initial seeder and begins to request pieces. The BitTorrent protocol breaks down files into a number of much smaller pieces, typically a quarter of a megabyte (256 KB) in size. Larger file sizes typically have larger pieces. For example, a 4.37 GB file may have a piece size of 4 MB (4096 KB). Pieces are checked as they are received using a hash algorithm to ensure that they are error free.

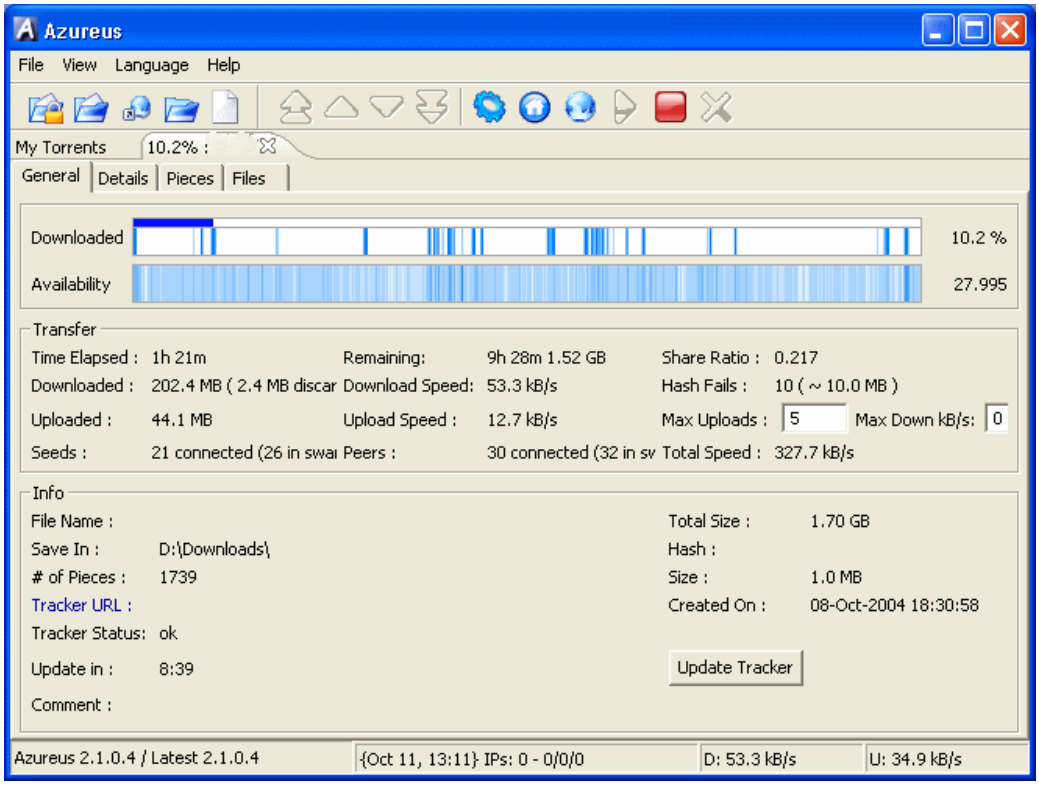
As peers enter the swarm, they begin sharing pieces with one another, instead of downloading directly from the seeder. Clients incorporate mechanisms to optimize their download and upload rates, for example using a

tit for tat scheme. Peers download pieces in a random order, to increase the opportunity to exchange data, which is only possible if two peers have a different subset of the file.

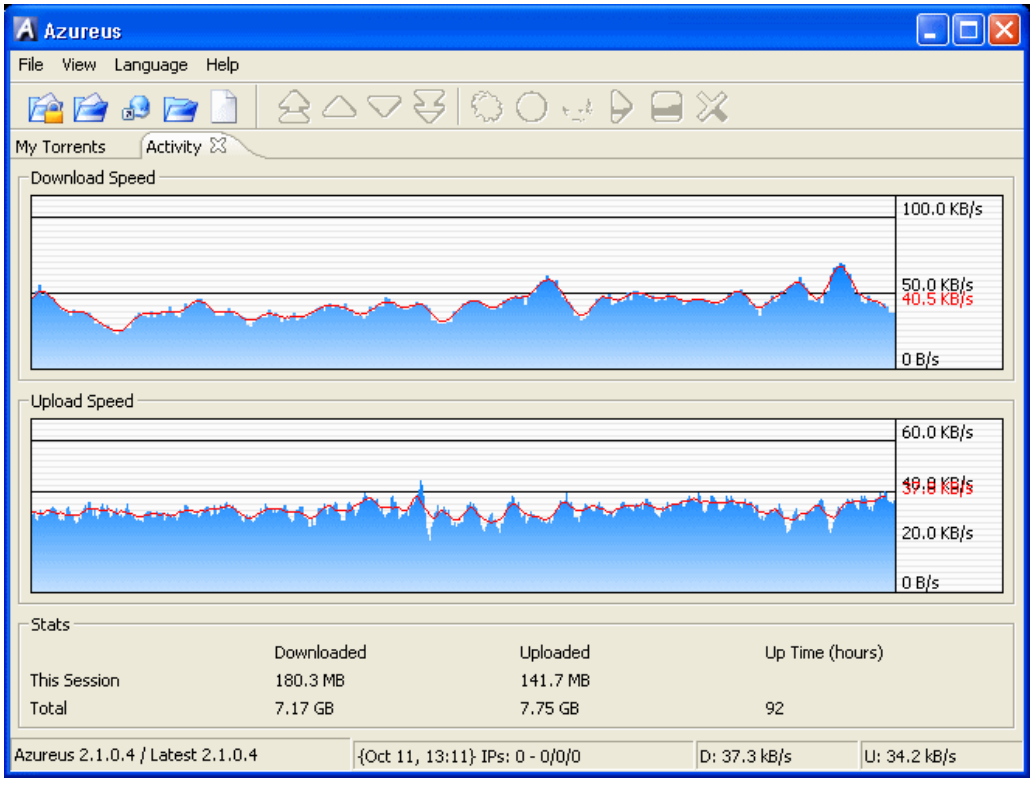
The effectiveness of the peer-to-peer data exchange depends largely on the policies used by clients to determine whom to send data to. Clients will prefer to send data to peers that send data back to them, which encourages fair sharing, but strict policies often result in suboptimal situations, where newly joined peers are unable to receive any data (because they don't have any pieces yet to share themselves) and two peers with a good connection between them do not exchange data simply because neither of them wants to take the initiative. To counter these effects, the official BitTorrent client uses a mechanism called “optimistic unchoking”, where the client will reserve a portion of its available bandwidth for sending pieces to random peers (not necessarily known-good partners, so called preferred peers), in hopes of discovering even better partners and to ensure newcomers get a chance to join the swarm.



In Azureus bittorrent client, while a file is downloading, right click and click **Show Details**. As you can see I've blocked out some information on mine as I don't want to go requesting permission from the tracker I was using before I post this guide. Therefore I have removed the **filename, Tracker URL, Hash** and the file **Comment**. On your screen you will see these details. Certain areas are important here like **Share Ratio**. As you can see my Share Ratio for this file is just 0.217. This is a bad ratio but expected while the file is still downloading. When the file finishes downloading, users are expected to keep the seeding going until the ratio goes over **1.000**. You will notice the **Update Tracker** button. Clients have to update the tracker on the parts of files they have and other such things, this button is just here in case you wish to update immediately, however if you look across the way, you will that there is already a countdown to next update. When you are seeding, there is a slightly different detail as you will see now.

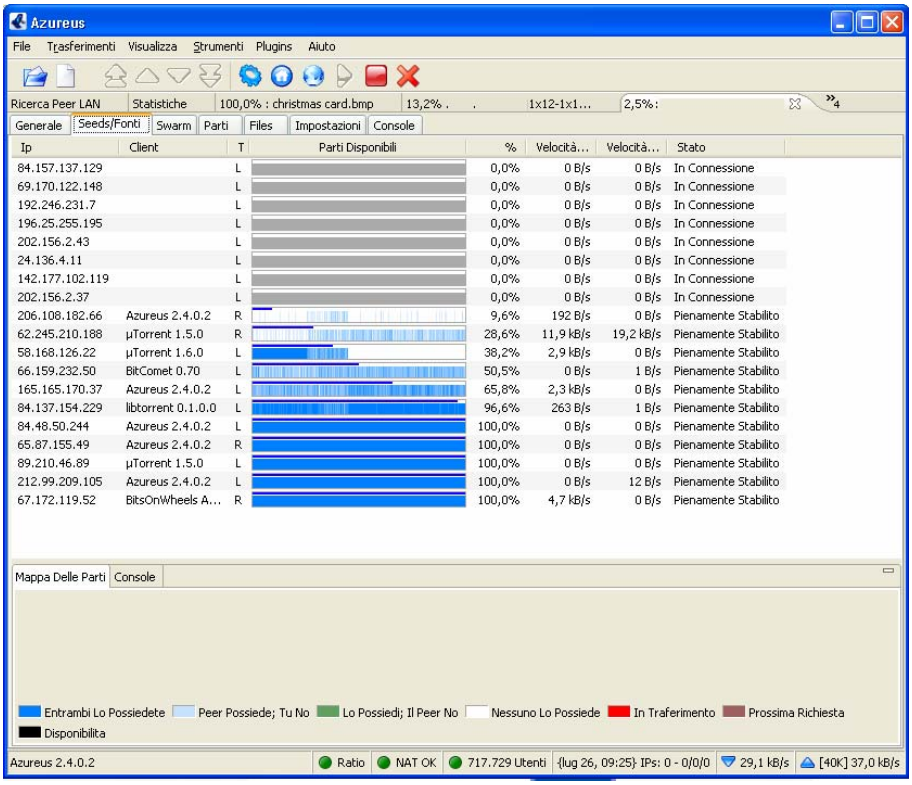


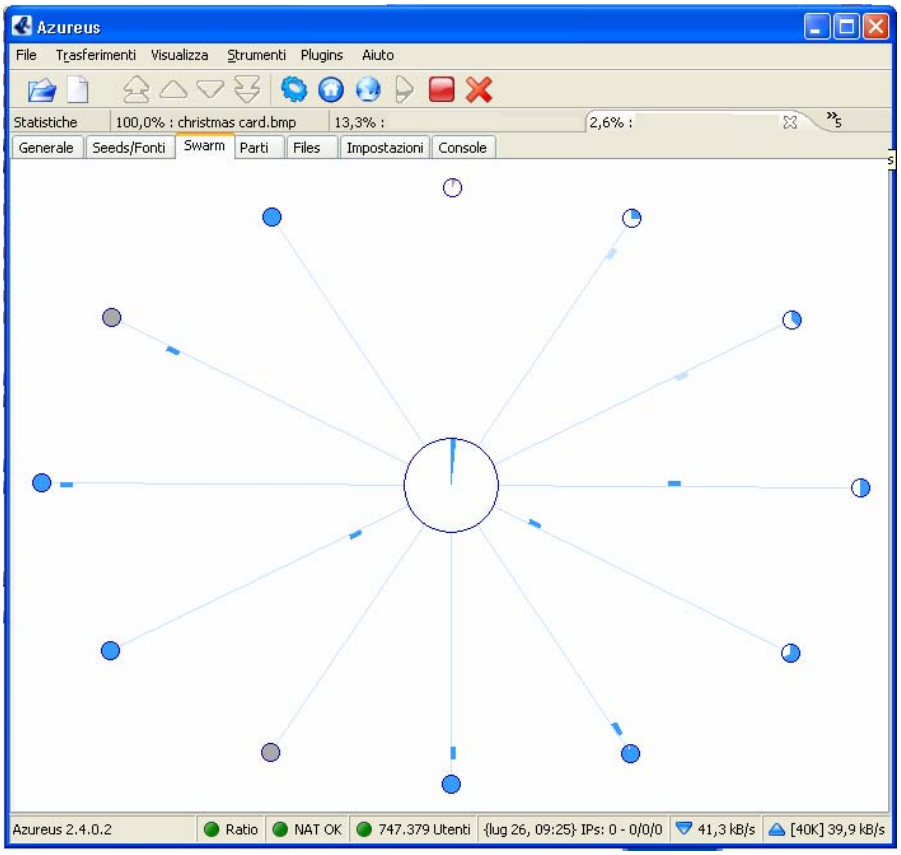
In Azureus bittorrent client, if you click **View --> Statistics**, you get some nice information on your upload and download bandwidth and how stable it has been. As you can see it also shows you your overall downloaded data since you began using Azureus, your uptime and the bandwidth transfers for this session.

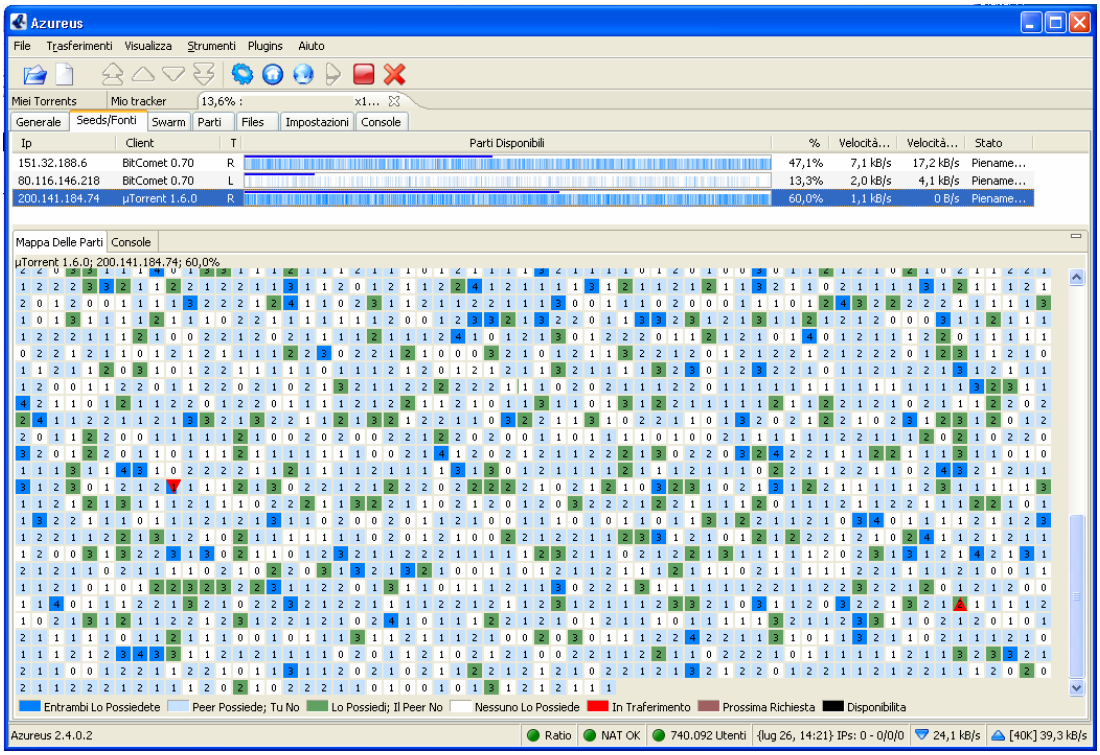


In the following, different representations of the status of the download are reported. The first one report the fonts with the rate of transfer for each of them. The second (the swarm) is a picture of the fonts with a simpler and immediate status of their capability. The latter is the status of the transfer for a given file in

which the status of each segment of the file is marked with a different color depending if the file segment is accessible, present, under download, etc.







The importance of Uploading

I had to write a small bit about uploading. Uploading on BitTorrent is vital. If you connect to a torrent that has just 3 seeds and 800 peers, then most of the sharing will be done between peers. If you download the whole file and have uploaded just 10% of that file and then leave, you are hurting the performance of that torrent. This kind of usage is very bad because if a lot of people begun doing it, then there would be very little seeds and eventually the seeds could disappear and there may be nobody left with 100% of the file. The full file still may be available as files are traded in small pieces, but if all users stopped uploading as much as they downloaded, torrents life wouldn't last long and when it was fully working, it would be very slow. Always make sure you upload as much as you download if not more. Someone who download 700MB and uploads 700MB still in the eyes of BitTorrent is not sharing properly. For the life of a torrent to go on for a long time at high speeds, all users should make sure they upload at least 150% of what they downloaded. When your torrents are done downloading keep them active until you reach this amount, or if you need to use the files, you can stop the torrent activity, use them (but do not alter them) and then click the same torrent again and you would be added to the tracker as a seed and would continue uploading again. Trackers has begun banning leechers, or keeping records of leechers as a way to deter their behavior, if you like BitTorrent, don't try and cheat it.

Limitations

BitTorrent does not offer its users anonymity. It is possible to obtain the IP addresses of all current, and possibly previous, participants in a swarm from the tracker. This may expose users with insecure systems to attacks to discover their identity. This is not a problem in a legal service for content sharing.

Another drawback is that BitTorrent file sharers compared to users of client/server technology often have little incentive to become seeders after they finish downloading. The result of this is that torrent swarms gradually die out, meaning a lower possibility of obtaining older torrents. Some BitTorrent websites have attempted to address this by recording each user's download and upload ratio for all or just the user to see, as well as the provision of access to older torrent files to people with better ratios.

BitTorrent is typically best suited in continuously connected broadband environments. Dial-up users find it less efficient due to frequent disconnects and slow download rates.

4.1 BitTorrent Terminology

- **Availability**: (also distributed copies) The number of full copies of the file available to the client. Each seed adds 1.0 to this number, as they have one complete copy of the file. A connected peer with a fraction of the file available adds that fraction to the availability, if no other peer has this part of the file. (ie. a peer with 65.3% of the file downloaded increases the availability by 0.653. However, if two peers both have the same portion of the file downloaded - say 50% - and there is only one seeder, the availability is 1.5).
- **Choked**: Describes a peer to whom the client refuses to send file pieces. A client chokes another client in several situations: The second client is a seed, in which case it does not want any pieces (ie. it is completely uninterested) The client is already uploading at its full capacity (ie. the value for max_uploads has been reached)
- **interested** Describes a downloader who wishes to obtain pieces of a file the client has. For example, the uploading client would flag a downloading client as 'interested' if that client did not possess a piece that it did, and wished to obtain it.
- **leech** A leech is usually a peer who has a negative effect on the swarm by having a very poor share ratio - in other words, downloading much more than they upload. Most leeches are users on asymmetric internet connections and do not leave their BitTorrent client open to seed the file after their download has completed. However, some leeches intentionally avoid uploading by using modified clients or excessively limiting their upload speed. The term leech, however, can be used simply to describe a peer - or any client that does not have 100% of the data.
- **peer** A peer is one instance of a BitTorrent client running on a computer on the Internet to which other clients connect and transfer data. Usually a peer does not have the complete file, but only parts of it. However, in the colloquial definition, "peer" can be used to refer to any participant in the swarm (in this case, it's synonymous with "client").
- **scrape** This is when a client sends a request to the tracking server for information about the statistics of the torrent, such as with whom to share the file and how well those other users are sharing.
- **seeder** A seeder is a peer that has a complete copy of the torrent and still offers it for upload. The more seeders there are, the better the chances are for completion of the file.
- **snubbed** An uploading client is flagged as snubbed if the downloading client has not received any data from it in over 60 seconds.
- **superseed** When a file is new, much time can be wasted because the seeding client might send the same file piece to many different peers, while other pieces have not yet been downloaded at all. Some clients, like ABC, Azureus, BitTornado, TorrentStorm, and µTorrent have a "superseed" mode, where they try to only send out pieces that have never been sent out before, making the initial propagation of the file much faster. This is generally used only for a new torrent, or one which must be re-seeded because no other seeds are available.
- **swarm** Together, all peers (including seeders) sharing a torrent are called a swarm. For example, six ordinary peers and two seeders make a swarm of eight.
- **torrent** A torrent can mean either a .torrent metadata file or all files described by it, depending on context. The torrent file contains metadata about all the files it makes downloadable, including their names and sizes and checksums of all pieces in the torrent. It also contains the address of a tracker that coordinates communication between the peers in the swarm.
- **tracker** A tracker is a server that keeps track of which seeds and peers are in the swarm. Clients report information to the tracker periodically and in exchange receive information about other clients to which they can connect. The tracker is not directly involved in the data transfer and does not have a copy of the file.

4.2 General Information on BitTorrent Tracker

A BitTorrent tracker is a server which assists in the communication between peers using the BitTorrent protocol. It is also, in the absence of extensions to the original protocol, the only major critical point, as clients are required to communicate with the tracker to initiate downloads. (Clients that have already begun downloading also communicate with the tracker periodically to negotiate with newer peers and provide

statistics, however, after the initial reception of peer data, peer communication can continue without a tracker.)

A tracker should be differentiated from a BitTorrent index by the fact that it does not necessarily list files that are being tracked. A BitTorrent index is a list of .torrent files (usually including descriptions and other information). Trackers merely coordinate communication between peers attempting to download the payload of the torrents.

Many BitTorrent websites act as both tracker and index. Sites such as these publicize the tracker's URL and allow users to upload torrents to the index with the tracker's URL embedded in them, providing all the features necessary to initiate a download.

Trackers are the primary reason for a damaged BitTorrent 'swarm'. (Other reasons are mostly related to damaged or hacked clients uploading corrupt data.) The reliability of trackers has been improved through two main innovations in the BitTorrent protocol:

Multi-tracker torrents feature multiple trackers in the one torrent. This way, should one tracker fail, the others can continue supporting file transfer.

There are two incompatible 'trackerless' BitTorrent transfer (aka. decentralized tracking) methods: DHT-based implementations, and Azureus's 'Distributed Database'.

The term 'trackerless' is something of a misnomer, as decentralized or distributed tracking essentially treats every peer in the swarm as a tracker. Original BitTorrent was the first client to offer decentralized tracking through its DHT method. Later, Azureus, µTorrent and BitComet adopted this feature, although Azureus's method of implementation is incompatible with the DHT offered by all other supporting clients.

4.2.1 Lists of Trackers

C/C++ Trackers

- ☐ BitCometTracker - C++(?), Windows Only
- ☐ Extended BitTorrent client and Tracker (XBTT) - Windows/Linux

BNBT Based

- ☐ BNBT - Port of the original Python BT tracker with many additional features
- ☐ BNBT Trinity Edition - A modified version of BNBT with a windows installer
- ☐ CBTT - Another tracker written in C++, based on BNBT
- ☐ =Xotic= Edition of BNBT - Based on BNBT with extra features; focused on Linux users
- ☐ BNBT - This is a C++ implementation of a BitTorrent tracker. It should compile under most any Unix with GCC available, as well as MS Windows with MSVC (binaries included.) It includes all of the functionality of the reference Python tracker, but it also includes many enhancements: user accounts, improved web interface, statistics, etc. See also the TrackPak for a bundled BNBT and installer that's easy to use.

PHP Trackers

- ☐ Blog Torrent - Supports webseed, does not require MySQL(?). ABANDONED
- ☐ Broadcast Machine - Successor to Blog Torrent, Focuses on Torrentcasts
- ☐ BT phpTracker Plus - Coded from scratch; lightweight
- ☐ Btittracker - A frontend for phpBTTracker
- ☐ ByteStats Tracker - External scrape support, installation script. ABANDONED
- ☐ PHP Nuke BitTorrent Module - BitTorrent tracker module for the popular PHPNuke CMS
- ☐ phpMyBitTorrent - Successor to the PHPNuke Tracker Module, this is a standalone tracker with many features
- ☐ BTChange 0.94a - For modifying tracker info in an existing .torrent file. Use this if the tracker changes, so that you don't have to recreate the file. See also: Sourceforge page.

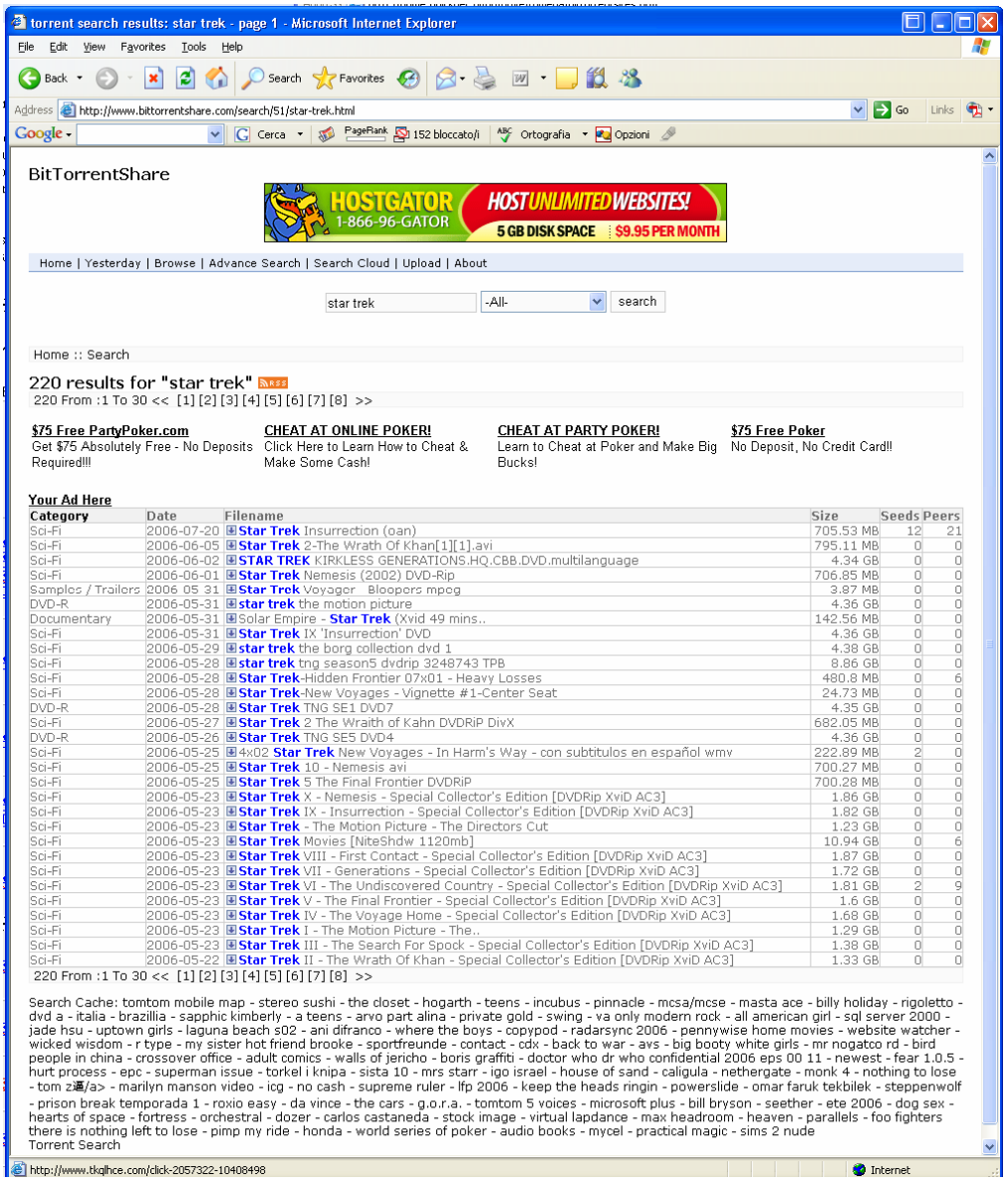
PHPBTTRACKER - Installer, MySQL. ABANDONED

- ☐ PHPBTTRACKER+ - Tracker based on PHPBTTRACKER. Many extra features
- ☐ PHPBTTracker-Admin - Another tracker based on PHPBTTRACKER, also with many additional features.
- ☐ PHPBTTracker - a free (GPL) tracker implementation in PHP with a MySQL back-end, includes built-in statistics collection and reporting.

4.3 Servers for listing and searching Torrent Files

They are servers that allow you to make some query. There are many sites that list torrent files. To use these sites, all you would have to do is click on a file listed, and it should download automatically and your client should immediately connect to the tracker and start the downloading. Some known sites that list torrent files are. Sites that list torrent files (like suprnova.org) have their trackers to handle the downloading and sharing that the users are doing. A torrent file would have the details on the tracker so you can only use a torrent file on the tracker it was made for. Software exists for anybody to setup their own trackers and build their own torrent files.

- http://home.quicknet.nl/qn/prive/romeria/bittorrentsites.htm list of BitTorrent trackers, servers with list of files, server in which it is possible to search for BitTorrent file.
- Suprnova.org - (Only works with original BT client)
- Torrentbits.org
- TorrentReactor.com
- FileList.org - (Requires registration and read FAQ)



4.4 Some additional issues of BitTorrent

4.4.1 Alternative approaches

The BitTorrent protocol provides no way to index torrent files. As a result, a comparatively small number of websites have hosted the large majority of torrents linking to copyright material, rendering those sites especially vulnerable to lawsuits. In response, some developers have sought ways to make publishing of files more anonymous while still retaining BitTorrent's speed advantage. The Shareaza client, for example, provides three altr.news.yahoo.com/27122006/185/gastro-c-est-la-treve-des-confiseurs.htmlernatives to BitTorrent: eDonkey2000, Gnutella, and Shareaza's native network, Gnutella2. If the tracker is down, it can finish the file over the other protocols, and/or find new (Shareaza) peers over G2. The use of distributed trackers is also one of the goals for Azureus 2.3.0.2 and BitTorrent 4.1.2. Another interesting idea that has surfaced recently in Azureus is virtual torrent. This idea is based on the distributed tracker approach and is used to describe some web resource. Right now, it is used for instant messaging. It is implemented using a special messaging protocol and requires an appropriate plugin. Anatomic P2P is another approach, which uses a decentralized network of nodes that route traffic to dynamic trackers.

4.4.2 Legal defenses

There are two major differences between BitTorrent and many other peer-to-peer file-trading systems, which advocates suggest make it less useful to those sharing copyrighted material without authorization. First, BitTorrent itself does not offer a search facility to find files by name. A user must find the initial torrent file by other means, such as a web search. Second, BitTorrent makes no attempt to conceal the host ultimately responsible for facilitating the sharing: a person who wishes to make a file available must run a tracker on a specific host or hosts and distribute the tracker address(es) in the .torrent file. While it is possible to simply operate a tracker on a server that is located where the copyright holder cannot take legal action, this feature of the protocol does imply some degree of vulnerability that other protocols lack. It is far easier to request that the server's ISP shut the site down than it is to find and identify every user sharing a file on a traditional peer-to-peer network. However, with the use of a distributed hash table (DHT), a tracker is no longer required, although they are often still used so that clients that do not support DHT can still connect to the swarm.

4.5 Other features of BitTorrent

4.5.1 Utilities

- TorrentSpy - An MS Windows tool which allows you to query a tracker about a specific torrent, view metadata info, check a file's hashes, etc. A new feature is the "create" tab for making torrent files to upload.
- maketorrent - A utility for creating torrent files, by the author of the burst! client. Version 1.x of MakeTorrent was a modified version of the Python 'completedir' program with extra features. Version 2.x is a complete rewrite in Delphi.
- completedir 1.0.1 - A utility for creating new .torrent files, part of the official BitTorrent client package. This is packaged as a Windows installer, get the source code for use with other platforms.
- Java BitTorrent Tools - A metafile viewer/editor, availability checker, and a basic tracker.
- DumpTorrentCGI - Use this handy web page to parse and output the contents of a .torrent file (from either your local system or a link URL.) Use this to quickly determine a torrent's hash ID or which tracker is hosting it.
- trackerlyze.pl 1.11 - A free (GPL) Perl script that analyzes the logfile of a tracker and creates graphs and reports of the bandwidth used and number of peers/seeds. See also: Home page.
- libbt - A library implementation of the BitTorrent protocol in C. This project is still under development, and is not suitable to end-users at this point.

4.5.2 Broadcatching

Another proposed feature combines RSS and BitTorrent to create a content delivery system dubbed broadcatching. Since a Steve Gillmor column for Ziff-Davis in December 2003, the discussion has spread

quickly among many bloggers (Techdirt, Ernest Miller, and former TechTV host Chris Pirillo, for example). As Scott Rar.news.yahoo.com/27122006/185/gastro-c-est-la-treve-des-confiseurs.htmlymond explained:

"I want RSS feeds of BitTorrent files. A script would periodically check the feed for new items, and use them to start the download. Then, I could find a trusted publisher of an Alias RSS feed, and 'subscribe' to all new episodes of the show, which would then start downloading automatically — like the 'season pass' feature of the TiVo."
While potential illegal uses abound as is the case with any new distribution method, this idea lends itself to a great number of ideas that could turn traditional distribution models on their heads, giving smaller operations a new opportunity for content distribution. The system leans on the cost-saving benefit of BitTorrent, where expenses are virtually non-existent; each downloader of a file participates in a portion of the distribution. One early adoption of this concept is IPTV show mariposaHD, which uses BitTorrent to distribute large (1-2 GB) WMVHD files of high-definition video.

RSS feeds layered on top keep track of the content, and because BitTorrent does cryptographic hashing of all data, subscribers to the feed can be sure they're getting what they think they're getting, whether that winds up being the latest Sopranos episode, or the latest Sveasoft firmware upgrade. (Naturally, however, ensuring that the same data reaches all nodes neglects the possibility that the original, source file may be corrupted or incorrectly labeled.)

One of the first open source attempts to create a client specifically for this was Democracy Player. The idea is already gaining momentum however, with other Free Software clients such as PenguinTV and KatchTV also now supporting broadcatching.

4.5.3 APIs

The BitTorrent web-service Prodigem has made available a feature to any web application capable of parsing XML through its standard Representational State Transfer (REST) based interface. Additionally, Torrenthut is developing a similar torrent API which will provide the same features, as well as further intuition to help bring the torrent community to Web 2.0 standards. Alongside this release is a first PHP application built using the API called PEP which will parse any Really Simple Syndication (RSS 2.0) feed and automatically create and seed a torrent for each enclosure found in that feed.

4.5.4 Multitracker

Another unofficial feature is an extension to the BitTorrent metadata format proposed by John Hoffman. It allows the use of multiple trackers per file, so if one tracker fails, others can continue supporting file transfer. It is implemented in several clients, such as BitTornado and µTorrent. Trackers are placed in groups, or tiers, with a tracker randomly chosen from the top tier and tried, moving to the next tier if all the trackers in the top tier fail.

4.6 BitTorrent Software Comparison

From: http://en.wikipedia.org/wiki/Comparison_of_BitTorrent_software

client	FLOSS	Runs on Linux/ Unix	Runs on Windows	Runs on Mac OS X	IPv6 Support	Max active torrents	Supports Super-seeding	Tracker	Malware-free	Supports UPnP Port Mapping [1]	Supports NAT Port Mapping Protocol	Supports NAT traversal	Supports DHT [2]	Supports Peer exchange	Supports BitTorrent protocol encryption
ABC [Yet Another BitTorrent Client]	Yes	Old version	Yes	No	?	∞	Yes	No	Yes	Yes	?	No	No	No	No
Acquisition	No	No	No	Yes	?	3, ∞ when registered	No	No	Yes	No	Yes	No	No	?	No
Anatomic P2P	Yes	Yes	Yes	Yes	?	∞	Yes	Separate download	Yes	Yes	?	?	?	?	No

Arctic Torrent	Yes	No	Yes	No	No	∞	No	No	Yes	No	?	No	No	?	No
Azureus	Yes	Yes	Yes	Yes	Yes	∞	Yes	Yes	Yes	Yes	?	DHT only	Yes [3]	Yes	Yes
BitComet	No	No	Yes	No	?	∞	No[4]	Separate download	Yes	Yes	?	Yes	Yes	Yes	Yes
BitLord	No	No	Yes	No	?	∞	No[4]	No	Adware	Yes	?	Yes	No	Yes	No
BitPump	?	?	Yes	?	?	?	?	?	?	?	?	?	?	?	?
BitTornado	Yes	Yes	Yes	Yes	Yes	∞	Yes	Yes [5]	Yes	Yes	?	No	No	Yes	No
BitTorrent / Mainline	Yes[6]	Yes	Yes	Yes	?	∞	No	Yes [5]	Yes	Yes	?	No	Yes	?	Yes
BitSpirit	No	No	Yes	No	?	∞	Yes	No	Yes	Yes	?	Yes	Yes	?	No
Bits on Wheels	No	No	No	Yes	?	∞	No	No	Yes	No	?	?	No	?	No

Blizzard Downloader	No	No	Yes	Yes	?	1	No	No	Yes	No	?	No	No	?	No
Blog Torrent	?	No	Yes	Yes	?	3	?	Yes	?	?	?	?	?	?	?
BtManager	Yes	Yes	Yes	Yes	?	?	?	?	?	?	?	?	?	?	No
burst!	Yes	No	Yes	No	?	20	Yes	No	Yes	?	?	?	?	?	No
ctorrent	Yes	Yes	No	No	No	∞	?	?	?	?	?	?	?	?	No
eDonkey2000	No	Yes	Yes	Yes	?	?	?	?	Adware [7]	?	?	?	?	?	No
freeloader	Yes	Yes	No	No	?	∞	?	No	Yes	?	?	?	?	?	No
G3 Torrent	Yes	No	Yes	No	?	∞	No	No	Yes	No	?	No	No	?	No
Gnome BitTorrent	Yes	Yes	No	No	?	∞	?	No	Yes	?	?	?	No	No	No

KTorrent	Yes	Yes	No	No	?	∞	No	No	Yes	Yes	?	No	Alpha	No	Alpha
Localhost	Yes	Yes	Yes	No	?	∞	Yes	Yes	Yes	Yes	?	Yes	Yes, required.	?	No
MLNet	Yes	Yes	Yes	Yes	No	∞	No	Yes	Yes	No	?	No	No	?	No
MooPolice	No	No	Yes	No	No	∞	No	No	Yes	Yes	?	No	Yes	Yes	No
Opera 9 browser	No	Yes	Yes	Yes	?	∞	?	No	Yes	No	?	?	No	No	No
QTorrent	Yes	Yes	No	No	?	∞	No	No	Yes	No	?	No	No	No	No
Qbittorrent	Yes	Yes	No	No	Yes	∞	No	No	Yes	Yes	?	No	No	Yes	No
rtorrent	Yes	Yes	No	Yes	No	∞	No	No	Yes	No	?	No	No	?	No
Rufus	Yes	Yes	Yes	No	No	∞	No	No	Yes	?	?	No	No	?	No
Shareaza	Yes	No	Yes	No	No	10	No	No	Yes	Yes	?	No	Yes [8]	?	No

Tomato Torrent	Yes	No	No	Yes	?	?	?	?	Yes	?	?	?	?	?	No
TorrentFlux	Yes	Yes	Yes	Yes	?	∞	Yes	No	Yes	Yes	?	No	No	No	No
Transmission	Yes	Yes	No	Yes	No	∞	No	No	Yes	No	?	No	No	?	No
Turbo Torrent	No	No	Yes	No	?	∞	Yes	Yes	No	Yes	?	No	No	?	No
TorrentSpy Rufus	Yes	No	Yes	No	?	∞	No	No	Adware [9]	?	?	?	No	?	No
µTorrent	No	No [10]	Yes	No	No	∞	Yes	Yes	Yes	Yes	?	No	Yes	Yes [11]	Yes
WizBit	Yes	No	No	No	?	?	No	No	Yes	No	?	No	No	No	No
XBT Client	Yes	No [12]	Yes	No	?	∞	Yes	Yes	Yes	Yes	?	?	No	?	No

ZipTorrent	No	No	Yes	No	?	?	?	?	Yes	Yes	?	No	?	?	No

BitTorrent client	FLOSS	Runs on Linux/ Unix	Runs on Windows	Runs on Mac OS X	IPv6 Support	Max active torrents	Supports Super-seeding	Tracker	Malware-free	Supports UPnP Port Mapping [1]	Supports NAT Port Mapping Protocol	Supports NAT traversal	Supports DHT [2]	Supports Peer exchange	Supports BitTorrent protocol encryption

BitTorrent client	Programming Language	Basis	Interface	Built-in disk cache[13]	Supports Web Seeding [14]	Supports Broadcatching (RSS)	Supports Prioritization	Supports Selective Downloading	Supports SOCKS for outgoing connections	Web Remote Control	Torrent Search Engine
ABC [Yet Another BitTorrent Client]	Python	BitTornado	GUI and web	?	Yes	No	Yes	Yes	?	Yes	Separate download
Acquisition	Objective-C and Cocoa	Limewire	GUI	?	No	No	No	No	No	No	No
Anatomic P2P	Python	BitTornado	GUI and old CLI	?	Yes	?	?	?	?	?	?

Arctic Torrent	C++	libtorrent	GUI	?	No	No	No	No	No	No	No
Azureus	Java and SWT	-	GUI, CLI, Telnet, Web, XMLoverHTTP remote control API	Yes	Yes[15]	Plugin	Yes	Yes	Yes	Plugin[16]	3rd party plugins
BitComet	C++	?	GUI	Yes	No	No	Yes	Yes	Yes	No	Built-in Web browser
BitLord	C++	BitComet	GUI	Yes	No	No	Yes	Yes	Yes	No	Built-in Web browser
BitPump	?	?	GUI	?	?	?	?	?	?	?	?
BitTornado	Python	BitTorrent	GUI and CLI	?	Yes	No	Yes	Yes	No	No	No
BitTorrent / Mainline	Python	-	GUI and CLI	?	No	No	No	No	No	No	Yes
BitSpirit	C++	BitComet	GUI	Yes	No	No	Yes	Yes	Yes	No	No

Bits on Wheels	Objective-C and Cocoa	-	GUI	?	?	?	?	No	?	No	?
Blizzard Downloader	?	BitTorrent client for early version	GUI	?	No	No	No	No	?	No	No
Blog Torrent	?	BitTorrent client for early version	GUI	?	No	No	No	No	?	No	No
BtManager	Python	?	GUI	?	?	?	?	?	?	?	?
burst!	Python	?	GUI	?	?	?	?	No	?	?	?
ctorrent	C++	?	CLI	?	?	?	?	Yes	?	?	?
eDonkey2000	C++	?	GUI	?	?	?	?	?	?	?	?
freeloader	Python	?	GUI	?	?	No	?	?	?	No	No
G3 Torrent	Python	BitTorrent	GUI and web	?	No	Yes	Yes	Yes	No	Yes	No

Gnome BitTorrent	Python	?	GUI	?	No	No	?	?	?	No	No
KTorrent	C++	-	GUI and CLI (Alpha)	?	No	No	Alpha	Yes	?	No	Built-in Web browser
Localhost	Java and SWT	Azureus	Web	Yes	No	Plugin	Yes	Yes	Yes	Plugin[16]	Yes
MLNet	Ocaml	-	CLI, Telnet, Web, Network GUI	Yes	No	No	Yes	No	No[17]	Yes	Yes[18]
MooPolice	C++	libtorrent	GUI	No	Yes	No	Yes	Yes	?	No	No
Opera 9 browser	C++	?	GUI	?	No	?	?	?	?	No	Built-into Web browser
QTorrent	C++	TheSHAD0W	GUI	?	?	?	?	?	?	?	?
Qbittorrent	C++	libtorrent	GUI	No	Yes	No	No	Yes	?	No	Yes[19]
rtorrent	C++	libTorrent	CLI	?	No	No	Yes	Yes	?	No	No

Rufus	Python	G3 Torrent	GUI and web	?	No	Yes	Yes	Yes	?	Yes	No
Shareaza	C++	Shareaza core	GUI and web	Yes	No	No	Yes	No	No	Yes	Yes
Tomato Torrent	Cocoa	BitTorrent?	GUI	?	?	?	?	?	?	?	?
TorrentFlux	PHP	BitTornado	Web	Yes	Yes	No	No	Yes	No	Yes	Yes
Transmission	C	-	GUI and CLI	?	No	No	No	No	No	No	No
Turbo Torrent	Python	G3 Torrent	GUI and web	?	No	No	Yes	Yes	?	Yes	Built-in Web browser
TorrentSpy Rufus	Python	Rufus	GUI	?	No	Yes	Yes	Yes	?	?	?
µTorrent	C++	-	GUI and Web (alpha)	Yes	No	Yes	Yes	Yes	Yes	Alpha	Yes
WizBit	Python	-	Mobile phone GUI (alpha)	No	No	No	No	?	No	No	Yes

XBT Client	C++	-	GUI and Web (alpha)	?	?	?	Yes	Yes	?	Yes	?
ZipTorrent	C++	libtorrent	GUI	?	?	Yes	?	?	?	?	?
BitTorrent client	**Programming Language**	**Basis**	**Interface**	**Built-in disk cache[13]**	**Supports Web Seeding [14]**	**Supports Broadcatching (RSS)**	**Supports Prioritisation**	**Supports Selective Downloading**	**Supports SOCKS for outgoing connections**	**Web Remote Control**	**Torrent Search Engine**

- ^ a b Automatically configure port forwarding (requires Router with UPnP support)
- ^ a b It allows you to use trackerless torrents or (with some clients) to resume normal torrents when their tracker is down
- ^ Not compatible with Official BitTorrent client DHT.
- ^ a b BitComet and BitLord clients interfere with Super-Seeding by other BitTorrent software as they do not acknowledge pieces sent by the Seeder. (**NOTE:** This has been fixed in version 0.70)
- ^ a b Tracker included with Linux binaries and with source, but not with Windows binary
- ^ BitTorrent Open Source License is not considered free by Debian Free Software Guidelines, see [1]
- ^ Used to contain both Adware and Spyware, non-"Pro" version continues to contain Adware. *New note* Latest versions (1.4.3+) of eDonkey all contain adware, which can not be opted out of, even with a "Pro" install.
- ^ It's not a DHT network, but Shareaza uses gnutella2 to find other Shareaza clients when the tracker is down. This option is disabled by default.
- ^ Unofficial version of Rufus created by TorrentSpy with WhenU SaveNow adware bundled in it; reportedly possible to opt-out during installation.
- ^ Runs perfectly under Wine.
- ^ Only exchanging with other µTorrent clients
- ^ Client backend can be run on linux or windows, client requires windows.
- ^ a b Reduces disk usage, file fragmentation (in case it is not preallocated) and latencies due to larger written blocks and cached data for hash checking finished pieces.
- ^ a b Recently implemented (unofficial) web seeding feature, see HTTP-Based Seeding Specification
- ^ Supports the Getright Webseeding spec in addition to the Bittornado spec[2]
- ^ a b Provides a Java-based and a simple HTML/JS based WebUI
- ^ See task #4401 Socks Proxy; however, http proxy is supported.
- ^ GUIs exist with built-in web browser
- ^ Integrated

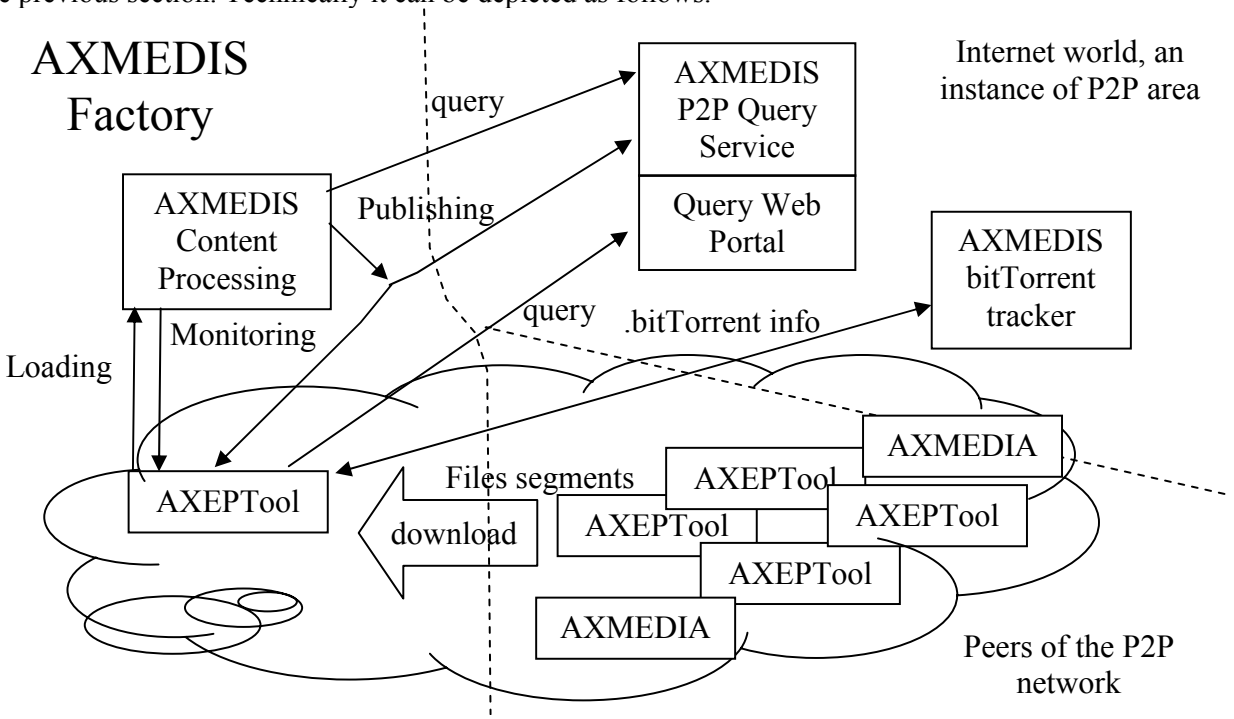
Relevant Features and solutions matching the interests and requirements of AXMEDIS:

client	FLOSS	Runs on Linux/ Unix	Runs on Windows	Runs on Mac OS X	Max active torrents	Supports Super-seeding	Tracker	Malware-free	Supports UPnP Port Mapping [1]	Supports NAT Port Mapping Protocol	Supports NAT traversal	Supports DHT [2]	Supports BitTorrent protocol encryption	Programming Language	Basis	Interface	Supports Web Seeding [14]	Supports Broadcatching (RSS)	Supports Prioritization	Supports Selective Downloading	Supports SOCKS for outgoing connections	Web Remote Control	Torrent Search Engine
Arctic Torrent	Yes	No	Yes	No	∞	No	No	Yes	No	?	No	No	No	C++	libtorrent	GUI	No	No	No	No	No	No	No
Azureus	Yes	Yes	Yes	Yes	∞	Yes	Yes	Yes	Yes	?	DHT only	Yes [3]	Yes	Java and SWT	-	GUI, CLI, Telnet, Web, XML over HTTP remote control API	Yes[15]	Plug in	Yes	Yes	Yes	Plugin[16]	3rd party plugins
Localhost	Yes	Yes	Yes	No	∞	Yes	Yes	Yes	Yes	?	Yes	Yes, required.	No	Java and SWT	Azureus	Web	No	Plug in	Yes	Yes	Yes	Plugin[16]	Yes
Shareaza	Yes	No	Yes	No	10	No	No	Yes	Yes	?	No	Yes [8]	No	C++	Shareaza core	GUI and web	No	No	Yes	No	No	Yes	Yes

XBT Client	Yes	No [12]	Yes	No	∞	Yes	Yes	Yes	Yes	?	?	No	No	C++	-	GUI and Web (alph	?	?	Yes	Yes	?	Yes	?

5 General B2B scenarios

The general solution and architecture of AXMEDIS framework tools integrated with P2P tools is reported in the previous section. Technically it can be depicted as follows.



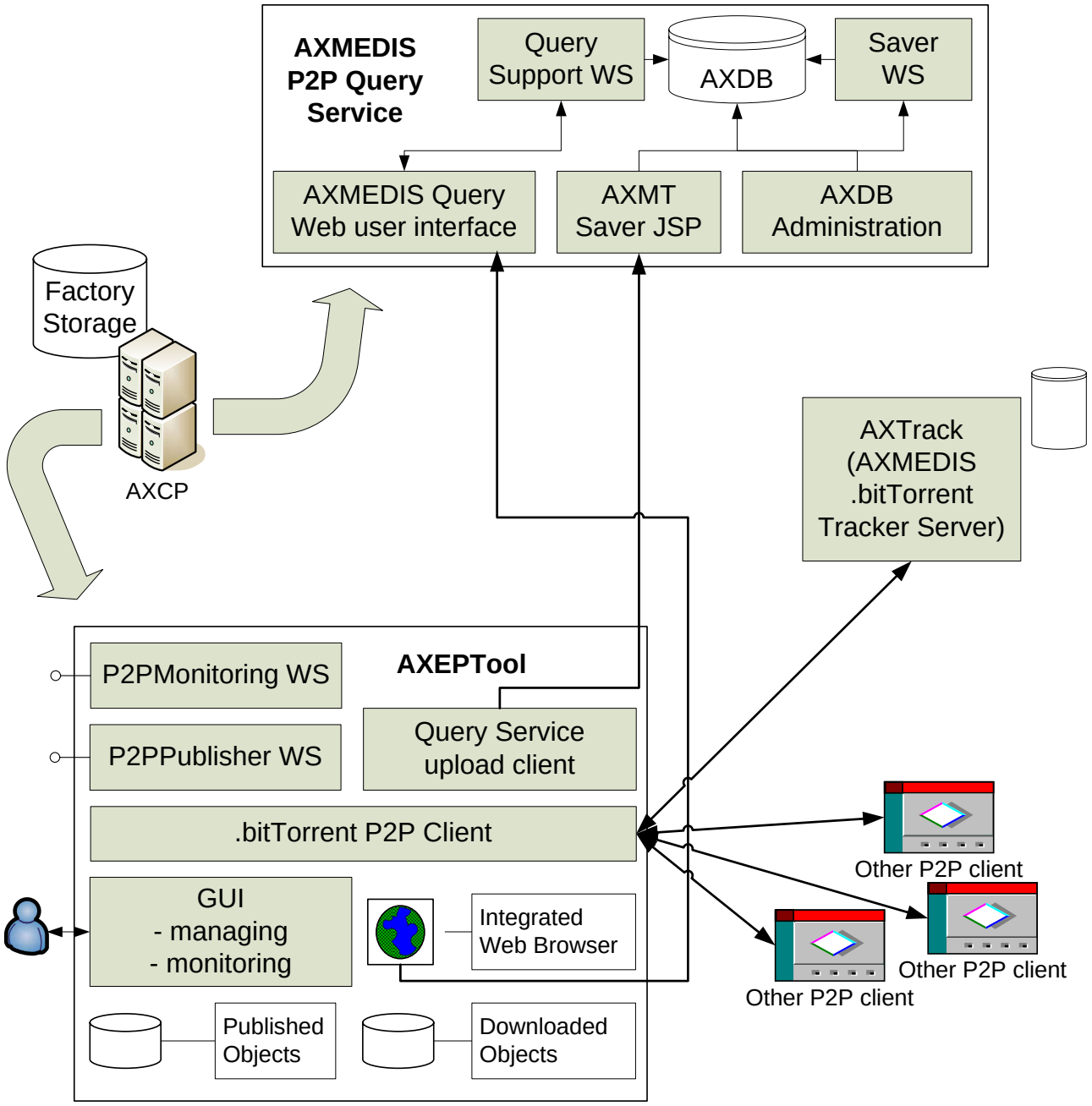
The P2P Clients can be both: AXEPTool and/or AXMEDIA P2P tools.

In this architecture for the B2B level we have:

- **AXMEDIS factory** (under responsibility of DSI), a place in which the AXMEDIS objects are created and/or are used to create other more complex objects, or are distributed towards other distribution channels, for example, broadcast,
- **AXMEDIS P2P Query Service** (under responsibility of EXITECH): a server located to cover a P2P community in which the database of metadata are located and indexed to recover AXOIDs which are at disposal for P2P download. AXMEDIS P2P Query Service may be connected with other AXMEDIS P2P Query Services to create a higher level of P2P sharing of content.
- **AXMEDIS BitTorrent Tracker**: a server derived from BitTorrent Tracker technology and solution to host BitTorrent information, update them, and provide them to AXMEDIS P2P clients according to the BitTorrent protocol. In a global geographic system, many AXMEDIS BitTorrent Trackers may survive to provide services.
- **.bitTorrent info:** BitTorrent information file containing the information created by a bitTorrent Maker processing the file to be shared into the P2P area. The AXMEDIS Content has to be treated in a different manner w.r.t. traditional media, so in the bittorrent info the AXOID of the Object has to be added in the custom information.
- **AXEPTool**: a P2P client tool derived from a BitTorrent Client for B2B P2P sharing of AXMEDIS objects.
- **AXMEDIA**: a P2P client tool derived from a BitTorrent Client for C2C AXMEDIS object sharing.

In sharing files, the AXEPTool may use the files contained into the local repositories of the AXMEDIA tools and vice versa. So that the P2P network is practically defined by the "AXMEDIS P2P Query Service" and by clients that get connection to the AXMEDIS BitTorrent Tracker to get the specific BitTorrent information associated to a download.

6 Integration of P2P in AXMEDIS for B2B



In the above figure the parts in Cyan are already present and available in the AXMEDIS Framework even if not fully used for this kind of context.

The P2P Clients can be both: AXEPTool and/or AXMEDIA P2P tools.

6.1 AXCP (AXMEDIS content processing) and relationships with the other tools

AXCP (AXMEDIS Content Processing) it is a set of AXMEDIS tools described in the introduction of this document. The AXCP Editor and Executor allow to write scripts and execute them. Thus, the AXCP tools allow scripting a large set of content processing activities including accessing at web services, producing objects (MPEG-21 and AXMEDIS and OMA formats), manipulating XML, and thus in connection of the P2P tools the AXCP has to permit the control of upload and download of files from the P2P area.
See the following document for details about the AXCP tools:
http://www.axmedis.org/documenti/view_documenti.php?doc_id=1728

In the AXMEDIS architecture for the content sharing at B2B level with the usage of the P2P AXEPTools the AXCP (AXMEDIS content processing) tool (engine processing scripts) has to be capable of:

- publishing the AXMEDIS objects having AXOID=ABCDX and a given AXMEDIS Object on file system (it could be previously recovered from the DB) by:
  i. Processing the AXMEDIS object for truncating the heavy media assets in order to produce the AXMT (AXMEDIS Truncated) version.
  ii. Sending AXMT (an simple AXMEDIS object without digital resources) to the AXMEDIS P2P Query service
  iii. Activating AXEPTool to publish the AXMEDIS object, the latter produces the bittorrent, notify on the tracker the new torrent file .
- performing queries on the AXMEDIS P2P Query Service to have as a results a list of AXMEDIS objects with their metadata and their AXOIDs.
- interacting with the AXEPTool via a WEB Service called “Monitoring Web Service” to:
  - ask for the downloads of some AXMEDIS objects passing to the AXEPTool the required AXOIDs
  - ask for the status of the objects under downloads/publishing, receiving back:
    - listing of objects published
    - list of objects under download including those already at the 100%
    - for each object the percentage of download performed with respect to their completion, and an estimated time to complete the download, for example: in 34 Minutes.
  - stop the download for some AXMEDIS object identified by its AXOID

6.2 AXMEDIS .bitTorrent files

BitTorrent is a common format that allows clients locate each others to exchanging media portions,
The **AXMEDIS .bitTorrent** has to be usable as by all interested AXMEDIS tools

- the AXCP
- the AXEPTools
- the AXMEDIA tools.

The .torrent file is produced as a result of a process that is typically performed by P2P BitTorrent Clients and can be passed to other tools as mentioned above to reach at the end the AXMEDIS BitTorrent Tracker Server.
The AXMEDIS .bitTorrent Maker is produced by customizing (if needed) a classical .bitTorrent maker, in many cases that functionality is directly present into the BitTorrent Client. The customization has to include at least the AXOID of a given AXMEDIS Object. This enables the possibility of activating the tracker only for AXMEDIS content super-distribution.

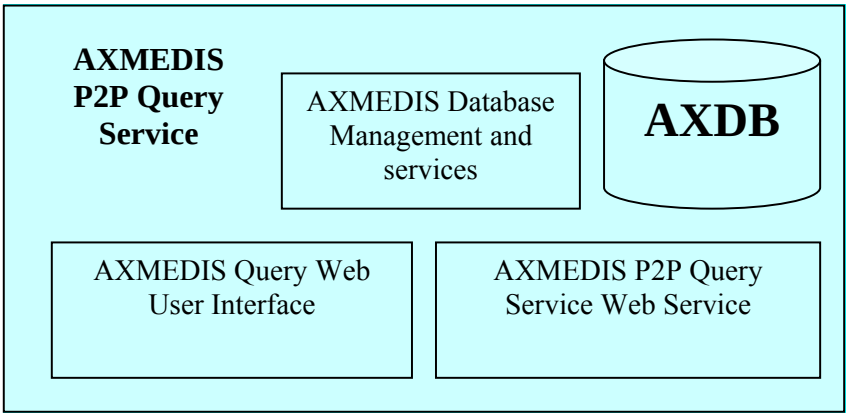
6.3 AXMEDIS P2P Query Service Server

Module/Tool Profile		
AXMEDIS P2P Query Service Server		
Responsible Name	Fioravanti	
Responsible Partner	EXITECH	
Status (proposed/approved)	Approved	
Implemented/not implemented	Under implementation	
Status of the implementation	under refinement	
Executable or Library/module (Support)	Web service	
Single Thread or Multithread	Multithread	
Language of Development	JAVA	
Platforms supported	All supported by JDK 1.5 and Tomcat 5.5.12 or higher	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/WebServices/P2PQSWs/source/	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.axmedis.org/repos/WebServices/P2PQSWs/bin/	
Reference to the AXFW location of the demonstrator executable tool for public download	None	
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any	To be defined	
Test cases (present/absent)	Absent	
Test cases location	None	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements	-- --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section

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

6.3.1 General Description of the Module

AXMEDIS P2P Query Service is a server that provides information about the objects indexed into the AXDB and at disposal of a community. Several Communities can be created, for example one for each distributor, by using and installing a different AXMEDIS P2P Query Service SERVER.
A query service that could be a simple redirection of the AXMEDIS database query on other AXMEDIS factories (see http://www.axmedis.org/documenti/view_documenti.php?doc_id=1728)
For more details on the structure of the AXMEDIS query support see:
http://www.axmedis.org/documenti/view_documenti.php?doc_id=1932



AXMEDIS P2P Query Service Server includes

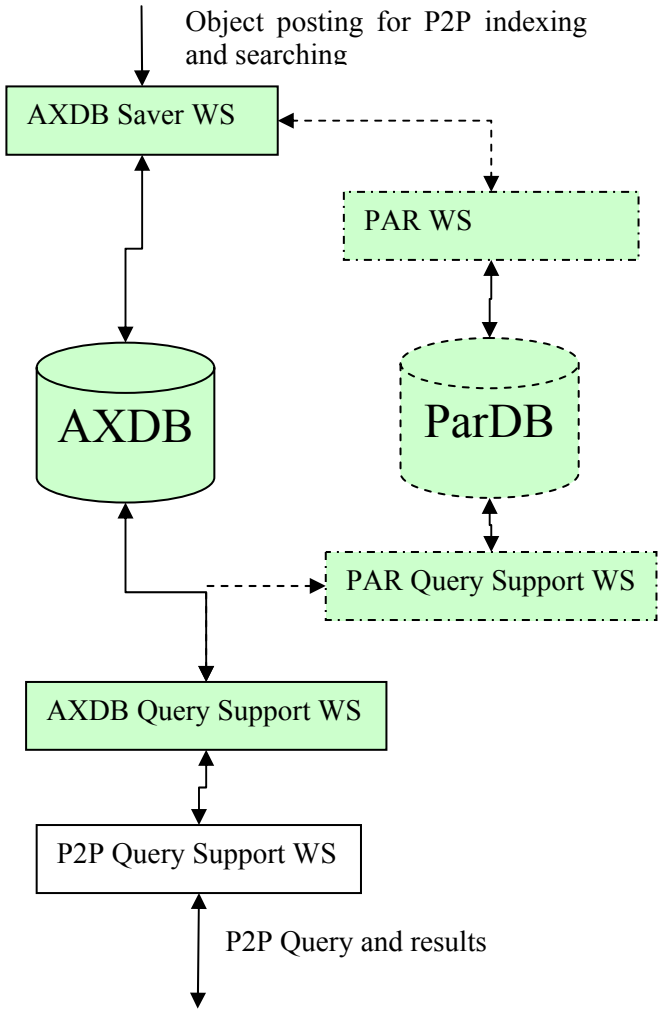
- **AXDB: AXMEDIS Database and its services** which is practically an AXMEDIS database that is mainly used to:
  - store the metadata of the objects. The objects that will be received are stored, but they are mainly stub objects without real resources inside in order to save time and bandwith
  - publishing/loading a new AXMEDIS object metadata and additional information
  - deleting/un-publishing of an AXMEDIS object.
- **AXMEDIS P2P Query Service Web Service** allows to establish a channel for:
  - receiving a query and providing a result in terms of AXOID, metadata. Torrent files will be recovered from the tracker from the AXOID.
- **AXMEDIS Web Query User Interface** allows to common users to make queries on the portal directly, the results are those described above. This is a simple and direct interface to make queries, probably used only by final users, unless a more suitable interface is implemented in the P2P tool. From that moment this will became an administrative interface only.

Please note that the **AXMEDIS P2P Query Service Web Service** can be used by:

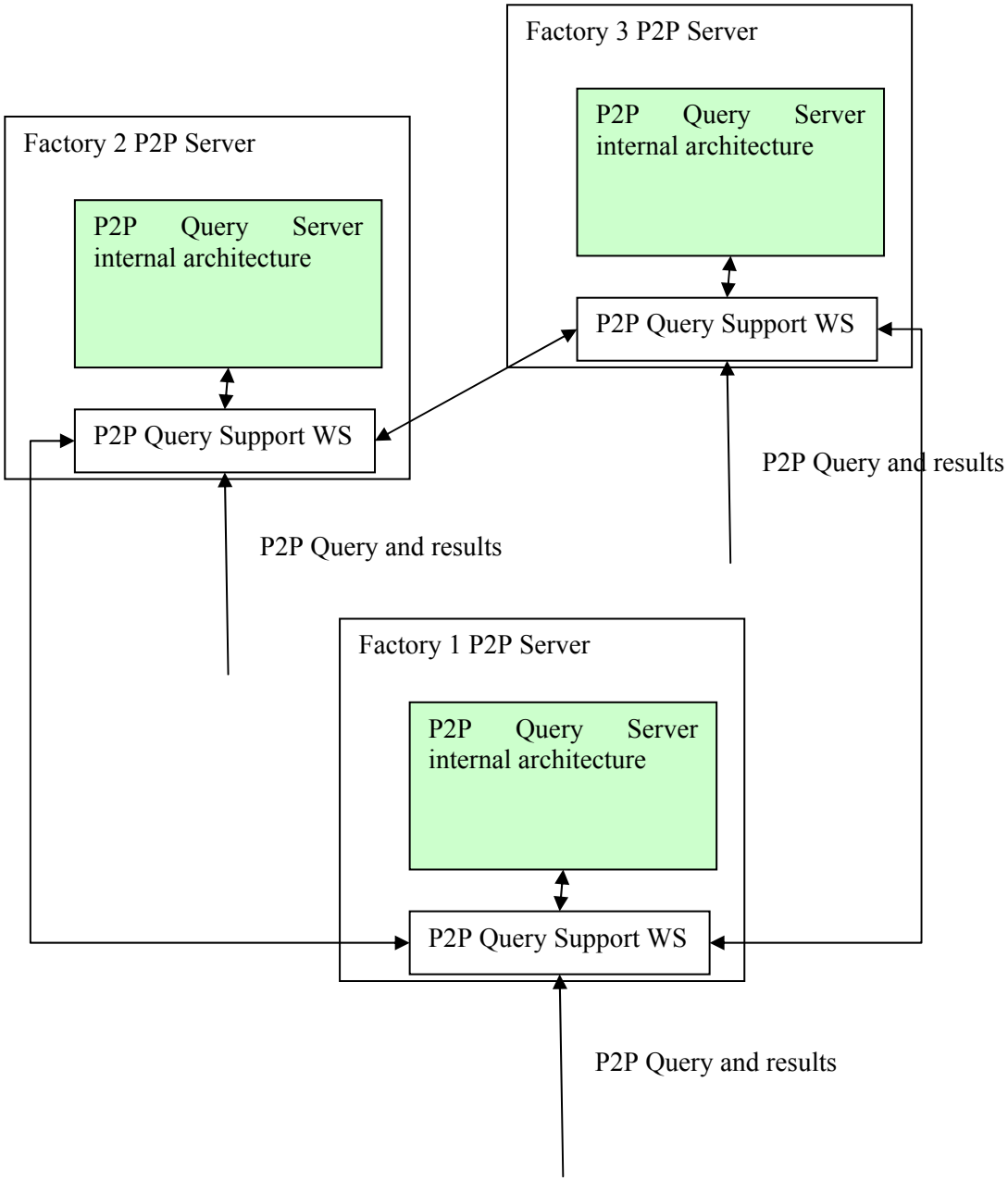
- the AXCP for automating the search for new objects in the P2P network
- the AXMEDIS query support for making a query on the Virtual Network of P2P tools of the several factories.

The architecture of tge P2P Query Web Service is aimed to minimize code duplication maximizing at the same time the reuse of stable and already tested code. This architecture estabilish the basis for a future scalability of the system on the basis of the needs that will be identified during the testing of the P2P system.

The component in green are reused for the Query Support architecture, while the white are developed for P2P. The green component with dashed lines can be considered optional.



The posting of an object to the AXDB is done by the submission of a simplified object without the resources since it is only necessary to index metadata of the object and not to store the entire object in the AXDB.
Moreover it is possible to save also bittorrent information by adding a custom metadata field that can be recovered by using the query mechanism together with the AXOID.
The P2P Query Support WS is the interface between the Main Query Support defined in DE 3.1.2.2.9 and the P2P Query server. This layer and the AXDB Query Support WS layer can be merged together in order to maximize performance if needed, but leaving them separate allows the scalability of the architecture toward a more general approach of querieng multiple node of the P2P architecture allowing for example to implement the architecture depicted below.



In this model the query on each factory can propagate to aumatically to other facories and the results are collected togheter by the P2P Query Support WS layer of each factory that became then a complex component replicating the functions of the MainQuerySupport at the P2P level.

6.3.2 Module Design in terms of Classes

The classes used as a basis for the P2P Query Web Service are a subset of those reported in DE 3.1.2.2.9 in Section 10.2.

The classes added are those related to the WebService itself that are equivalent to thos of DE 3.1.2.2.9 but placed in a different package: this is due to the fact that the server classes are generated from the same WSDL.
The engine for loading and saving are the same with respect to Chapter 7 of DE 3.1.2.2.9 and therefore the classes organization can be gathered from that document.

6.3.3 User interface description

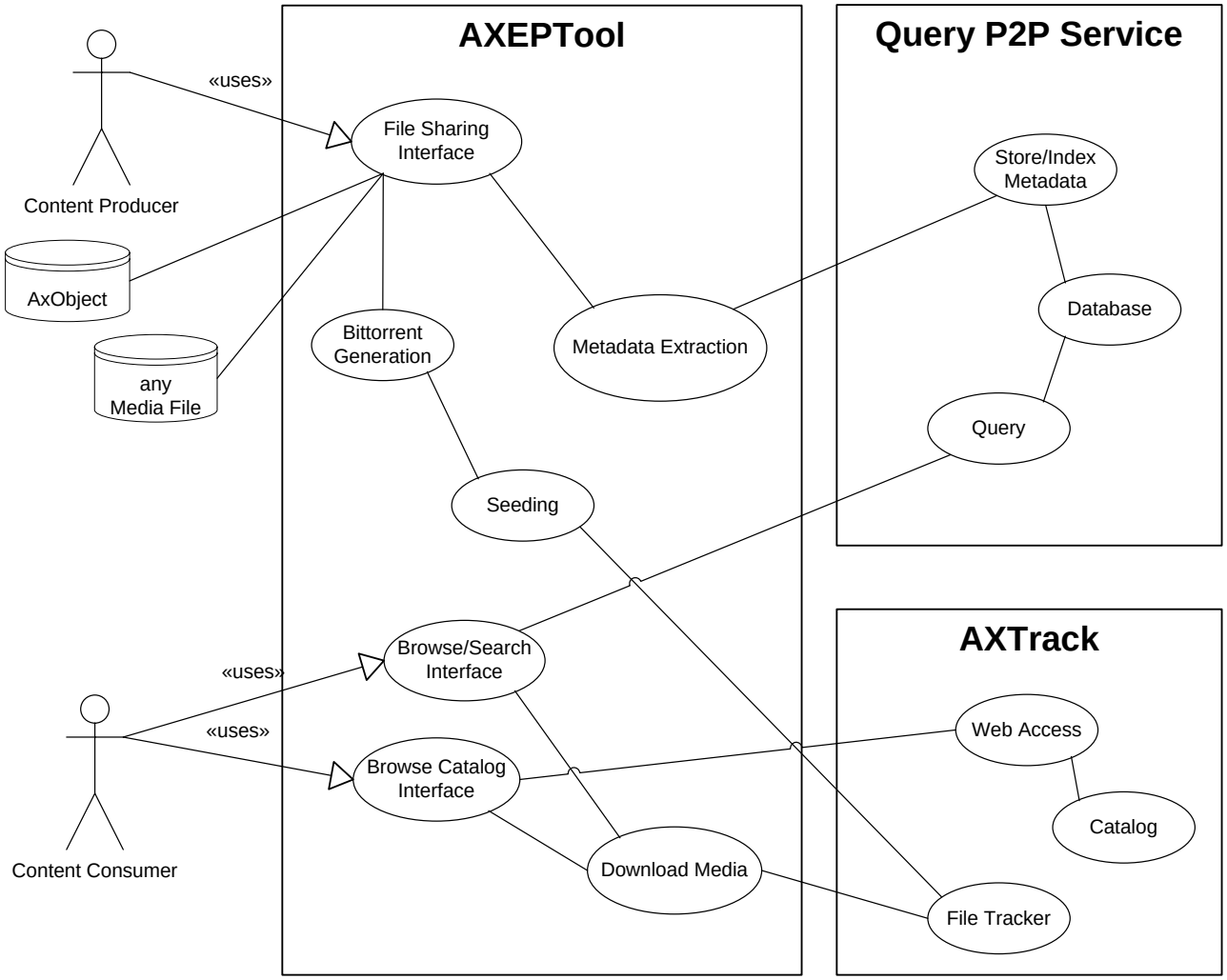
This module has no user interface since it is a webservice

6.3.4 Technical and Installation information

In order to install this service it is necessary to install the AXDB with its basic services (at least saving web service) and to deploy the P2P WS after configuring it.
In order to avoid duplication and incorrect information the deployment and installation of all the Exitech WebServices are documented in AXFW Maintenance in generic directory of the AXMEDIS document portal.

7 Low level peer to peer Use Cases

Here is the general use-case covering the exchange of files between a content producer and a customer. In the typical scenario, there is one content producer, but there are many customers.



The content producer connects to a file sharing interface and use it to put the file on the peer to peer network. First metadata and bittorrent info are generated locally from the file. Those metadata are then uploaded to one or more AXMEDIS P2P Query Service, where they are indexed for subsequent query, and to one or more AXTrack Web Service as seed notification. The File Tracker starts keeping track of the files.

After the tracker is ready to track the files, the AXEPTool could automatically start a seeding process (not mandatory, but configurable option). The seeding process will then register itself to the tracker, so that potential customers will know that the content producer has one full copy of the file.

The content download can start by operating in two different manners:

1. the search approach and the browse approach. The first one is based on the Query P2P Service and the information it stores.
2. The second it is simply a list which exhaustively reports all the content which is downloadable by the AxTrack instance.

The downloading process will connect to the tracker sending the information that he has an empty file list. The tracker will then inform it of the places where it can retrieve the file parts.

In the above example, there is only one seed and one downloader, so the answer of the tracker will be to grab it from the content producer. But, as soon as the customer has downloaded at least one small file part, the tracker will redirect other willing customers to both the content producer and the customer.

The traditional bittorrent protocol take places at this phase, whit the sole variation of being able to select to filter media in AXMEDIS form and/or traditional formats (e.g., avi, mp3, zip).

8 General architecture and relationships among the modules produced

The AXMEDIS P2P stack is decomposed in two modules. The first one being the AXEPTool (or AXMedia) and the second one being the AXTrack.

The AXTrack is technically speaking a tracker. It keeps track of each piece of file shared on the network. In Bittorrent, each file is decomposed into small pieces. Keeping the location of each piece on the network is the work of the tracker. Thanks to it, the AXEPTool is able to know where it can retrieve the pieces constituting a file.

But there's more than a simple Bittorrent stack in AXMEDIS tools. The AXTrack has been extended to do much more than simple bittorrent tracking :

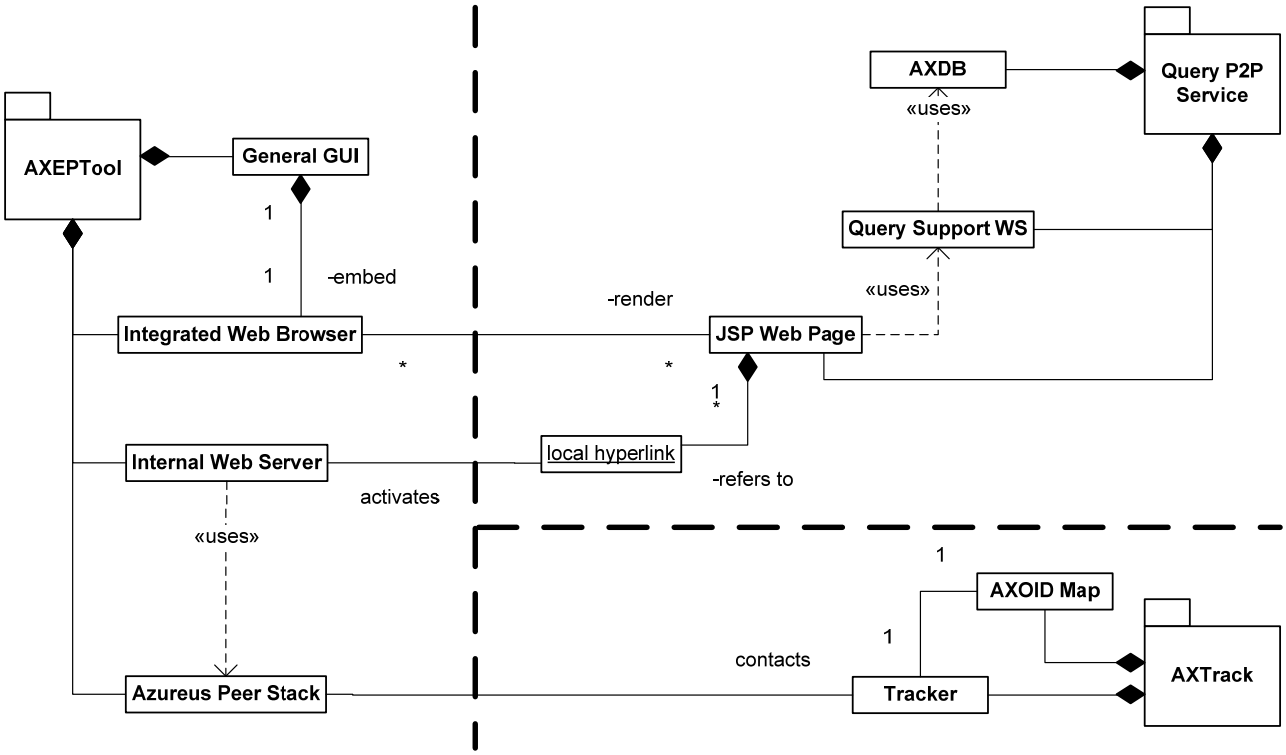
- It optionally keeps a central directory of shared files.
- It is a central directory of downloadable files on the network.

The main AXMEDIS modules are:

- AXEPTool and in AXMEDIA tool
- AxTrack.

Interconnection between the modules is done via a web-service interface. During Milestone 1 and Milestone 2, the web-services are implemented as a REST (Representational State Transfer) Web Services. They may be ported a classic SOAP web service architecture in subsequent milestones or some of them may stay as REST services as for many task REST is sufficient.

Here is a detailed diagram of the Peer To Peer subsystem :



All modules are quite simple in terms of interaction. As most of Web Services implemented in Milestone 1 and 2 are REST web services, the JSP is the main entry point of the communication between server and the client. The integrated web server also serves as a user interface. Then the tracker communicates with the usual peer to peer stack.

The only tricky point in the architecture is the integrated web browser inside the AXEPTool. This has been done as a replacement to the LiveConnect technology. Usually when you have a web browser and a java applet inside it, you can interact between JavaScript and the applet using the LiveConnect technology. Here we wanted to interact between the AXEPTool (containing the web browser), and some javascript inside the pages. But the only integrated web browser supporting LiveConnect was the Jrex component and after testing it was deemed inappropriate for stability issues. As a result, we used Sun's JDIC which does not support LiveConnect and had to find a replacement.

The replacement is the idea of having an integrated web browser listening only on incoming connections from localhost. As a result, if we generate in the JSP pages a link pointing to localhost on the integrated web server of the client, we are able to generate pages which will interact with their web browser.

On deployment versions, care should be taken in choosing a port number high enough so that the software does not interfere with installed software. The port can be configured and the localhost mechanisms are customized on the base of http request parameter. The default port (if not provided) is 10080.

Please note the additional component in AXTrack called AXOID map, which simply represent a table of associations that links received AXMEDIS Objects to their identities in AXMEDIS (their AXOIDS), this is the basis for enabling:

- AXEPTool can ask for the download of a specific AXOID (and not for a simple “file” through the hash), which is a main feature for AXMEDIS distribution
- AXTrack can turned ON/OFF to be exclusively used for AXMEDIS content distribution as a administration feature.
- Download statistics reporting related to AXMEDIS objects and related IDs.

9 Module or Executable Tool AXEPTool

Module/Tool Profile		
AXEPTool		
Responsible Name	Franck Coppola	
Responsible Partner	DSI (contractor) Hexaglobe (sub)	
Status (proposed/approved)	Approved	
Implemented/not implemented	Implemented	
Status of the implementation	Work In Progress	
Executable or Library/module (Support)		
Single Thread or Multithread	Multithreaded	
Language of Development	Java	
Platforms supported	Windows, Linux (any POSIX compliant system on whose JDIC can build would be supported).	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Applications/axeptool	
Reference to the AXFW location of the demonstrator executable tool for internal download		
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)	Absent	
Test cases location	http://////////////////	
Usage of the AXMEDIS configuration manager (yes/no)	No	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved		
Major pending requirements	-- GUI not yet tailored to needs -- Integration with remaining of the project	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section

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

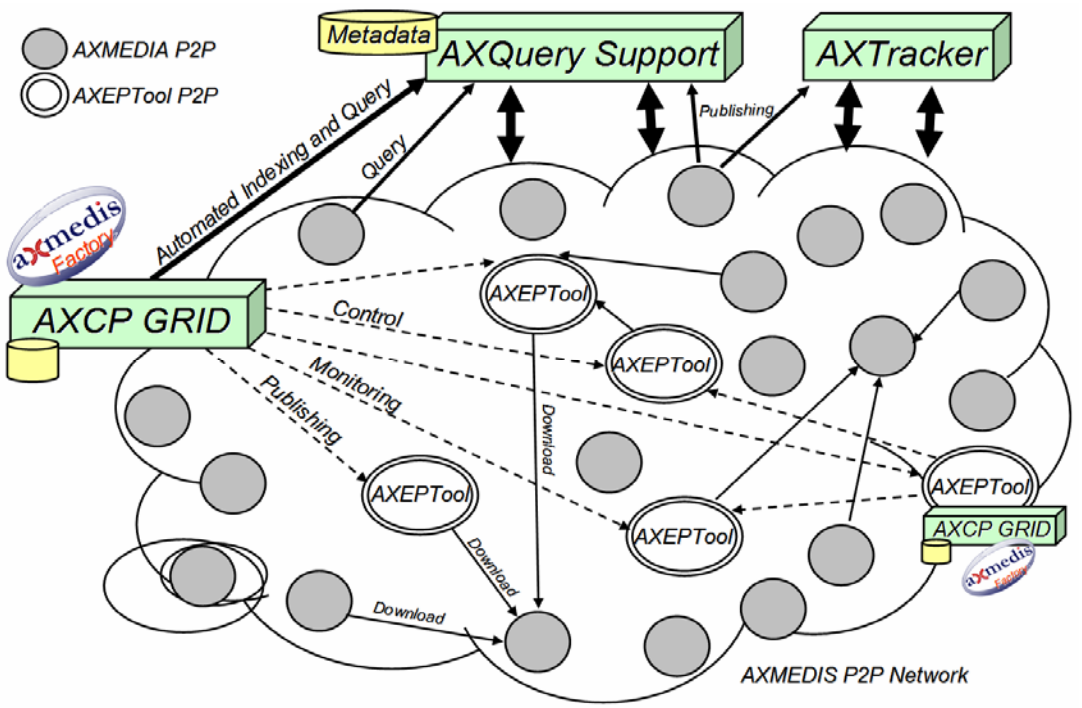
9.1 General Description of the Module

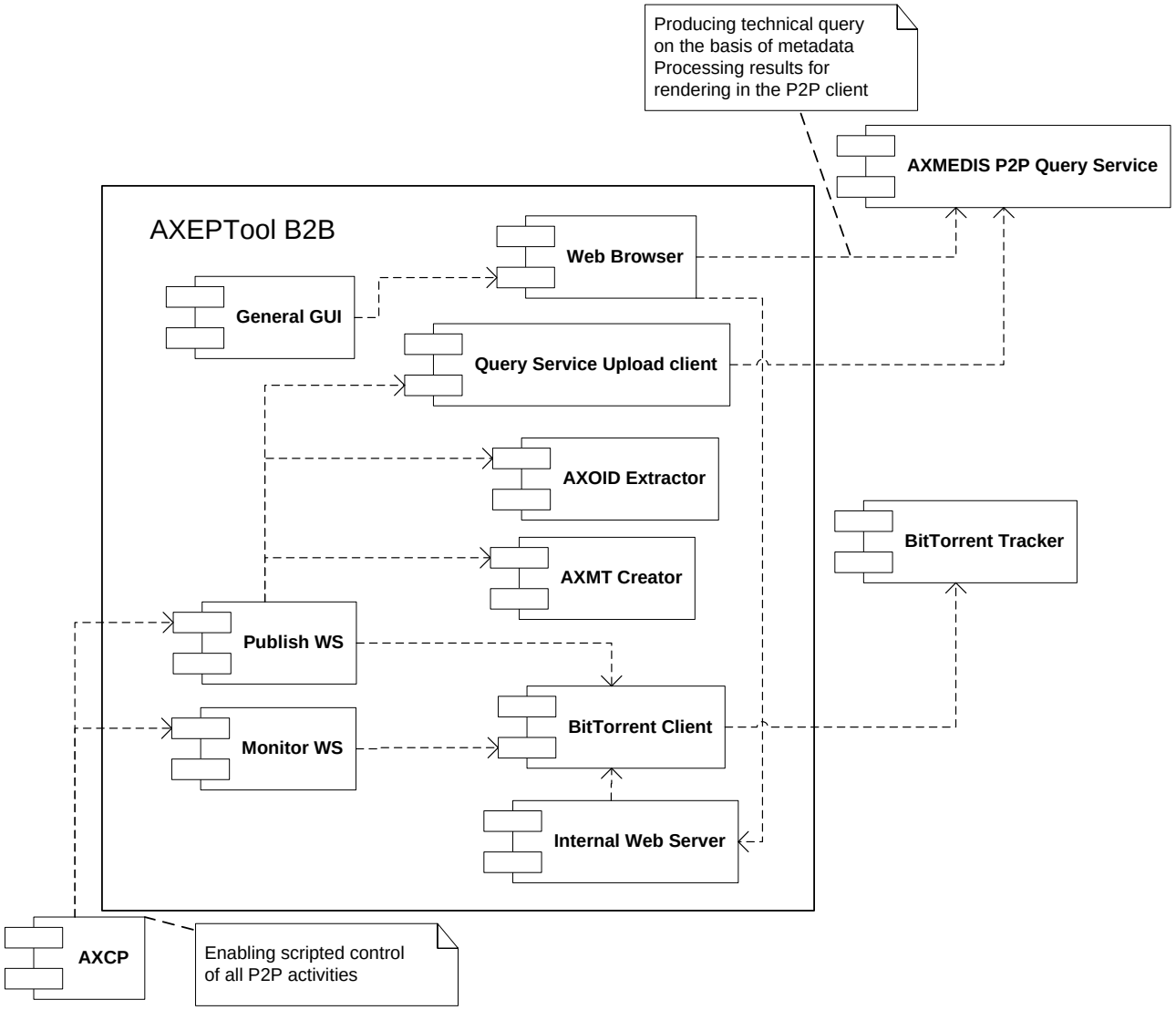
AXEPTool is a P2P client tool derived from bitTorrent Client for B2B P2P sharing and thus also for interacting with the AXMEDIS Factory and specifically with the AXCP tool.

The AXMEDIS AXEPTool P2P Client is going to be realized with .bitTorrent Azureus client (see the list in the previous section), which has as major features those presented in the previous table that for the purpose can be summarized as:

- Developed in Java
- Support protected communication with the Tracker
- Presents a monitor interface

It seems that the present Plugin technology of Azureus does not allow to do much and for sure not to control downloads and other facts that we need to enforce. So that the work is in effect to the modification of the P2P client.


AXMEDIA P2P
AXEPTool P2P
Metadata
AXQuery Support
AXTracker
Automated Indexing and Query
Query
Publishing
AXCP GRID
Control
Monitoring
Publishing
Download
AXEPTool
AXEPTool
AXEPTool
AXEPTool
AXEPTool
AXCP GRID
Download
Download
AXMEDIS P2P Network

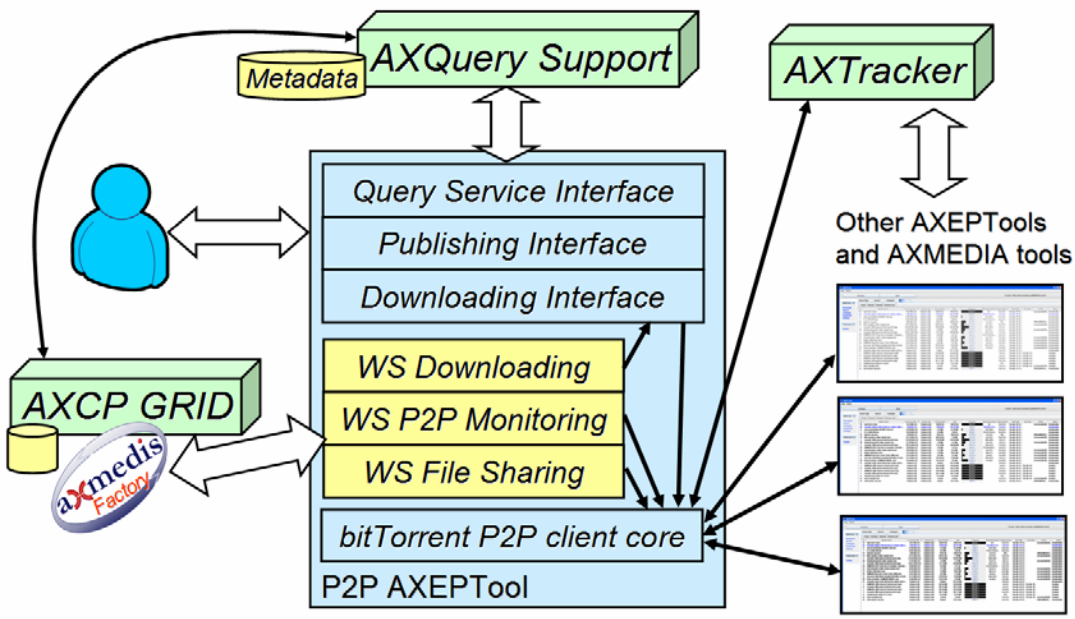

Producing technical query on the basis of metadata Processing results for rendering in the P2P client
AXMEDIS P2P Query Service
AXEPTool B2B
Web Browser
General GUI
Query Service Upload client
AXOID Extractor
AXMT Creator
BitTorrent Tracker
Publish WS
Monitor WS
BitTorrent Client
Internal Web Server
AXCP
Enabling scripted control of all P2P activities

The AXEPTool is a special P2P BitTorrent Client Node, suitable to cover the above mentioned requirements and roles in the P2P Network for B2B. It is used by content producers, distributors, integrators, aggregators, and expert users, etc., for automating activities of publishing, downloading and monitoring of the P2P network. Therefore, it accepts formal requests of publishing (file sharing), downloading, and monitoring via Web Services. In the AXMEDIS solution, the AXCP GRID Node may use those Web Services from the Java Script code which can be executed by any GRID node of the AXCP GRID. In addition, specific tools may be created for interacting with the AXEPTool via Web Services. In more details, the Web Services exposed by the AXEPTool P2P Clients, the AXQuery Support and the information provided by the AXTrack allow the AXCP GRID to automate procedures publishing, downloading and monitoring the AXMEDIS P2P Network.

The AXMEDIS AXEPTool P2P Client:

- Include an Integrated Web Browser and an Internal Web Server in order to easily communicate with the Web Portal realized by P2P Query Service.
- Provide a **P2PPublisher WS** for receiving objects from another tool (for example from the AXMEDIS AXCP Engine) to be published on the P2P network and thus for posting the BitTorrent information file and/or making accessible the full version of the file for other peers. It provides a functionality via WEB service to **bitTorrentURL publishObject(localFileName or filepath, AXOID, trackerURL)** publication of an object, its metadata have been already posted on the AXMEDIS P2P Query Service Server; the contract consist in posting/saving the AXMEDIS object passed into the **Published Object repository** of the P2P client tool and create a bittorrent file, thus posting bittorrent file on the selected tracker URL. If you have already published an object, and you call this functionality with a local file name it tries to post the correspondent .torrent file on the Tracker. Using filepath instead provides object publishing too on the Tracker's catalogue.
- The objects (AXMEDIS or not) can also be published manually from the user interface of the AXEPTool.
- Include two different repository one for Downloaded (or downloading) and one for Published objects. The bittorrent will automatically make available a downloaded media (or media) part, the distinction is only for clearly distinguish which content has been introduced in the P2P network from the client itself.
- Provide a **Monitoring Tools User Interface** for:
    - Observing the status of the plan to Downloaded files, % of completion, who is involved, download rate, etc.
    - Estimation of the time to completion
    - All the examples related to the Azureus Client have been depicted in the previous sections.
- Provides a **P2PMonitoring WS** with the following functionalities:
    - **P2PObjectList getPublishedObjectsList()**, returns a list of objects which are present in the directory of Published objects. The list depicts all media (both AXMEDIS or general), including filename, AXOID (if any) and infoHash.
    - **PublishStatus getPublicationStatus(AXOIDFileName selectedMedia)**, provides the current status of a given object which is in the **Published Object area of the P2P** environment and selected by the proper identification (AXOID or localFileName).
      It returns the amount of bytes, seeds and peers for each segment, and the list of contacting peer for a given media.
    - **result removePublishedObject(AXOIDFileName selectedMedia)**, deletes the object specified by the ID from the database/directory of Published objects. This is something that should not to be done in P2P systems, but when they are used for B2B the directory could reach the limit of their size and housekeeping is needed.
    - **result downloadObject(AXOID)**, starts the download of the object especified by the AXOID in the P2P environment. The AXOID field has to be added into the Tracker database.
    - **result downloadObjectURL(bittorrentURI)**, starts the download of the object specified by the torrent file URL in the P2P environment.

- **DownloadStatus monitorDownloadingStatus(AXOIDFileName selectedMedia)**, provides the current status a downloading objects in the P2P environment. It returns the current percentage of download, estimated time to completion (if possible), peers and seeds.
- **P2PObjectList getDownloadingObjectsList(),** returns the list of objects in the directory of Downloaded objects.
- **result controlDownloadingObject(AXOIDFileName selectedMedia, boolean start/stop)**, start/stops the download of the object specified by identification (AXOID or localFileName).
- **result removeDownloadedObject(AXOIDFileName selectedMedia),** deletes the object specified by the ID from the directory of Downloaded objects. This is something that should not to be done in P2P systems, but in some cases the directory could reach the limit of their size and housekeeping is needed.
- **mediaURL getDownloadedObjectURI (AXOIDFileName selectedMedia),** retrieves the object URI from which the downloaded at 100% object can be copied into the AXMEDIS database by the AXCP. This action will be performed by the AXCP directly accessing to that URL.
- **ObjectStatus getStatus (AXOIDFileName selectedMedia),** retrieves the status of an object in the AXEPTool's transfer list. This can be Downloading, Paused, Checking, Seeding, Error, Queued.
- **Sysinfo getSystemInformations(),** retrieves the main system informations and profile of the AXEPTool: CPU type, Operating System, RAM, AXEPTool's configuration file in XML format, User Name, AXEPTool's version, free HD space where AXEPTool is running.



On the basis of the basic functionalities of the AXMEDIS P2P network reported above it is possible to create procedures/scripts for:

- Publishing an object on all the AXEPTools of the P2P Network for example to accelerate the seeding of a given object on the network:

Axoid=getAXOID(MyFile);

publishing(MyAXEPTool, MyFile, TheAXTrack);

LA = listAXEPTool(TheAXTrack);

For each la of LA: download(la, Axoid);

- Notifying the publication of new objects/files in the network among the different AXCP GRIDs/AXEPTools controlled;
- Removing/deleting of objects/files from the network, at least from the AXEPTool nodes, AXTracks and AXQuery Support:

```
Axoid=getAXOID(MyFile);
axeptoolList=listAXEPTool(AXTrackURL);
For each axeptool of axeptoolList:
    delete(axeptool, AXOID);
```

- Monitoring the status of the P2P Network, discovering which are the most active/virtuous AXEPTools, their capabilities, how many downloads have been performed, how many segments have been provided and for whose objects/files, if they are active, etc. This allows to perform specific analysis to assess the reputation of Business actors on the basis of their behavior on the corresponding AXEPTool;
- Controlling the content seeded by the AXEPTools, for example constrained them to become an exact replica of each other (for an uniform seeding distribution), or imposing some distribution for the content in the network of the AXEPTools on the basis of the content distribution and statistical analysis:

```
LA=listAXEPTool(AXTrackURL);
rootNode=LA[0];
List= listDownloaded(rootNode);
For each la of LA:
    For each list in List:
        download(la, list);
```

- Activating automated queries for obtaining, downloading and posting these objects into the database of specific content collection on the basis of complex queries. So that, these active queries can be periodically activated to verify if some new content satisfies the criteria and, in the positive case, the automated download can be activated as well;
- Activating automated publishing on the P2P Network of accessible collections from the AXCP GRID and crawling facilities etc.

The solution has been obtained by exploiting and modifying the BitTorrent protocol and tools; integrating the P2P networking with those of the AXMEDIS Content Processing facilities, to allow setting up a large range of strategies for content distribution, seeding, publishing, promoting, etc. The main benefits of the proposed solution and architecture are scalability, content sharing control, efficient rules scheduling, fast object publishing and dissemination, and in general network controllability without constraining the actors of the network.

A javascript example that retrieves downloading object list from an AXEPTool is showed below:

```
//Prints the downloaded/downloading object list of an AXEPTool, (manager = AXP2PManager)
function printDownloadObjectList(manager)
{
print("Downloading Object List\n");
```

```
var IP = manager.uri.substring(7,manager.uri.indexOf(":",5));
print("AXEPTool IP: "+IP+"\n");
var download = DownloadObjectList(manager);
print("This AXEPTool has "+download.length+" downloaded/downloading objects\n");
if(download.length>0)
for(i=0;i<download.length;i++)
   print(download[i]);
else
print("Download Object List empty");
}
var start = true;
//write here your AXEPTool's IP
var axeptoolUri = "http://localhost:7780/WebServices/P2PMonitoring";

var trackerUri = "http://axtrk.axmedis.org:8080/AXTrackv2/";
var manager = new AXP2PManager(axeptoolUri, trackerUri);

//set the response timeout in seconds
manager.setTimeout(10);

//write here the object's URI (AXOID, filename or filepath)
var Uri = "";
var s = true;
printDownloadObjectList(manager);
```

In the following sections the WSDL formalization of the above two web services are reported.

Integration into the AXMEDIS AXEPTool of:

- certification and authentication mechanisms of AXMEDIS to be sure that the AXEPTool is a certified and authorized tool and person.
- A module for estimating fingerprints of resources, connection to the AXCS to verify if the estimated FP corresponds to the resource effectively contained in the digital object.

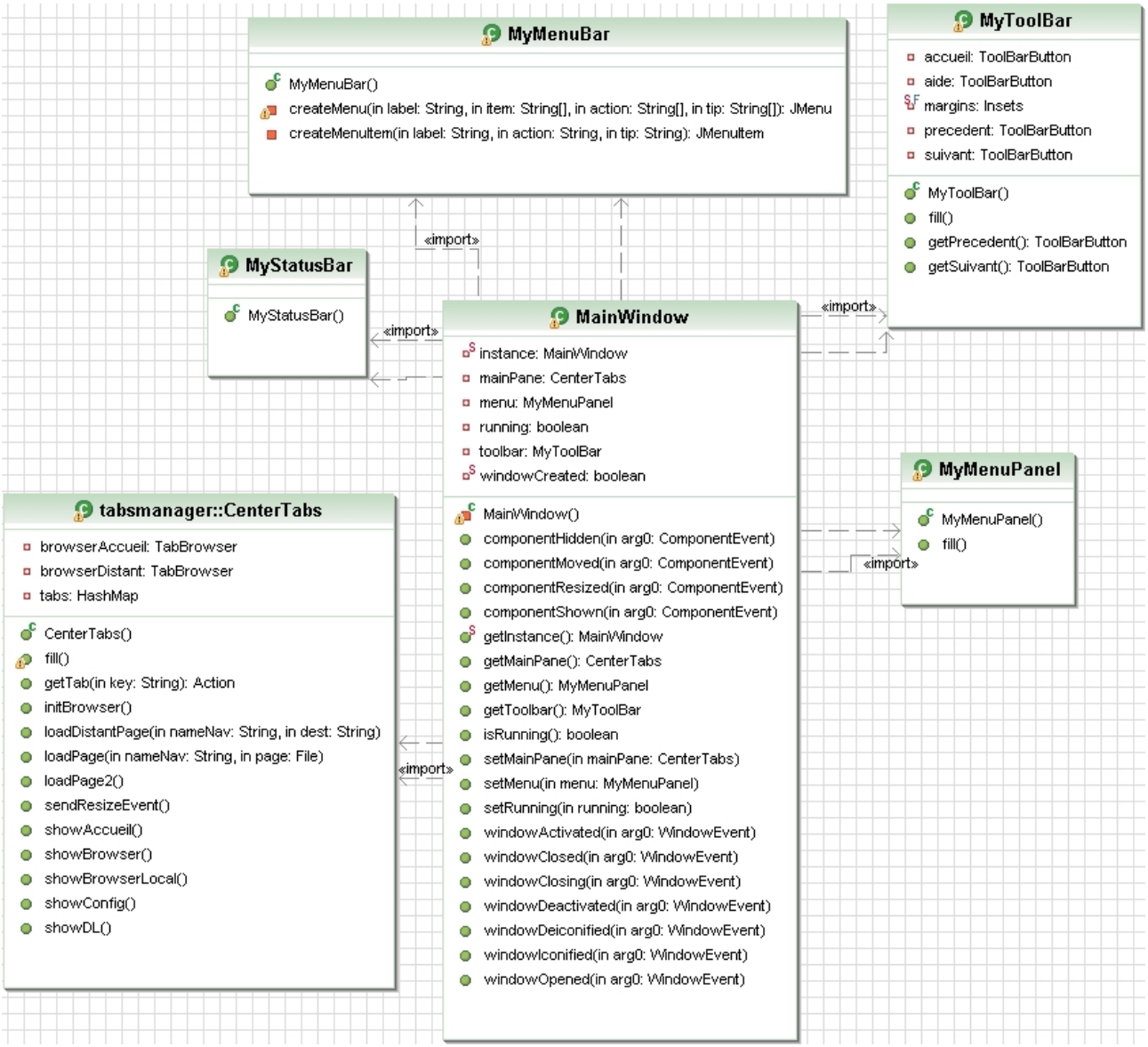
In order to operate, AXEPTool requires the use of the AXTrack. The AXTrack is a tracker server located on the network, usually in a well connected server room. It keeps tracks of files currently distributed on the peer to peer network. As a result, it also serves as an authoritative source for listing the available files on the network and tracking statistics.

As a result, part of the interface was implemented inside the AXTrack as a web interface, but it is not designed to be used directly. This is why the AXEPTool integrates a native web browser (either managed by the internet explorer engine or by mozilla's gecko engine) which displays the web pages served by the AXTrack.

The AXEPTool also integrates an internal web-server. It is used as a replacement of the LiveConnect technology to notify the AXEPTool of events inside the browser window. The result is fully cross-platform which is not the case with LiveConnect. Connection to this internal web-server will be restricted to connections from localhost. For the moment, it is mainly used to automatically launch downloads directly when browsing the catalog from the browser window connected to the AXTrack.

The application uses a customized version of the Azureus engine. We have a way to use the Azureus jar file (which contains a full executable Azureus instance) as a library. As a result, the AXEPTool contains a standard compliant and full featured BitTorrent stack.

9.2 Module Design in terms of Classes

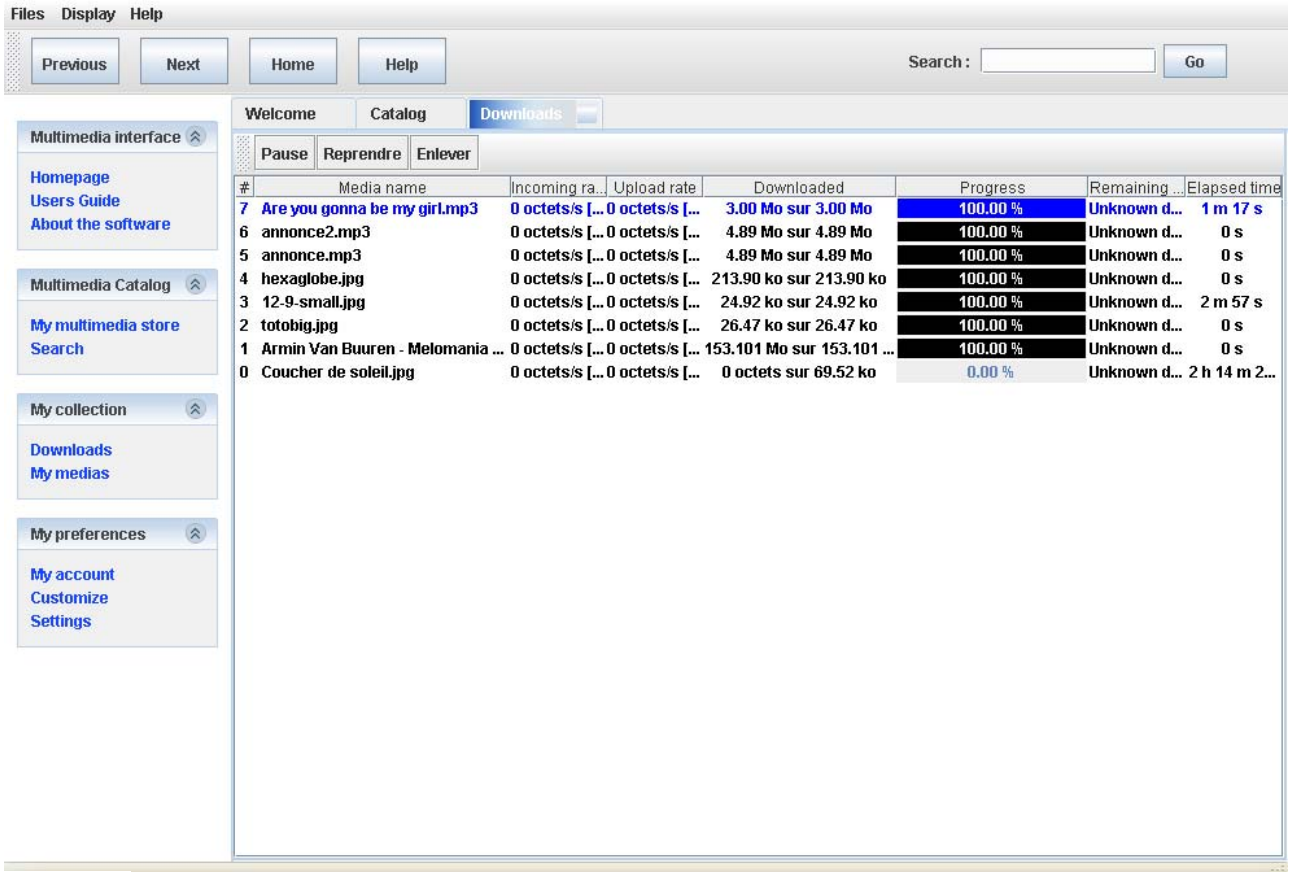


The MainWindow is the core of the graphic interface, it respect a Singleton Pattern. Its first instantiation is done by the "Main" program class. The MainWindow's private constructor instantiate graphics classes, which are shown around the MainWindow in the UML diagram.

- The MenuBar is the Highest bar, its menus are “File”, “help” etc … when clicked this menu call the MenuBarListener Class witch perform actions on the models, like generating a new torrent, or exiting the application.
- The ToolBar is the second bar with some buttons like “next” or “previous”, it inherits from the java swing class : JToolBar and its actions are caught by the JToolBarListener.
- MyStatusbar can modify the display of the text area at the bottom of the application.
- MyMenuPanel is the left menu of the application with pretty graphics possibilities. It inherits of the JXPanel properties, a Swing-X class which has specials functionalities for displaying text, like hiding menus used in the application.
- The “*tabsmanager*” package contains all classes needed for using the central application panel with some tabs. CenterTabs is the main class, a JTabbedPane, when it is initialized, it instantiate some CenterTab , interface pattern between the CenterTabs (tabs manager) and the contents of tabs. Contents of tabs are in the “*tabs*” package. Once tabs created, the CenterTabs create needed by calling the MyBrowser class. MyBrowser is an interface pattern for using the class WebBrowser from “*jdic*” jar package, which provides a web browser independent from operating systems.

9.3 User interface description

SnapShot of the User Interface : a new snapshot with AXOID identification and both Search and Catalog tabs.



The menu:

The menu allows users to perform some actions on the applications.

For the moment the only actions available is to “add a new torrent”.

Display : load particulars tabs (Browser/Catalog/Downloads/Parameters)

Help: a clickable hyperlink to the AXMEDIS portal web page http://www.axmedis.org

About: about the software version with a clickable hyperlink to the AXMEDIS portal web page http://www.axmedis.org

The toolbar :

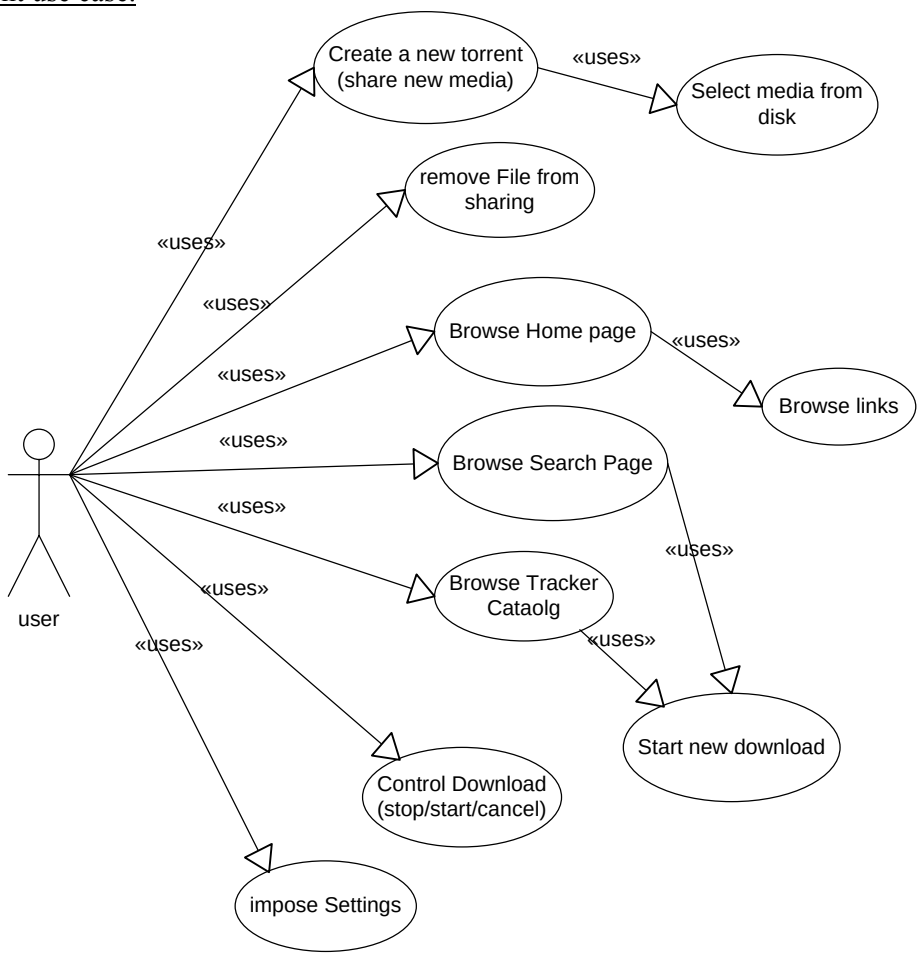
The tool bar act on the Browser Tabs, like Welcome and Catalog tabs. It allows the user to perform actions like those you can find in web browser :

- Go to Previous page
- Go to Next page
- Go to Home Page.
- Search for a torrent, will open a new tab with the torrents available for download.

The left menu :

It is a list of each tab that can be open in the main widget. The left menu is cut in four parts, each one are in a box that can collapse. Each part got some links for tabs, if the tab has been closed, it will be open, else nothing is done.

The main content use case.



Details :

In the main widget the user can switch from one tab to another. Some tabs are web browser, like Homepage and catalog.

- Homepage allow the user to browse a particular web site, set as homepage in the configuration. The user can use the web browser to follow links given on the homepage, but he can't go in any website that hasn't a link from the home page or a page that come from the home page.
- Catalog is a webpage static, which list all Medias available from an html page, generated and stored on the AXTrack project. Clicking on a media in the list will add it to the download and he will start automatically.

- A particular tab is the Transfer tab where download are listed in a widget. You will get information on :
    - Number of the download
    - Name of the Download
    - Download rate
    - Upload rate
    - Quantity of Bytes received
    - percentage of the downloaded Bytes
    - A estimation time
    - Time since the download has started.
    - AXOID/infoHash
    - Protected Object or not

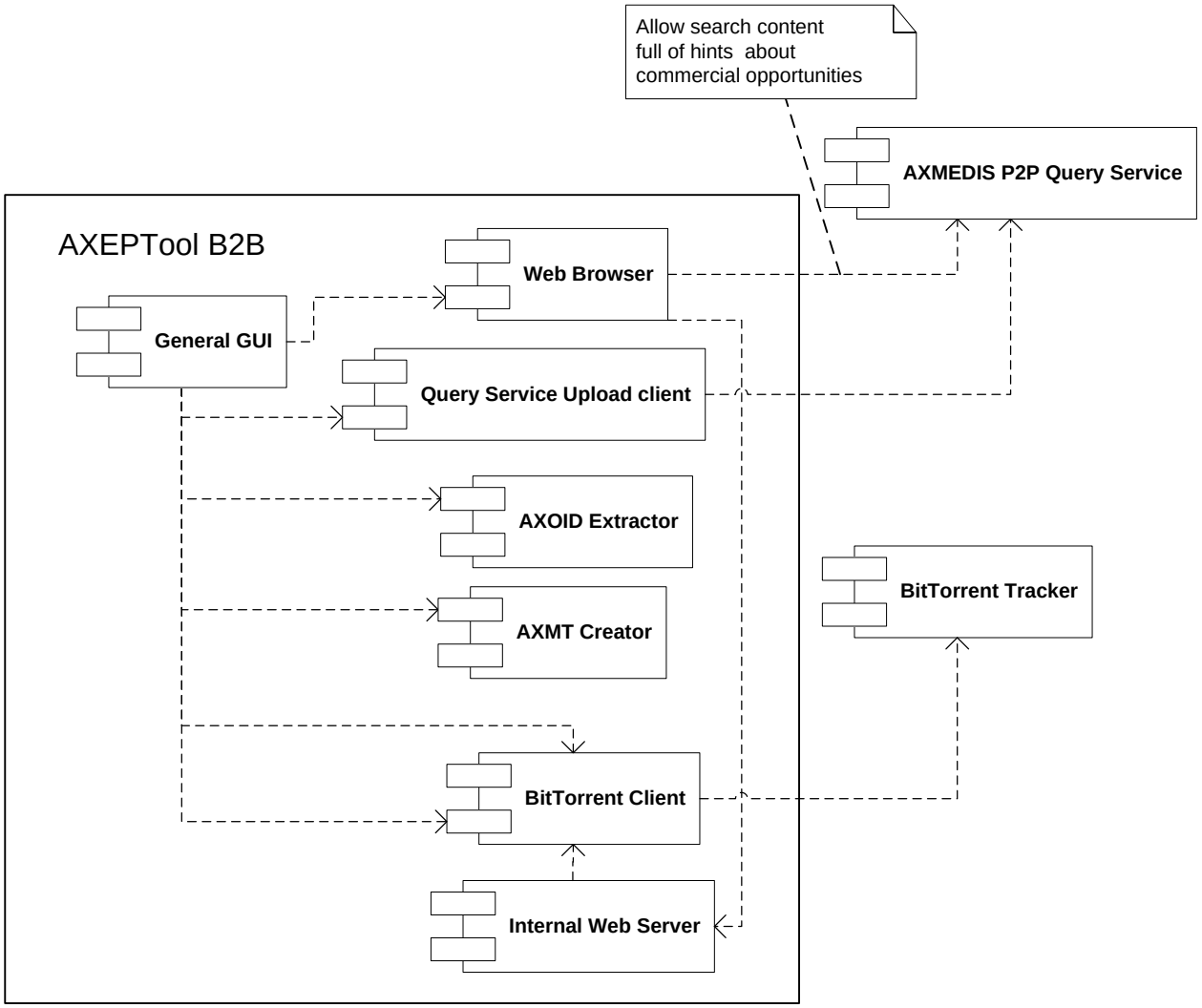
  And you can perform actions on each download:
    - Remove and…(deletes torrent file, delete file, delete both)
    - Pause (pauses an object downloading)
    - Resume (resumes an object downloading)
    - Remove (removes an object from the transfer list without deleting it from the disk)
- Open a Configuration Panel where you can set
    - proxy settings
    - Tracker's URL
    - Download directory
    - AXEPTool's URL
    - Listen start port for incoming connections (TCP)
- You can also edit the above settings in the AXEPTool configuration file and specify
    - UDP port for DHT
    - Tracker's catalogue URL
    - Local port (TCP) for Web Service functionalities
    - Query Server uploader URL
    - Query Web User Interface URL

Each of this tab can be closed, for opening you have to use the left menu link or the "display" menu.

9.4 Integration of P2P in AXMEDIA for C2C

A stand alone Java based P2P tool without Web Service server.
It has to be capable to access to the AXMEDIS P2P Query Service Server by means of a Web Service client.
What has to be done:

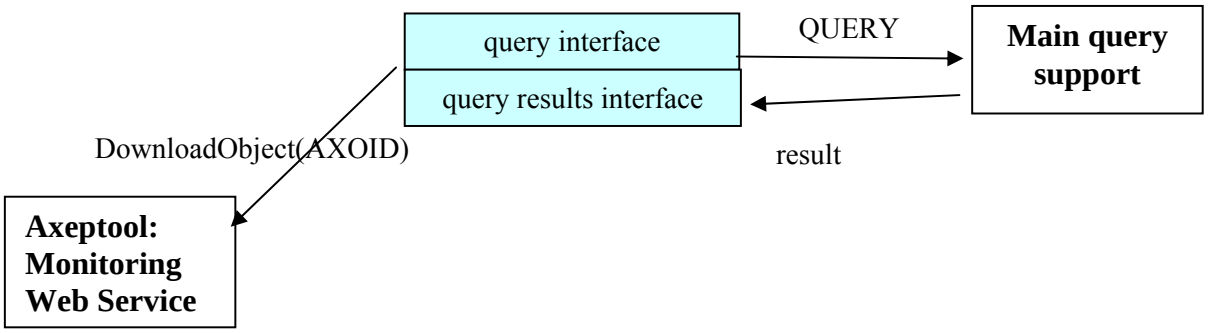
1) It has to work as the AXEPTool on AXOID
2) It has to allow making simple queries on the AXMEDIS P2P Query Service Server by means of a Web Service client. The query has to expose that is performed by an AXMEDIA tool.
3) It has to allow to provide results of the query, direct selection of the file to be downloaded, selecting the line in the query result

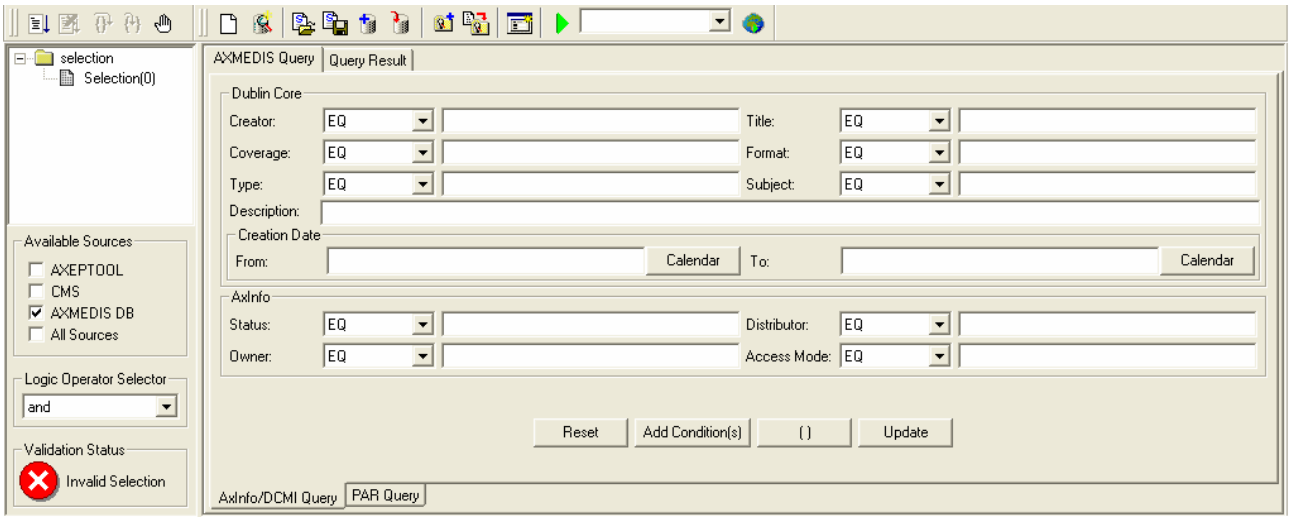


The AXMEDIA tools presents:

- a nicer user interface
- a simplified query user interface
- a simplified monitoring tool and user interface

9.5 Query Interface/Query Results Interface (DSI resp)





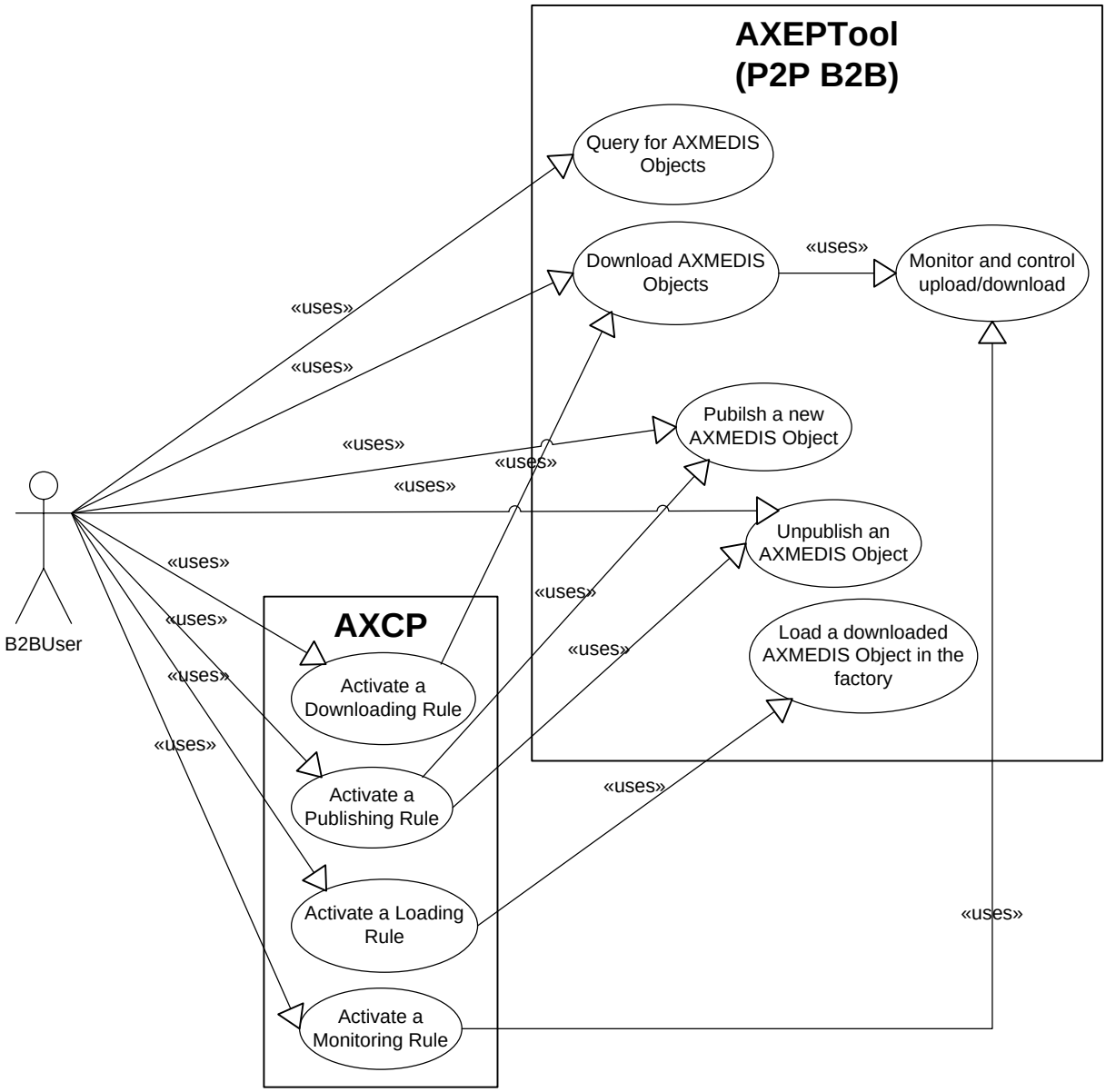
The above sophisticated Query interface will be at disposal in the AXCP tools in order to select content to be download automatically.

9.6 Usage Walk Through

In this section, the main use cases have been depicted in order to clarify functionalities to be exposed by AXEPTool (P2P B2B client).
In the following, use-case diagrams the main B2B user actions have been included.

The use cases of main interest are those considered for AXEPTool:

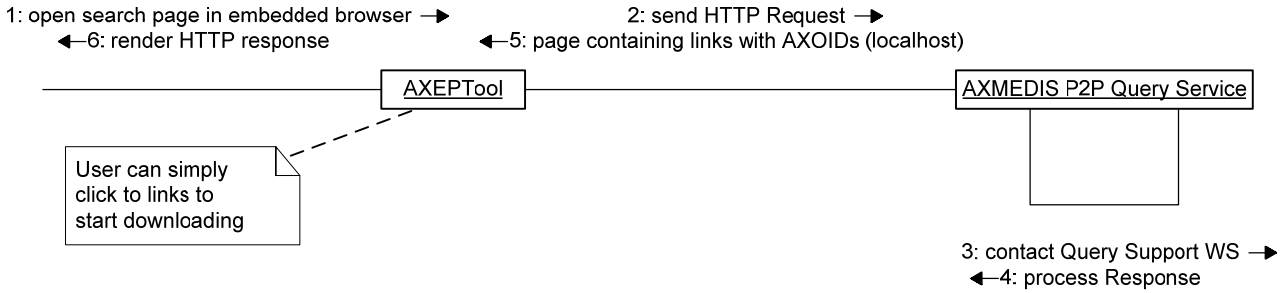
- The user can query for content directly on the client interface (even embedding a web-page rendering);
- The user can command to initiate the download process for a given query result; this use case include the typical action of monitoring and controlling the active downloading process.



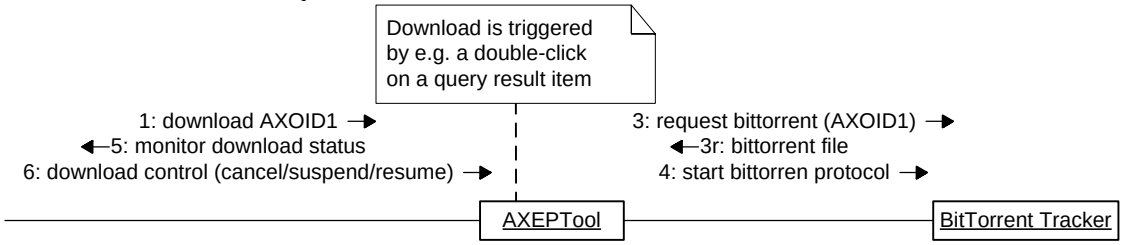
The other use cases are reported for two main reasons:

- The capability of controlling the AXEPTool (by a well-known WS interface) remotely. This action is particularly needed in B2B context since download processes can be managed in an automatic manner. This highlights that the downloading technology is reused, while it is accessed by a different way w.r.t. to the GUI.
- The role of the AXCP in completing the full P2P distribution life-cycle. A B2B user can actually create procedures for publish/unpublishing AXMEDIS objects by using scripting language. It can also "script" (i.e. program) what to import from the downloaded objects in the factory database for further usage.

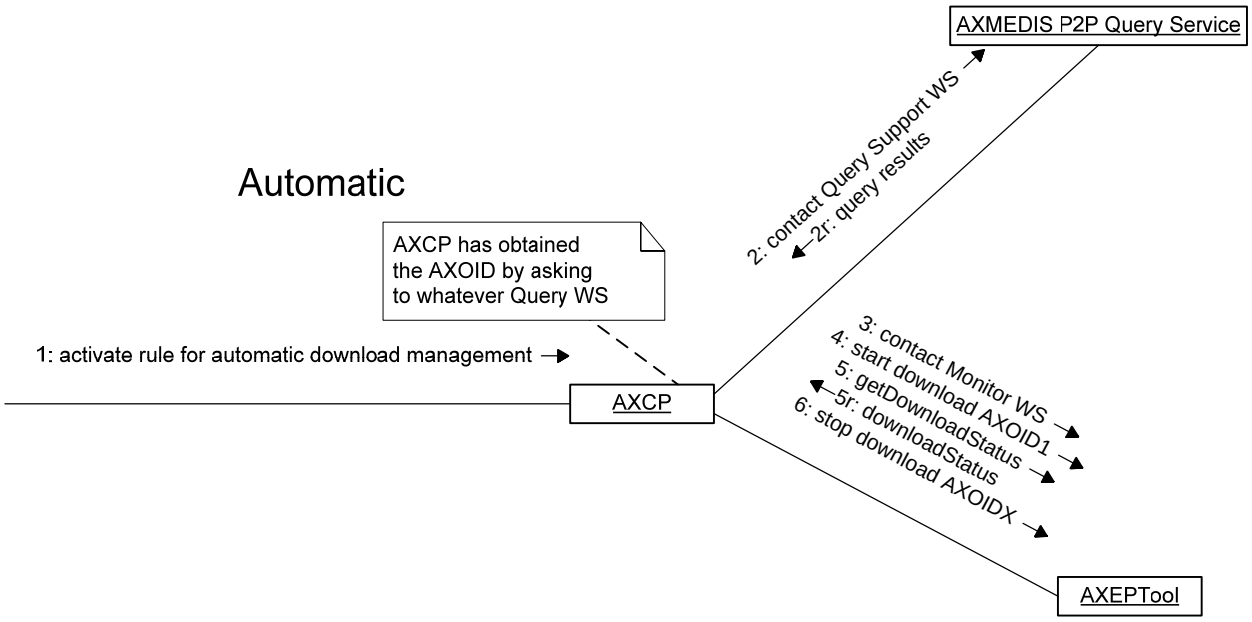
In the following basic collaboration diagram are reported in order to show AXEPTool P2P client functionalities for query and download.



The first diagram shows the interaction between AXEPTool and AXMEDIS P2P Query Service. The action is simply to redirect the user query to the Query Service collecting in the response information regarding the matching objects. The WS response is processed, since all the present metadata have to be attached to the query result items.
Please note that in the response also bittorrent metainfo files are returned.



In the second diagram the typical action, being consequent to a query, has been presented.
In fact the user can decide of starting the download of an object. In this case the interaction is based on bittorrent protocol established among AXEPTool and the interested BitTorrent Tracker.

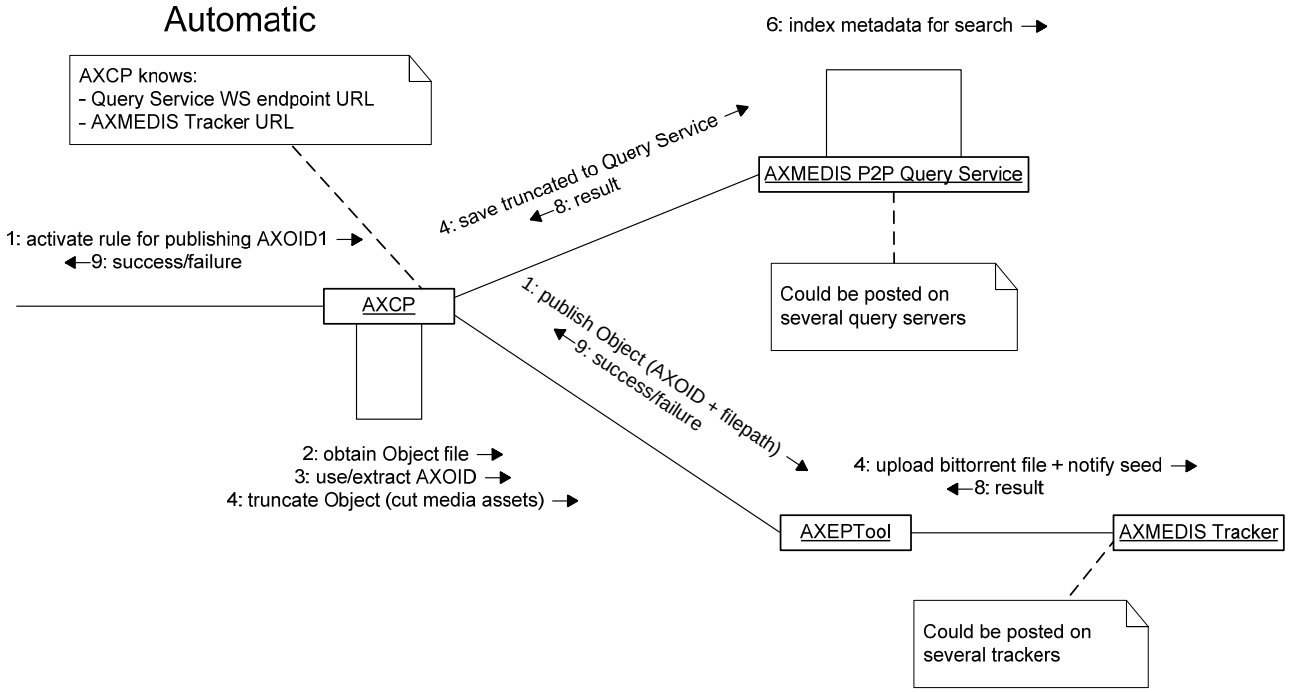


In the last diagram the AXCP is put as an intermediate in the AXEPTool control. In this case it allows the script function to operate as the user on the GUI (i.e. looking what is downloading and controlling those processes).

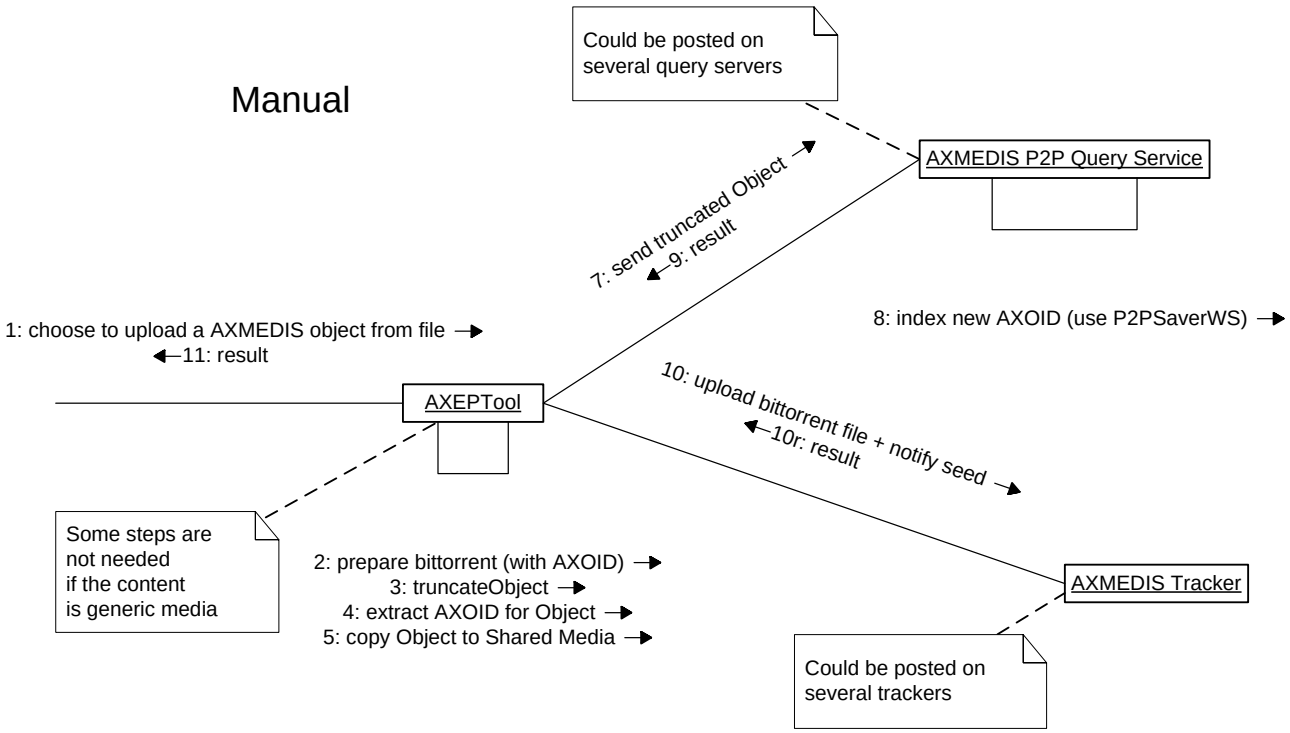
9.6.1 Usage walk-through for AXMEDIS Query Service and AXCP (EXITECH)

In this diagram a possible solution for avoiding posting of metadata by the B2B user is presented.
This solution can reuse the knowledge of the AXMEDIS infrastructure regarding an object metadata, since any object that can be distributed/published have been registered to AXCS (with metadata).

The solution of using the information located into the AXCS can be useful especially if the AXCS and the AXMEDIS P2P Query Service are located and maintained by the same organization. That can be AXMEDIS.ORG or by the channel distributor.

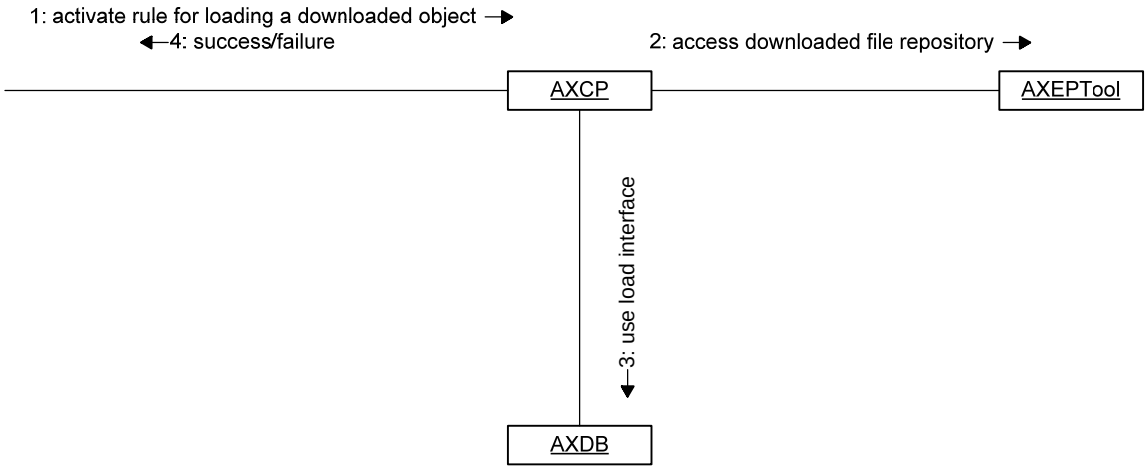


Please note that publishing task is totally carried out by AXCP and its functionalities of accessing WS. AXCP has also to compute bittorrent file.



In the manual version some content manipulations which has been performed by AXCP, has to be done by AXEPTool, the AXOID has to be extracted for augmenting bittorrent, and the AXMT file (truncated object) has to be posted for future queries.

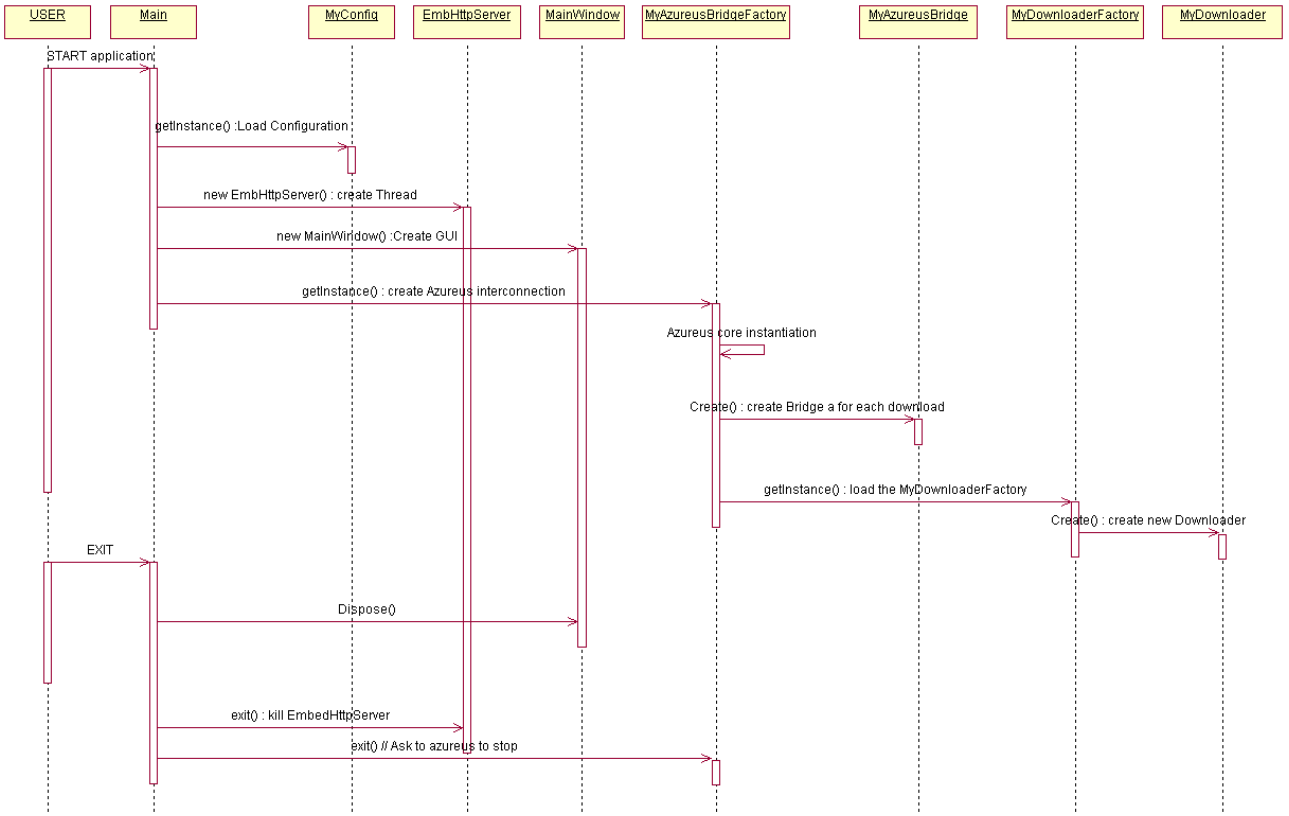
Another diagram has been produced to explain how import of newly downloaded object can be realized by only using AXCP functionality.
The latter has to access downloaded file repository of AXEPTool, and to use AXDB loading service.



9.7 Communication between AXEPTool and Azureus

9.7.1 Initialization and download control

The following diagram explains the whole initialization of the AXEPTool, the especially important part is the Azureus stack initialization. Other initializers (including the Embedded HTTP Server) are here to indicate the launch sequence. This sequence diagram and subsequent explanations also explain most of the communication structure between AXEPTool and Azureus.



Our main goal was to separate the Peer to Peer engine from the GUI as we want :

- To integrate quickly new releases of Azureus : This is important because those new release may contain important bugfixes or security fixes that need to be applied without delay. With the proposed design, integrating a new version of Azureus even if it contains major changes can be done quickly.
- Change the Peer To Peer engine to something else than Azureus. Even if BitTorrent is currently the best suited Peer To Peer protocol available for Axmedis, Peer To Peer is a relatively new technology. As a result, new protocols or engines may appear and offer drastic improvements. In order to protect the investment in the AXMEDIS project, we need to be able to change the engine without having everything to re-implement. There is also the case where Azureus could become unmaintained by its authors. We could then decide to switch to another engine quickly.

Application exit is also provided in the diagram as it is quite particular. The Azureus engine needs to be explicitly and properly killed. Just killing the process will NOT do the job : A java security manager is started by Azureus and prevent killing the process without properly shutting down Azureus. Moreover, starting a new AXEPTool process if Azureus has not been killed properly would result in a dysfunctional AXEPTool (this is currently detected at startup and the AXEPTool would exit cleanly if it happens).

The integrated web server also needs to be exited cleanly so that the port on which it is bound can be freed. Killing the process the hard way would leave a lingering bound socket, making it impossible to start a new AXEPTool for a few minutes. If despite all our precautions, this case happens, it is correctly detected on startup of the application, so that it can exit cleanly with an error message.

The whole application closure process can take a long time (approximately 10 seconds) as a result, we close the window first (so that the user can have a good feedback when he clicks on the window close widget) and do all the actions in the background right after.

The main class is responsible for calling the initializers. It first creates an instance of MyConfig a singleton which is the configuration file loader and parser. After instantiation, MyConfig will contain all the AXEPTool's configuration information.

Then the embedded HTTP server, OOWeb is loaded in a separate thread and the main servlet is bound to it.

After this, a separate thread is created and the GUI is launched inside. Lastly, the MyAzureusBridgeFactory class will be instantiated for the first time.

This class allows creating an interface between Azureus and the AXEPTool. It will first instantiate Azureus core and will then get the list of all pending download inside Azureus. For each of those downloads, it will use the factory design pattern to create one MyAzureusBridge for each download.

Those MyAzureusBridgeFactory objects will allow AXEPTool to query the state or act on all of those downloads. (Stopping a download, starting it again or getting download statistics).

Before finishing its construction, the factory will instantiate for the first time the singleton MyDownloaderFactory. This class use an Interface Design Pattern between the GUI of AXEPTool and downloads of AXEPTool. It's the link between the model and the view of the Model View Controller Pattern.

MyDownloaderFactory will then create, following a Factory design pattern a vector of MyDownloader, each linked with its MyAzureusBridge.

This double interface allows to :

- Change completely the Peer To Peer engine we use. We are currently using Azureus, but we could use another engine without rewriting the AXEPTool.

- As Azureus is a moving target which we do not control fully, it is important to be able to adapt to its changes quickly. Integrating with a new version of Azureus should not require changes above the level of those interfaces.
- Use Peer To Peer from Java easily so we can change the GUI easily in the future.

The factory principle is used to retrieve information on a collection of objects and to use static methods linked to all of these objects.

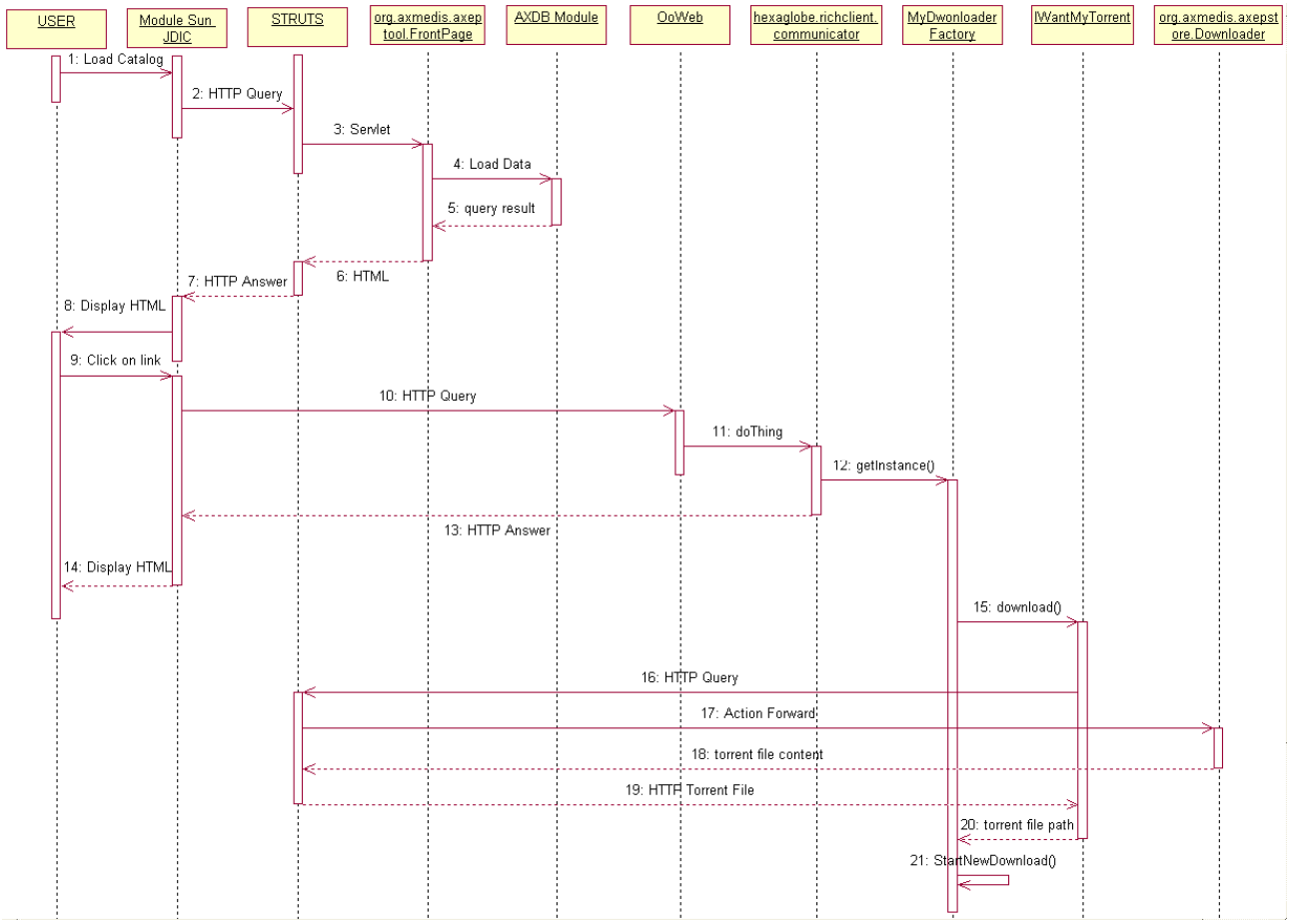
9.7.2 Monitoring download status

As seen precedently the main class of the GUI is MainWindow which creates each tabs of the main panel, CenterTabs. One of the Tabs implemented in the TabDownloads class lists all the pending downloads. Implementing tabs within Swing require the data of the tabs to be stored in an AbstractTableModel here called DownloadsTableModel. This data manager instantiate a thread named TabDownloadsUpdateThread which polls regularly all pending downloads. We have chosen to do the polling every seconds, which seems to be a good tradeoff between CPU utilization and application responsiveness. To poll the data, the factory for MyDownloader (MyDownloaderFactory class) is used. This factory lists all the pending downloads producing a vector of MyDownloader objects. All these objects are then queried thanks to the getDLRate and getUPRate methods for the download and upload transfer rate respectively. Each of the MyDownloader objects will query his corresponding MyAzureusBridge object for the transfer rate. The MyAzureusBridge will in turn ask Azureus which will provide the value.

A similar mechanism is used to get the file size actually downloaded, by using the getTotalSize method. Once again, this mechanism allows us to change the peer to peer engine quickly and keep the existing GUI.

9.8 Communication with the Tracker

The following diagram shows the sequence of events occurring in a typical user interaction.



This involves most parts of the software, so it's a good introduction to the general architecture of the whole Axmedis P2P infrastructure. In this scenario, the user first browses the catalog of available media files from the FrontPage servlet (which gets it from AxDB). The result is then displayed in the integrated web browser.

The following part (starting the download) requires a user action (selecting the file from the catalog). The click starts another HTTP query but this time not to the tracker but to the integrated web server of the client itself. The doThing method parses the parameters of the request and routes the call to the appropriate handler function. Then the MyDownloaderFactory is retrieved via getInstance() and user to start a new Download giving it the address of the metadata file. It will take care of downloading the metadata file by itself. The MyDownloaderFactory can be considered as the interface between the AXEPTool application and the azureus bittorrent stack.

The OOWeb will return to the JDIC browser an HTML page containing a HTTP redirection page. The redirection page returns the user to the catalog and the process can start again.

9.9 Technical and Installation information

Building the application is done with standard ANT build files. As installation could be quite complex, we provided a NSIS (freeware installer program for Windows) build files. Once you build the application, you can run the NSIS (follow instructions in README files) and it'll provide you an EXE file which is a windows installer. The resulting installer can be distributed freely.

Installation using the installer is automatic. It installs the main JAR file of AXEPTool and all the dependencies (located in the 3rdParty dependencies) whether they are platform independent (JAR files) or platform dependent (JDIC binaries provided for the integrated browser).

Here is a few technical information about the process (which may be useful if you want to install by a hand on an unsupported platform) :
Platform dependent files are currently provided for Windows and Linux. Running on other platform will require building JDIC for this platform. The platform dependent tools should be installed in the system default binary directory and default library directory. Otherwise, please be sure to update your search paths so that the libraries and executables will be found on runtime.

If you are running the application on a Unix or Linux system, you must export the MOZILLA_FIVE_HOME environment variable so that it points to a working Mozilla installation. After it you should be done with installation.

Once the installation is complete, you can start the application. It should connect to the tracker and display pages inside the integrated web browser. If it does not, the installation was not done properly.

You can troubleshoot the integrated web browser by setting a debug flag to “true”. This is normally not needed but may be useful especially if you are trying to build the AXEPTool on an unsupported platform. The debug flag is located inside fr.axmedis.ft.ihm.tabsmanager;. This turns the application as very verbose, so launching it under eclipse should provide you enough information on the debug console to figure out what is wrong.

If you encounter problems on startup saying that the internal web server (OOWEB) could not bind, please ensure that you do not have any application bound on the port that internal web server uses. If it is the case, you should stop that application.

References to other major components needed	AXTrack
Problems not solved	
Configuration and execution context	

9.10 Draft User Manual

Installation:
The setup.exe will install AXEPTool on your computer. Only Windows can use this installer.
Linux must use another package.
During installation the installer will ask you to enter a Tracker URL, if no information are given to you let the default value. (http://axtrk.axmedis.org:8080/AXTrackv2)

You must install java, jre 1.5 or more, if you want to use AXEPTool.
http://www.java.com

First use:
This Software has for aim the propagation of documents by peer to peer, so if you download you will have to send data.
If you got a firewall you must allowed that AXEPTool to access the web.
If you want to share files you must stay connected to the internet while anyone share this file too and stay connected.

Advanced:
You can use AXEPTool on a local network and not connected to the internet if the tracker is on this network and if all users are connected to it.

9.11 Examples of usage

You need to share a new video, from a conference to all absent, but you can't store it.
Connect you to the Internet.
Install AXEPTool if it's not already done.

Then Chose from File menu “Create New Document”.
Here you will get a File prompt, select your video.
Wait while the software let you do something (more bigger the file is, more longer you'll have to wait)
Then go to the Download tab, you will see a new download with percentage at one hundred.

Now you can tell to all absent to start an AXEPTool, and go to the Catalog.
Here they click on the File you have added, and then they wait for downloading the file.

Don't turn off your AXEPTool if you want the absent download from your connection.

Verification can be done by looking the upload rate on the download tab, if someone is downloading the file, your upload rate will be more than 0.

9.12 Configuration Parameters

Config parameter	Possible values
`downloadDirectory`	Set the directory where the file will be downloaded.
`homePage`	The home page : default is `http://www.axmedis.org`
`catalogueURL`	The url of the catalog, default : `http://axtrk.axmedis.org:8080/AXTrack/do/AXEPStore`
`TrackerURL`	`The url of the tracker Default : http://axtrk.axmedis.org:8080/AXTrack`

9.13 Errors reported and that may occur

Error code	Description and rationales
Error Azureus core not instanced	An other copy of axeptool is already in your process

`Exception in thread "EventThread" java.lang.NullPointerException`
`at org.jdesktop.jdic.browser.internal.MsgClient.<init>(Unknown Source)`
`at org.jdesktop.jdic.browser.internal.NativeEventThread.run(Unknown Source)`
	The Embed browser is crashed.
	Please restart the application.
	If it doesn't work, kill process IeEmbed or MozEmbed

9.14 Formal description of protocol Bittorrent Peer Wire

9.14.1 Overview

The peer protocol facilitates the exchange of pieces as described in the 'metainfo file.

Note here that the original specification also used the term "piece" when describing the peer protocol, but as a different term than "piece" in the metainfo file. For that reason, the term "block" will be used in this specification to describe the data that is exchanged between peers over the wire.

A client must maintain state information for each connection that it has with a remote peer: ' choked*: Whether or not the remote peer has choked this client. When a peer chokes the client, it is a notification that no requests will be answered until the client is unchoked. The client should not attempt to send requests for blocks, and it should consider all pending (unanswered) requests to be discarded by the remote peer. ' interested*: Whether or not the remote peer is interested in something this client has to offer. This is a notification that the remote peer will begin requesting blocks when the client unchokes them.

Note that this also implies that the client will also need to keep track of whether or not it is interested in the remote peer, and if it has the remote peer choked or unchoked. So, the real list looks something like this:

- **am_choking**: this client is choking the peer
- **am_interested**: this client is interested in the peer
- **peer_choking**: peer is choking this client
- ***peer_interested:*** *peer is interested in this client*

Client connections start out as "choked" and "not interested". In other words:

- **am_choking** = 1
- **am_interested** = 0
- **peer_choking** = 1
- **peer_interested** = 0

A block is downloaded by the client when the client is interested in a peer, and that peer is not choking the client. A block is uploaded by a client when the client is not choking a peer, and that peer is interested in the client.

It is important for the client to keep its peers informed as to whether or not it is interested in them. This state information should be kept up-to-date with each peer even when the client is choked. This will allow peers to know if the client will begin downloading when it is unchoked (and vice-versa).

9.14.2 Data Types

Unless specified otherwise, all integers in the peer wire protocol are encoded as four byte big-endian values. This includes the length prefix on all messages that come after the handshake.

9.14.3 Message flow

The peer wire protocol consists of an initial handshake. After that, peers communicate via an exchange of length-prefixed messages. The length-prefix is an integer as described above.

9.14.4 Handshake

The handshake is a required message and must be the first message transmitted by the client.

handshake: <pstrlen><pstr><reserved><info_hash><peer_id>

- **pstrlen**: string length of <pstr>, as a single raw byte
- **pstr**: string identifier of the protocol
- **reserved**: eight (8) reserved bytes. All current implementations use all zeroes. Each bit in these bytes can be used to change the behavior of the protocol. An email from Bram suggests that trailing bits should be used first, so that leading bits may be used to change the meaning of trailing bits.
- **info_hash**: 20-byte SHA1 hash of the info key in the metainfo file. This is the same info_hash that is transmitted in tracker requests.
- **peer_id**: 20-byte string used as a unique ID for the client. This is the same peer_id that is transmitted in tracker requests.

In version 1.0 of the BitTorrent protocol, pstrlen = 19, and pstr = "BitTorrent protocol".

The initiator of a connection is expected to transmit their handshake immediately. The recipient may wait for the initiator's handshake, if it is capable of serving multiple torrents simultaneously (torrents are uniquely identified by their infohash). However, the recipient must respond as soon as it sees the info_hash part of the handshake. The tracker's NAT-checking feature does not send the peer_id field of the handshake.

If a client receives a handshake with an info_hash that it is not currently serving, then the client must drop the connection.

If the initiator of the connection receives a handshake in which the peer_id does not match the expected peerid, then the initiator is expected to drop the connection. Note that the initiator presumably received the peer information from the tracker, which includes the peer_id that was registered by the peer. The peer_id from the tracker and in the handshake are expected to match.

9.14.4.1 peer_id

There are mainly two conventions how to encode client and client version information into the peer_id, Azureus-style and Shadow's-style.

Azureus-style uses the following encoding: '-', two characters for client id, four ascii digits for version number, '-', followed by random numbers.

For example: '-AZ2060-'...

known clients that use this encoding style are:

- 'AR' - *Arctic*
- 'AX' - *BitPump*
- 'AZ' - *Azureus*
- 'BB' - *BitBuddy*
- 'BC' - *BitComet*
- 'BF' - *Bitflu*
- 'BR' - *BitRocket*
- 'BS' - *BTSlave*
- 'BX' - ~Bittorrent X
- 'CD' - *Enhanced CTorrent*
- 'CT' - *CTorrent*
- 'DE' - *DelugeTorrent*
- 'EB' - *EBit*
- 'ES' - *electric sheep*
- 'HL' - *Halite*
- 'KT' - *KTorrent*
- 'LP' - *Lphant*
- 'LT' - *libtorrent*
- 'lt' - *libTorrent*
- 'MP' - *MooPolice*
- 'MT' - *MoonlightTorrent*
- 'qB' - *qBittorrent*
- 'QT' - Qt 4 Torrent example
- 'RT' - *Retriever*
- 'SB' - ~Swiftbit
- 'SS' - SwarmScope
- 'SZ' - *Shareaza*
- 'TN' - TorrentDotNET
- 'TR' - *Transmission*
- 'TS' - *Torrentstorm*
- 'UL' - uLeecher!
- 'UT' - *µTorrent*
- 'XT' - *XanTorrent*
- 'ZT' - *ZipTorrent*

Shadow's style uses the following encoding: one ascii alphanumeric for client identification, three ascii digits for version number, '----', followed by random numbers.

For example: 'S587----'...

known clients that uses this encoding style are:

- 'A' - *ABC*

- 'O' - *Osprey Permaseed*
- 'R' - *Tribler*
- 'S' - *Shadow's client*
- 'T' - *BitTornado*
- 'U' - *UPnP NAT Bit Torrent*

Bram's client now uses this style... 'M3-4-2--' or 'M4-20-8-'.

BitComet does something different still. Its peer_id consists of four ASCII characters 'exbc', followed by two bytes x and y, followed by random characters. The version number is x in decimal before the decimal point and y as two decimal digits after the decimal point. *BitLord* uses the same scheme, but adds 'LORD' after the version bytes. An *unofficial patch* for BitComet once replaced 'exbc' with 'FUTB'. The encoding for BitComet Peer IDs changed to Azureus-style as of BitComet version 0.59.

XBT Client has its own style too. Its peer_id consists of the three uppercase characters 'XBT' followed by three ASCII digits representing the version number. If the client is a debug build, the seventh byte is the lowercase character 'd', otherwise it is a '-'. Following that is a '-' then random digits, uppercase and lowercase letters. Example: 'XBT054d-' at the beginning would indicate a debug build of version 0.5.4.

Opera 8 previews use the following peer_id scheme: The first two characters are 'OP' and the next four digits equal the build number. All following characters are random lowercase hexdecimal digits.

MLdonkey use the following peer_id scheme: the first characters are '-ML' followed by a dotted version then a '-' followed by randomness. e.g. '-ML2.7.2-kgjjfkd'

Bits on Wheels uses the pattern '-BOWAxx-yyyyyyyyyyyy', where y is random (uppercase letters) and x depends on the version. Version 1.0.6 has xx = 0C.

Queen Bee uses Bram's new style: 'Q1-0-0--' or 'Q1-10-0-' followed by random bytes.

Many clients are using all random numbers or 12 zeroes followed by random numbers (like older versions of *Bram's client*).

9.14.5 Messages

All of the remaining messages in the protocol take the form of <length prefix><message ID><payload>. The length prefix is a four byte big-endian value. The message ID is a single decimal character. The payload is message dependent.

9.14.5.1 keep-alive: <len=0000>

The **keep-alive** message is a message with zero bytes, specified with the length prefix set to zero. There is no message ID and no payload. Peers may close a connection if they receive no messages (**keep-alive** or any other message) for a certain period of time, so a keep-alive message must be sent to maintain the connection alive if no command have been sent for a given amount of time. This amount of time is generally two minutes. The **keep-alive** message is a message with zero bytes, specified with the length prefix set to zero. There is no message ID and no payload. Peers may close

a connection if they receive no messages (**keep-alive** or any other message) for a certain period of time, so a keep-alive message must be sent to maintain the connection alive if no command have been sent for a given amount of time. This amount of time is generally two minutes.

9.14.5.2 choke: <len=0001><id=0>

The **choke** message is fixed-length and has no payload.

9.14.5.3 unchoke: <len=0001><id=1>

The **unchoke** message is fixed-length and has no payload.

9.14.5.4 interested: <len=0001><id=2>

The **interested** message is fixed-length and has no payload.

9.14.5.5 not interested: <len=0001><id=3>

The **not interested** message is fixed-length and has no payload.

9.14.5.6 have: <len=0005><id=4><piece index>

The **have** message is fixed length. The payload is the zero-based index of a piece that has just been successfully downloaded and verified via the hash.

Implementer's Note: That is the strict definition, in reality some games may be played. In particular because peers are extremely unlikely to download pieces that they already have, a peer may choose not to advertise having a piece to a peer that already has that piece. At a minimum "HAVE supression" will result in a 50% reduction in the number of HAVE messages, this translates to around a 25-35% reduction in protocol overhead. At the same time, it may be worthwhile to send a HAVE message to a peer that has that piece already since it will be useful in determining which piece is rare.

A malicious peer might also choose to advertise having pieces that it knows the peer will never download. Due to this attempting to model peers using this information is a **bad idea**.

9.14.5.7 bitfield: <len=0001+X><id=5><bitfield>

The **bitfield** message may only be sent immediately after the handshaking sequence is completed, and before any other messages are sent. It is optional, and need not be sent if a client has no pieces.

The **bitfield** message is variable length, where X is the length of the bitfield. The payload is a bitfield representing the pieces that have been successfully downloaded. The high bit in the first byte corresponds to piece index 0. Bits that are cleared indicated a missing piece, and set bits indicate a valid and available piece. Spare bits at the end are set to zero.

A bitfield of the wrong length is considered an error. Clients should drop the connection if they receive bitfields that are not of the correct size, or if the bitfield has any of the spare bits set.

9.14.5.8 request: <len=0013><id=6><index><begin><length>

The **request** message is fixed length, and is used to request a block. The payload contains the following information:

- **index**: integer specifying the zero-based piece index
- **begin**: integer specifying the zero-based byte offset within the piece
- **length**: integer specifying the requested length.

View #1 According to the official specification, "All current implementations use 2^15 (32KB), and close connections which request an amount greater than 2^17 (128KB)." As early as version 3 or 2004, this behavior was changed to use 2^14 (16KB) blocks. As of version 4.0 or mid-2005, the mainline disconnected on requests larger than 2^14 (16KB); and some clients have followed suit. Note that block requests are smaller than pieces (>=2^18 bytes), so multiple requests will be needed to download a whole piece.

Strictly, the specification allows 2^15 (32KB) requests. The reality is near all clients will now use 2^14 (16KB) requests. Due to clients that enforce that size, it is recommended that implementations make requests of that size. Due to smaller requests resulting in higher overhead due to tracking a greater number of requests, implementers are advised against going below 2^14 (16KB).

The choice of request block size limit enforcement is not nearly so clear cut. With mainline version 4 enforcing 16KB requests, most clients will use that size. At the same time 2^14 (16KB) is the semi-official (only semi because the official protocol document has not been updated) limit now, so enforcing that isn't wrong. At the same time, allowing larger requests enlarges the set of possible peers, and except on very low bandwidth connections (<256kbps) multiple blocks will be downloaded in one choke-timeperiod, thus merely enforcing the old limit causes minimal performance degradation. Due to this factor, it is recommended that only the older 2^17 (128KB) maximum size limit be enforced.

View #2 This section has contained falsehoods for a large portion of the time this page has existed. This is the third time I (uau) am correcting this same section for incorrect information being added, so I won't rewrite it completely since it'll probably be broken again... Current version has at least the following errors: Mainline started using 2^14 (16384) byte requests when it was still the only client in existence; only the "official specification" still talked about the obsolete 32768 byte value which was in reality neither the default size nor maximum allowed. In version 4 the request behavior did not change, but the maximum allowed size did change to equal the default size. In latest mainline versions the max has changed to 32768 (note that this is the first appearance of 32768 for either default or max size since the first ancient versions). "Most older clients use 32KB requests" is false. Discussion of larger requests fails to take latency effects into account.

piece: <len=0009+X><id=7><index><begin><block>

The **piece** message is variable length, where X is the length of the block. The payload contains the following information:

- **index**: integer specifying the zero-based piece index
- **begin**: integer specifying the zero-based byte offset within the piece
- **block**: block of data, which is a subset of the piece specified by index.

9.14.5.9 cancel: <len=0013><id<=8><index><begin><length>

The **cancel** message is fixed length, and is used to cancel block requests. The payload is identical to that of the "request" message. It is typically used during "End Game" (see the Algorithms section below).

9.14.5.10 port: <len=0003><id=9><listen-port>

The **port** message is sent by newer versions of the Mainline that implements a DHT tracker. The listen port is the port this peer's DHT node is listening on. This peer should be inserted in the local routing table (if DHT tracker is supported).

9.15 Algorithms

9.15.1 Queuing

View #1 : In general peers are advised to keep a few unfullfilled requests on each connection. This is done because otherwise a full round trip is required from the download of one block to begining the download of a new block (round trip between PIECE message and next REQUEST message). On links with high BDP (bandwidth-delay-product, high latency or high bandwidth), this can result in a substantial performance loss.

Implementer's note: This the 'most crucial performance item. A static queue of 10 requests is reasonable for 16KB blocks on a 5mbps link with 50ms latency. Links with greater bandwidth are becoming very common so UI designers are urged to make this readily available for changing. Notably cable modems were known for traffic policing and increasing this might of aleviated some of the problems caused by this.

9.15.2 Super Seeding

(This was not part of the original specification)

The super-seed feature in S-5.5 and on is a new seeding algorithm designed to help a torrent initiator with limited bandwidth "pump up" a large torrent, reducing the amount of data it needs to upload in order to spawn new seeds in the torrent.

When a seeding client enters "super-seed mode", it will not act as a standard seed, but masquerades as a normal client with no data. As clients connect, it will then inform them that it received a piece -- a piece that was never sent, or if all pieces were already sent, is very rare. This will induce the client to attempt to download only that piece.

When the client has finished downloading the piece, the seed will not inform it of any other pieces until it has seen the piece it had sent previously present on at least one other client. Until then, the client will not have access to any of the other pieces of the seed, and therefore will not waste the seed's bandwidth.

This method has resulted in much higher seeding efficiencies, by both inducing peers into taking only the rarest data, reducing the amount of redundant data sent, and limiting the amount of data sent to peers which do not contribute to the swarm. Prior to this, a seed might have to upload 150% to 200% of the total size of a torrent before other clients became seeds. However, a large torrent seeded with a single client running in super-seed mode was able to do so after only uploading 105% of the data. This is 150-200% more efficient than when using a standard seed.

Super-seed mode is 'NOT recommended for general use. While it does assist in the wider distribution of rare data, because it limits the selection of pieces a client can downlad, it also limits the ability of those clients to download data for pieces they have already partially retrieved. Therefore, super-seed mode is only recommended for initial seeding servers.

9.15.3 Piece downloading strategy

Clients may choose to download pieces in random order.

A better strategy is to download pieces in rarest first order. The client can determine this by keeping the initial bitfield from each peer, and updating it with every 'have message. Then, the client can download the pieces that appear least frequently in these peer bitfields.

9.15.4 End Game

When a download is almost complete, there's a tendency for the last few blocks to trickle in slowly. To speed this up, the client sends requests for all of its missing blocks to all of its peers. To keep this from becoming horribly inefficient, the client also sends a cancel to everyone else every time a block arrives.

There is no documented thresholds, recommended percentages, or block counts that could be used as a guide or Recommended Best Practice here.

When to enter end game mode is an area of discussion. Some clients enter end game when all pieces have been requested. Others wait until the number of blocks left is lower than the number of blocks in transit, and no more than 20. There seems to be agreement that it's a good idea to keep the number of pending blocks low (1 or 2 blocks) to minimize the overhead, and if you randomize the blocks requested, there's a lower chance of downloading duplicates. More on the protocol overhead can be found here: http://hal.inria.fr/inria-00000156/en

9.15.5 Choking and Optimistic Unchoking

Choking is done for several reasons. TCP congestion control behaves very poorly when sending over many connections at once. Also, choking lets each peer use a tit-for-tat-ish algorithm to ensure that they get a consistent download rate.

The choking algorithm described below is the currently deployed one. It is very important that all new algorithms work well both in a network consisting entirely of themselves and in a network consisting mostly of this one.

There are several criteria a good choking algorithm should meet. It should cap the number of simultaneous uploads for good TCP performance. It should avoid choking and unchoking quickly, known as 'fibrillation'. It should reciprocate to peers who let it download. Finally, it should try out unused connections once in a while to find out if they might be better than the currently used ones, known as optimistic unchoking.

The currently deployed choking algorithm avoids fibrillation by only changing choked peers once every ten seconds.

Reciprocation and number of uploads capping is managed by unchoking the four peers which have the best upload rate and are interested. This maximizes the client's download rate. These four peers are referred to as downloaders, because they are interested in downloading from the client.

Peers which have a better upload rate (as compared to the downloaders) but aren't interested get unchoked. If they become interested, the downloader with the worst upload rate gets choked. If a client has a complete file, it uses its upload rate rather than its download rate to decide which peers to unchoke.
For optimistic unchoking, at any one time there is a single peer which is unchoked regardless of it's upload rate (if interested, it counts as one of the four allowed downloaders). Which peer is optimistically unchoked rotates every 30 seconds. Newly connected peers are three times as likely to start as the current optimistic unchoke as anywhere else in the rotation. This gives them a decent chance of getting a complete piece to upload.

9.15.5.1 Anti-snubbing

Occasionally a BitTorrent peer will be choked by all peers which it was formerly downloading from. In such cases it will usually continue to get poor download rates until the optimistic unchoke finds better peers. To mitigate this problem, when over a minute goes by without getting a single piece from a particular peer, BitTorrent assumes it is "snubbed" by that peer and doesn't upload to it except as an optimistic unchoke. This frequently results in more than one concurrent optimistic unchoke, (an exception to the exactly one optimistic unchoke rule mentioned above), which causes download rates to recover much more quickly when they falter.

10 Module or Executable Tool AXTrack

Responsible Name	Franck Coppola
Responsible Partner	DSI (contractor) Hexaglobe (sub)
Status (proposed/approved)	
Implemented/not implemented	Implemented
Status of the implementation	Work In Progress
Executable or Library/module (Support)	
Single Thread or Multithread	Multithreaded (inside Tomcat)
Language of Development	Java (with some Python dependencies for bittorrent legacy)
Platforms supported	Windows, Linux (should work on any POSIX system)
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/newrepos/Applications/bt_tracker/AXTrack
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.
Reference to the AXFW location of the demonstrator executable tool for public download	
Address for accessing to WebServices if any, add accession information (user aNd Passwd) if any	
Test cases (present/absent)	Absent
Test cases location	http://////////////////
Usage of the AXMEDIS configuration manager (yes/no)	No
Usage of the AXMEDIS Error Manager (yes/no)	No
Major Problems not solved	-- --
Major pending requirements	-- Web Services Implementation (Pending) -- Ease of installation deployment. -- Porting application to AXDB usage.

Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)

Formats Used	Shared with	format name or reference to a section

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

10.1 General Description of the Module

AXTrack is a full featured bittorrent tracker. It aims to become fully integrated inside Axmedis. Currently it is able to interact with the AXEPTool. Main features are :

- Track BitTorrent files
- Catalog of tracked files
- Track download statistics.
- Integration with AXDB
- Integration with AXEPTool/AXMEDIA

This application has been developed as a Tomcat/Struts application. The first release is not integrated with AXDB but integration is going to be done as a result of M3 milestone.

The Tomcat application features both the tracker implemented through a classical announce URL and a web interface which will usually be accessed through AXEPTool integrated web browser. The web interface allows tasks such as :

- Statistics consultation.
- Adding a tracked file
- Downloading a file.

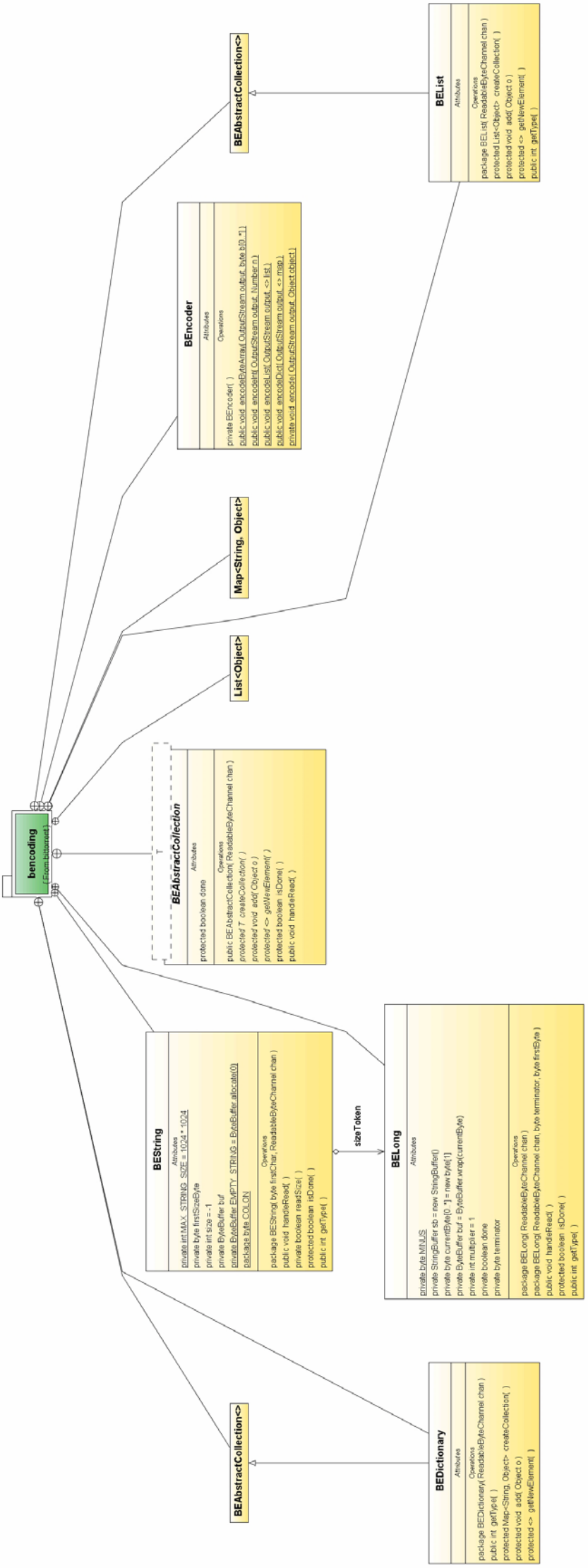
In order to increase the ease of use of the whole solution, the AXTrack is able to interact with a legacy BitTorrent client. This feature is currently only supported on UNIX and Linux systems. This is useful to seed documents from the server. As a result, an user can upload a file on the tracker and the tracker will both track

it and seed it. As the tracker server is more likely to be in a server room with suitable bandwidth access than the AXTrack this feature is useful for better productivity. It's usage is however optional. The user can use the AXTrack only as a tracker and seed the file from the AXEPTool. The tracker is also able to generate a torrent file on upload. This functionality is redundant to what's included in the AXEPTool and is here to provide ease of use to advanced users.

details about Tracker Administration pages:

- on/off: local copy of shared file in the tracker
- on/off: NON-AXMEDIS content sharing
- delete of uploaded media entry (could be a catalog view with checkbox at any item + a button "Delete selected")

details about Catalog web pages which shows AXIOD-Filename-infoHASH


bencoding
(From bittorrent)
BEAbstractCollection<>
BEList
Attributes
Operations
package BEList(ReadableByteChannel chan)
protected List<Object> createCollection( )
protected void add(Object o)
protected <> getNewElement( )
public int getType( )
BEncoder
Attributes
Operations
private BEncoder( )
public void encode(ByteArray OutputStream output, byte b[0..*])
public void encodeInt(OutputStream output, Number n)
public void encodeList(OutputStream output, <> list)
public void encodeDict(OutputStream output, <> map)
private void encode(OutputStream output, Object object)
Map<String, Object>
List<Object>
BEAbstractCollection
Attributes
protected boolean done
Operations
public BEAbstractCollection(ReadableByteChannel chan)
protected T createCollection( )
protected void add(Object o)
protected <> getNewElement( )
protected boolean isDone( )
public void handleRead( )
T
BEString
Attributes
private int MAX_STRING_SIZE = 1024 * 1024
private byte firstSizeByte
private int size = -1
private ByteBuffer buf
private ByteBuffer EMPTY_STRING = ByteBuffer.allocate(0)
package byte COLON
Operations
package BEString(byte firstChar, ReadableByteChannel chan)
public void handleRead( )
private boolean readSize( )
protected boolean isDone( )
public int getType( )
sizeToken
BELong
Attributes
private byte MINUS
private StringBuffer sb = new StringBuffer()
private byte currentByte[0..1] = new byte[1]
private ByteBuffer buf = ByteBuffer.wrap(currentByte)
private int multiplier = 1
private boolean done
private byte terminator
Operations
package BELong(ReadableByteChannel chan)
package BELong(ReadableByteChannel chan, byte terminator, byte firstByte)
public void handleRead( )
protected boolean isDone( )
public int getType( )
BEAbstractCollection<>
BEDictionary
Attributes
Operations
package BEDictionary(ReadableByteChannel chan)
public int getType( )
protected Map<String, Object> createCollection( )
protected void add(Object o)
protected <> getNewElement( )

10.2 Module Design in terms of Classes

10.2.1 The Bencoder

The Bencoder is the central place of the tracker. Most of the bittorrent infrastructure including the peer wire protocol, the tracker protocol and the metainfo files are encoded in the Bencode format. The bencoder modules serves as a Marshaller / Unmarshaller for Bencoded data, allowing us to us them in the program. The structure is quite simple as Bencoding uses no complex structures : Only lists and dictionaries.

Other code modules are very simple and do not express any complex layout. Most complexity in this code is due to the fact that implementing BitTorrent protocol requires bitwise operations. As java was designed to be a cross platform language, bitwise operators are typically difficult to use and not recommanded in order to avoid endianness problems.

Please keep in mind that even if bitwise operations are considered a bad thing in terms of software engineering, peer to peer is a bad thing on a telecom operator point of view as it is a huge consummer of bandwidth. The main risk for a P2P protocol designer is that the ISPs may decide to put restrictions on a particular protocol, especially if it is so resource hungry that routers and/or main lines become saturated. This already happens if France in parts of the territory (rural). As a result, designers of BitTorrent did all they could to implement a very compact protocol. The unfriendliness of the protocol for software engineers is the price to pay.

You can have an overview of the Bencoder design in terms of classes in the diagram located at the beginning of this chapter. The modules reads and/or write into a byte array which can then be integrated directly into BitTorrent's protocol stream. (or read from a protocol stream).

10.2.2 Identification of tracked file

All tracked files are identified thanks to their info_hash. This is a protocol requirement. The info hash is a SHA1 hash of the info key from the metainfo file. The info key is a Bencoded dictionary containing a list of all the pieces constituting the file. This is why the info_hash constitute a good unique identifier for files on the bittorrent network.

On the tracker side, decoding the info field is normally not necessary, but the following operation on info_hashes are necessary :

- Compute the info_hash from a torrent metainformation file.
- Retrieve the info_hash from tracker requests.

The last point is a bit tricky as requests are received as binary parameters of the GET request (not ascii string but escaped binary). The struts framework will by default convert this to a String, while taking the charset into account during unescaping. This obviously completely breaks the transmitted info_hash.

As a result, we must get the QueryString directly and do all the urldecode job by hand.

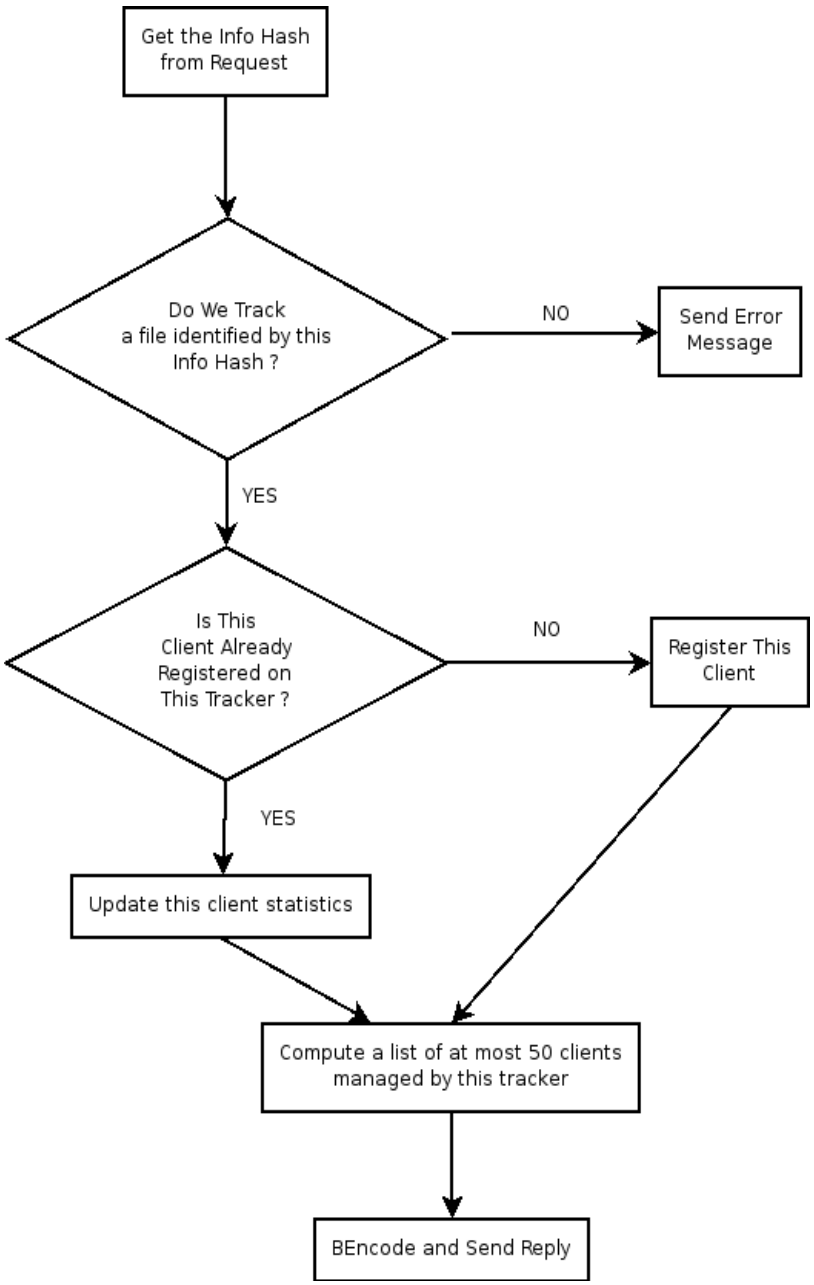
10.2.3 Tracker announce

If we care only about the “tracker” part, most of it is contained in the org.hexaglobe.bittorrent.tracker module and more closely in the AnnounceAction class which is bound to a struts URL. The struts URL is what is written in the file's metadata. When a Bittorrent client gets the metadata it then knows the tracker announce URL and will forge GET requests to it respecting the Bittorrent tracker protocol which is very simple.

The main actions performed by the Announce request :

- Get the info_hash from the query string, convert it to ASCII.
- Query the database, to see if we track this info_hash
- If yes, compute a list of peers to return and return them to the client (bencoded).

The **doTrack** method of this object is where you must look if you want to understand it at high level. Other methods typically performs low level / system tasks. The following diagram explains the algorithm :



10.2.4 The AXEPStore (Web interface for the client browsing)

One of the important part of the AXTrack is it's communication capabilities with the AXEPTool. The systems function as a big torrent repository and provide a full list of the content it is aware of, in order to allow client selecting and starting download of content.

The integrated Web Browser of the AXEPTool connects to a servlet (in the org.hexaglobe.axepstore package) managed by the Apache Struts framework. For the sake of clarity, the internal details of integration with Apache Struts have been removed from this specification (they are standard).

The servlet provides the following services :

- Browsing the catalog of available files : This is the main store interface. It can be extended to provide a full featured online store (with payment and full marketing informations) . The catalog is generated from the AXDB with metainformations stored using Dublin Core. The only difference between this web interface and a regular interface is that part of the URLs it returns connect to localhost, expecting to find AXEPTool's integrated web server. As a result, connecting to this interface with a regular web browser will produce a non working broken site. Only the AXEPTool will use this ser vice.
- Downloading metainformation files : This is only a catalog of torrent files. After checking the rights of a particular client to download the files it request, it'll allow the downloading of a torrent metainfo file. Technically speaking this is a servlet. On a functional point of view, the user just requests an URL and download a torrent file as a result. This request is done by the Azureus stack. Once Azureus sucessfully downloaded the metainformation file, it starts the download.

10.2.5 Integration with the AxDB.

The AXDB has been integrated to the BitTorrent tracker server. It will be extended to provide efficient support for Peer To Peer. The AXDB provide efficient storage for the axmedis objects.

For a specific description of AXDB please see :
axmedis-de3-1-2-2-9-spec-of-ax-database-and-query-support-v1-14.pdf

Concerning our part, we are going to use mainly the DCMI table in order to do search querys on metainformations. The DCMI table contains for each AXOID all relevent Dublin Core metadatas as specified by the Dublin Core Metadata Initiative working group.

We'll also use the p2phub database to make the link to bittorrent metadatas. The p2phub table provide the correspondance between AXOIDs and the URI of the metadatas on the bittorrent network.

This URI will be usable by the AXEPTool to retrieve the metadatas and start the download as explained in the AXEPTool section of this documentation.

As a result of direct use of the AXDB, we will integrate the QuerySupport Web Service directly into the tracker, allowing any client to query available files of a given tracker directly and start a download.

Web services interfaces expressed in WSDL are given later in this documentation.

10.3 User interface description

The AXTrack was not designed to be used directly. It exposes a web interface, but it was designed to be used through the AXEPTool. As a result, the description of the interface is done in the AXEPTool section.

As the system is currently under developpement we are allowing connexions from any web browser so ordinary clients are able to connect to the AXEPTool interface. In the future we will restrict that possibility so that only the AXEPTool or AXMedia will be able to connect and browse the web interface of the AxTrack.

This is because due to intensive use of page to browser communication (Liveconnect replacement), most of the pages are not functionnal outside the AXEPTool. As a result, we'll lock access to avoid mistakes.

10.4 Technical and Installation information

Installing the AXTrack is currently easy on a UNIX/Linux system. Installation of a minimal but working Windows AXTrack is also easy. Installing the full featured AXTrack on Windows is currently tricky. We hope to have it fixed upon the next release.

Before compiling the AXTrack please go in the default package and fix the configuration parameters. Some of them absolutely need to be tweaked if you want your installation to work.

Compiling the AXTrack should produce a WAR file containing a struts application. Compilation is easy and is using standard ANT build files. The WAR file should install seamlessly in a Tomcat server. If you have trouble installing the WAR in your Tomcat, please see the integration information chapter in this file.

However, the user should create two things on the system :

- A directory layout starting from the root of the server designed to contain the files. (TODO) . This directory layout must be readable and writable by the user owning the tomcat server.
- A working legacy bittorrent installation. This should be callable from the command line. This is used for the seeding system and for the metafile generation system. On Windows, you should be able to install from source the metafile generation system easily. However the btdownload program is tricky to install from the source on Windows. Windows binary installations provide only GUI bittorrent tools which are of course unsuitable for this use.

All other dependencies are included in the 3rdParty directory of the source files as jar files. As such they are included in the WAR files. All these dependencies are distributed under open-source OSI compliant licenses and should pose no problem for redistribution.

References to other major components needed	Tomcat Legacy Bittorrent
Problems not solved	• Windows Installation for seeding feature • AXDB Integration
Configuration and execution context	

10.5 Draft User Manual

Once installed, using the AXTrack is done through the AXEPTool. This is NOT a standalone component.

10.6 Examples of usage

To test the AxTrack and especially the tracker capabilities, do the following :

- Start an AXEPTool.
- Put a file for distribution on the Peer To Peer Network
- Start 4 AXEPTool on different systems and download that file after searching for it on the catalog.
- Then start 4 other AXEPTools on different systems and download again.
- You should be able to see that the download is spread between the file sources.

10.7 Integration and compilation issues

10.7.1 Databases

Compilation should not be problematic, due to the use of ANT files. We build the project on Netbeans 5.5 and it should build out of the box with Netbeans. You only need to grab the source from the SVN and import the project from the ANT file.

After building, you must integrate the project to your TOMCAT server. This should not be tricky. But please ensure that you have the following :

- A Working driver for your database system.
- A datasource entry in tomcat

As a matter of help here is the entry to add in the server.xml to connect to a database server. (Valid since Milestone 3, Milestone 2 does not need it).

```
        <Context path="/" debug="0" reloadable="true" displayName="DB Application">
      <Resource name="jdbc/listenerResource"
              scope="Shareable"
              type="javax.sql.DataSource"/>

        <ResourceParams name="jdbc/listenerResource">
          <parameter>
            <name>url</name>
            <value>jdbc:mysql://localhost:3306/axdbv3?jdbcCompliantTruncation=false</value>
          </parameter>
          <parameter>
            <name>username</name>
            <value>axdbuser</value>
          </parameter>
          <parameter>
            <name>password</name>
            <value>mkzamk</value>
          </parameter>
        </ResourceParams>

    </Context>
```

On Milestone 2, we use Hibernate as a database driver. Hibernate support may be dropped in the future (Milestone 3) if we decide to integrate the tracker database inside the AxDB. Or we may decide to separate the two databases. The M3 is currently a work in progress and we are evaluating our options regarding the integration.

In any case, if you have trouble with the integration, you may be advised to check Hibernate's configuration file. They are integrated into the WAR file. Please adapt them so that they correctly describe your database connection.

Like all configuration files they are located in the default package of the WAR file. As a courtesy, here is a samble file for Hibernate configuration.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
   <session-factory>
      <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
      <property name="hibernate.connection.url">
        jdbc:mysql://127.0.0.1:3306/hexatrack
      </property>
      <property name="show_sql">false</property>
      <property name="hibernate.connection.username">utente</property>
      <property name="hibernate.connection.password">ciccioformaggio</property>
      <property name="hibernate.connection.pool_size">10</property>
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
      <mapping resource="Peers.hbm.xml"/>
      <mapping resource="Torrents.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
```

You should NOT tweak the Peers.hbm or Torrents.hbm files unless you change the database schema.

10.7.2 Networking

The AXTrack has been specially designed to be used inside a network with a NAT gateway inside. As a result, it will provide files both inside out outside the network where it is installed unless configured to do otherwise.

To make it work properly on a NAT network, you must edit the configuration files located in the default package and fill the values according to your network parameters. It will then work automatically.

Please DO NOT forget to open the ports necessary for bittorrent operation on your firewall. If you do not do this, you'll have either bad performance or either no download at all. Please remember that the ports to open are given as a range. The more port you open in that range, the more simultaneous downloads you can serve. If you open too few ports, your downloading capacity will be severly restricted.

On the other side, opening a few ports m ay be a cheap way to do traffic throttling for people who do not want to invest in a Layer 3 switch.

10.7.3 Other problems

If something goes wrong, do NOT forget that you need to have a vanilla bittorrent installation on your system to enjoy the tracker full functionality. On linux, just use the package manager of your distribution to do the job. We tested it on Debian GNU/Linux with the standard debian shipped version of bittorrent. In case of oddities with your distribution especially with the command line which is sometime symlinked to hell, we provided externalized configuration so that you can tweak it easily. It should not be needed on most commonplace distributions.

So the next section list all the parameters you can tweak if you have problem with integration.

10.8 Configuration Parameters

Please note that you are going to have two sets of configuration parameters, one for linux and one for windows. This way we can have truly multiplatform WAR files which should work anywhere in the java spirit. If it does not, you can tweak them.

Config parameter	Possible values
axtrack.host	Hostname of the box where the tracker is installed.
axtrack.url	URI of the tracker inside that box.
axtrack.linux.path	The path of AXTrack file storage directory.
axtrack.linux.maketorrentcommand	Full path to the maketorrent command or executable name if the maketorrent command is in the PATH environment variable.
axtrack.windows.path	Full path to the AXTrack file storage directory on Windows
axtrack.windows.maketorrentcommand	Full path to the maketorrent command or executable name if the maketorrent command is in the PATH environment variable.
axtrack.externalipadress	Adresse IP externe du serveur
axtrack.localnetwork	Adresse IP locale du reseau
axtrack.localipadress	Adresse IP locale du serveur

11 Table description for database HexaTrack

Here is the table structure of the database HexaTrack.

```
 DROP TABLE IF EXISTS `ConfigurationAxtrack`;
CREATE TABLE `ConfigurationAxtrack` (
`Id` bigint(20) default NULL,
`AcceptAxmedis` varchar(10) default NULL
) DEFAULT CHARSET=latin1;

INSERT INTO `ConfigurationAxtrack` VALUES (1,'YES');

DROP TABLE IF EXISTS `Peers`;
CREATE TABLE `Peers` (
`Id` bigint(20) NOT NULL auto_increment,
`Idt` bigint(20) default NULL,
`PeerIp` bigint(20) default NULL,
`LastUpdate` bigint(20) default NULL,
`PeerPort` int(11) default NULL,
`Dl_sec` bigint(11) default NULL,
`Status` varchar(20) default NULL,
`Completed` tinyint(1) default NULL,
`StartingTime` bigint(20) default NULL,
PRIMARY KEY  (`Id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `Torrents`;
CREATE TABLE `Torrents` (
`Id` bigint(20) NOT NULL auto_increment,
`Info_Hash` varchar(80) default NULL,
`Active` tinyint(1) default NULL,
`Banned` tinyint(1) default NULL,
`Description` varchar(255) default NULL,
`NbDownload` bigint(20) default NULL,
`Axoid` varchar(80) default NULL,
`FileSize` bigint(20) default NULL,
`Protected` tinyint(1) default NULL,
PRIMARY KEY  (`Id`)
) DEFAULT CHARSET=latin1;
```

12 WSDL description of the Publishing and Monitoring Objects Interface

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://p2p.axmedis.org" xmlns:soapenc12="http://www.w3.org/2003/05/soap-encoding"
xmlns:tns="http://p2p.axmedis.org" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap11="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:soapenc11="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
- <wsdl:types>
- <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" attributeFormDefault="qualified"
elementFormDefault="qualified" targetNamespace="http://p2p.axmedis.org">
- <xsd:complexType name="AXOIDFileName">
- <xsd:sequence>
<xsd:element minOccurs="0" name="AXOID" nillable="true" type="xsd:string" />
<xsd:element minOccurs="0" name="localFileName" nillable="true" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
- <xsd:element name="getStatus">
- <xsd:complexType>
- <xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="selectedMedia" nillable="true" type="tns:AXOIDFileName" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
- <xsd:element name="getStatusResponse">
- <xsd:complexType>
- <xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
- <xsd:element name="removePublishedObject">
- <xsd:complexType>
- <xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="selectedMedia" nillable="true" type="tns:AXOIDFileName" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
- <xsd:element name="removePublishedObjectResponse">
- <xsd:complexType>
- <xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="out" type="xsd:int" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
- <xsd:element name="publishObject">
- <xsd:complexType>
- <xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="AXOID" nillable="true" type="xsd:string" />
<xsd:element maxOccurs="1" minOccurs="1" name="trackerURL" nillable="true" type="xsd:string" />
<xsd:element maxOccurs="1" minOccurs="1" name="localBitTorrent" nillable="true" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
- <xsd:element name="publishObjectResponse">
- <xsd:complexType>
- <xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="out" type="xsd:int" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
- <xsd:element name="downloadObjectURL">
- <xsd:complexType>
- <xsd:sequence>
<xsd:element maxOccurs="1" minOccurs="1" name="torrentURL" nillable="true" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

```
- <xsd:element name="downloadObjectURLResponse">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="out" type="xsd:int" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="getDownloadStatus">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="selectedMedia" nillable="true" type="tns:AXOIDFileName" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:complexType name="DownloadStatus">
- <xsd:sequence>
  <xsd:element minOccurs="0" name="downloadPercentage" type="xsd:int" />
  <xsd:element minOccurs="0" name="downloadedKBytes" type="xsd:long" />
  <xsd:element minOccurs="0" name="elapsed" type="xsd:long" />
  <xsd:element minOccurs="0" name="eta" type="xsd:long" />
  <xsd:element minOccurs="0" name="numberofSeeds" type="xsd:long" />
  <xsd:element minOccurs="0" name="peersList" nillable="true" type="tns:ArrayOfString" />
  <xsd:element minOccurs="0" name="status" nillable="true" type="xsd:string" />
  </xsd:sequence>
  </xsd:complexType>
- <xsd:complexType name="ArrayOfString">
- <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="string" nillable="true" type="xsd:string" />
  </xsd:sequence>
  </xsd:complexType>
- <xsd:element name="getDownloadStatusResponse">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true" type="tns:DownloadStatus" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="getVersion">
  <xsd:complexType />
  </xsd:element>
- <xsd:complexType name="SystemInformations">
- <xsd:sequence>
  <xsd:element minOccurs="0" name="configFile" nillable="true" type="xsd:string" />
  <xsd:element minOccurs="0" name="cpu" type="xsd:long" />
  <xsd:element minOccurs="0" name="memory" type="xsd:long" />
  <xsd:element minOccurs="0" name="os" nillable="true" type="xsd:string" />
  <xsd:element minOccurs="0" name="userName" nillable="true" type="xsd:string" />
  <xsd:element minOccurs="0" name="version" nillable="true" type="xsd:string" />
  </xsd:sequence>
  </xsd:complexType>
- <xsd:element name="getVersionResponse">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true" type="tns:SystemInformations" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="removeDownloadedObject">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="selectedMedia" nillable="true" type="tns:AXOIDFileName" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="removeDownloadedObjectResponse">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="out" type="xsd:int" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
```

```
- <xsd:element name="getDownloadingObjectList">
  <xsd:complexType />
  </xsd:element>
- <xsd:complexType name="P2PObjectList">
- <xsd:sequence>
  <xsd:element minOccurs="0" name="publishedObjectList" nillable="true" type="tns:ArrayOfP2PObjectItem" />
  </xsd:sequence>
  </xsd:complexType>
- <xsd:complexType name="ArrayOfP2PObjectItem">
- <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="P2PObjectItem" nillable="true" type="tns:P2PObjectItem" />
  </xsd:sequence>
  </xsd:complexType>
- <xsd:complexType name="P2PObjectItem">
- <xsd:sequence>
  <xsd:element minOccurs="0" name="AXOID" nillable="true" type="xsd:string" />
  <xsd:element minOccurs="0" name="infoHash" nillable="true" type="xsd:string" />
  <xsd:element minOccurs="0" name="localFileName" nillable="true" type="xsd:string" />
  </xsd:sequence>
  </xsd:complexType>
- <xsd:element name="getDownloadingObjectListResponse">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true" type="tns:P2PObjectList" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="downloadObject1">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="axoidfn" nillable="true" type="tns:AXOIDFileName" />
  <xsd:element maxOccurs="1" minOccurs="1" name="trackerURL" nillable="true" type="xsd:string" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="downloadObject1Response">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="out" type="xsd:int" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="getDownloadedObjectURL">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="selectedMedia" nillable="true" type="tns:AXOIDFileName" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="getDownloadedObjectURLResponse">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true" type="xsd:string" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="downloadObject">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="torrentFile" nillable="true" type="xsd:string" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="downloadObjectResponse">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="out" type="xsd:int" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="getPublishedObjectList">
```

```
  <xsd:complexType />
  </xsd:element>
- <xsd:element name="getPublishedObjectListResponse">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true" type="tns:P2PObjectList" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="getPublicationStatus">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="selectedMedia" nillable="true" type="tns:AXOIDFileName" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:complexType name="PublishStatus">
- <xsd:sequence>
  <xsd:element minOccurs="0" name="numberOfSeeds" type="xsd:long" />
  <xsd:element minOccurs="0" name="peersList" nillable="true" type="tns:ArrayOfString" />
  <xsd:element minOccurs="0" name="segmentsPeers" nillable="true" type="tns:ArrayOfSegment" />
  <xsd:element minOccurs="0" name="uploadedKBytes" type="xsd:long" />
  </xsd:sequence>
  </xsd:complexType>
- <xsd:complexType name="ArrayOfSegment">
- <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="Segment" nillable="true" type="tns:Segment" />
  </xsd:sequence>
  </xsd:complexType>
- <xsd:complexType name="Segment">
- <xsd:sequence>
  <xsd:element minOccurs="0" name="idPiece" type="xsd:int" />
  <xsd:element minOccurs="0" name="ip" nillable="true" type="tns:ArrayOfString" />
  </xsd:sequence>
  </xsd:complexType>
- <xsd:element name="getPublicationStatusResponse">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="out" nillable="true" type="tns:PublishStatus" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="controlDownloadingObject">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="selectedMedia" nillable="true" type="tns:AXOIDFileName" />
  <xsd:element maxOccurs="1" minOccurs="1" name="startstop" type="xsd:boolean" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
- <xsd:element name="controlDownloadingObjectResponse">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="out" type="xsd:int" />
  </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
  </xsd:schema>
  </wsdl:types>
- <wsdl:message name="downloadObject1Request">
  <wsdl:part name="parameters" element="tns:downloadObject1" />
  </wsdl:message>
- <wsdl:message name="getDownloadedObjectURLResponse">
  <wsdl:part name="parameters" element="tns:getDownloadedObjectURLResponse" />
  </wsdl:message>
- <wsdl:message name="downloadObject1Response">
  <wsdl:part name="parameters" element="tns:downloadObject1Response" />
  </wsdl:message>
- <wsdl:message name="getDownloadedObjectURLRequest">
  <wsdl:part name="parameters" element="tns:getDownloadedObjectURL" />
  </wsdl:message>
```

```
- <wsdl:message name="publishObjectRequest">
  <wsdl:part name="parameters" element="tns:publishObject" />
  </wsdl:message>
- <wsdl:message name="controlDownloadingObjectResponse">
  <wsdl:part name="parameters" element="tns:controlDownloadingObjectResponse" />
  </wsdl:message>
- <wsdl:message name="downloadObjectURLResponse">
  <wsdl:part name="parameters" element="tns:downloadObjectURLResponse" />
  </wsdl:message>
- <wsdl:message name="removeDownloadedObjectRequest">
  <wsdl:part name="parameters" element="tns:removeDownloadedObject" />
  </wsdl:message>
- <wsdl:message name="getStatusResponse">
  <wsdl:part name="parameters" element="tns:getStatusResponse" />
  </wsdl:message>
- <wsdl:message name="getDownloadingObjectListResponse">
  <wsdl:part name="parameters" element="tns:getDownloadingObjectListResponse" />
  </wsdl:message>
- <wsdl:message name="getDownloadStatusResponse">
  <wsdl:part name="parameters" element="tns:getDownloadStatusResponse" />
  </wsdl:message>
- <wsdl:message name="removeDownloadedObjectResponse">
  <wsdl:part name="parameters" element="tns:removeDownloadedObjectResponse" />
  </wsdl:message>
- <wsdl:message name="controlDownloadingObjectRequest">
  <wsdl:part name="parameters" element="tns:controlDownloadingObject" />
  </wsdl:message>
- <wsdl:message name="getPublishedObjectListRequest">
  <wsdl:part name="parameters" element="tns:getPublishedObjectList" />
  </wsdl:message>
- <wsdl:message name="publishObjectResponse">
  <wsdl:part name="parameters" element="tns:publishObjectResponse" />
  </wsdl:message>
- <wsdl:message name="getVersionResponse">
  <wsdl:part name="parameters" element="tns:getVersionResponse" />
  </wsdl:message>
- <wsdl:message name="downloadObjectRequest">
  <wsdl:part name="parameters" element="tns:downloadObject" />
  </wsdl:message>
- <wsdl:message name="getPublicationStatusResponse">
  <wsdl:part name="parameters" element="tns:getPublicationStatusResponse" />
  </wsdl:message>
- <wsdl:message name="getVersionRequest">
  <wsdl:part name="parameters" element="tns:getVersion" />
  </wsdl:message>
- <wsdl:message name="removePublishedObjectResponse">
  <wsdl:part name="parameters" element="tns:removePublishedObjectResponse" />
  </wsdl:message>
- <wsdl:message name="getDownloadStatusRequest">
  <wsdl:part name="parameters" element="tns:getDownloadStatus" />
  </wsdl:message>
- <wsdl:message name="getStatusRequest">
  <wsdl:part name="parameters" element="tns:getStatus" />
  </wsdl:message>
- <wsdl:message name="getDownloadingObjectListRequest">
  <wsdl:part name="parameters" element="tns:getDownloadingObjectList" />
  </wsdl:message>
- <wsdl:message name="getPublicationStatusRequest">
  <wsdl:part name="parameters" element="tns:getPublicationStatus" />
  </wsdl:message>
- <wsdl:message name="getPublishedObjectListResponse">
  <wsdl:part name="parameters" element="tns:getPublishedObjectListResponse" />
  </wsdl:message>
- <wsdl:message name="downloadObjectURLRequest">
  <wsdl:part name="parameters" element="tns:downloadObjectURL" />
  </wsdl:message>
- <wsdl:message name="removePublishedObjectRequest">
  <wsdl:part name="parameters" element="tns:removePublishedObject" />
  </wsdl:message>
- <wsdl:message name="downloadObjectResponse">
  <wsdl:part name="parameters" element="tns:downloadObjectResponse" />
```

```
  </wsdl:message>
- <wsdl:portType name="P2PMonitoringPortType">
- <wsdl:operation name="getStatus">
  <wsdl:input name="getStatusRequest" message="tns:getStatusRequest" />
  <wsdl:output name="getStatusResponse" message="tns:getStatusResponse" />
  </wsdl:operation>
- <wsdl:operation name="removePublishedObject">
  <wsdl:input name="removePublishedObjectRequest" message="tns:removePublishedObjectRequest" />
  <wsdl:output name="removePublishedObjectResponse" message="tns:removePublishedObjectResponse" />
  </wsdl:operation>
- <wsdl:operation name="publishObject">
  <wsdl:input name="publishObjectRequest" message="tns:publishObjectRequest" />
  <wsdl:output name="publishObjectResponse" message="tns:publishObjectResponse" />
  </wsdl:operation>
- <wsdl:operation name="downloadObjectURL">
  <wsdl:input name="downloadObjectURLRequest" message="tns:downloadObjectURLRequest" />
  <wsdl:output name="downloadObjectURLResponse" message="tns:downloadObjectURLResponse" />
  </wsdl:operation>
- <wsdl:operation name="getDownloadStatus">
  <wsdl:input name="getDownloadStatusRequest" message="tns:getDownloadStatusRequest" />
  <wsdl:output name="getDownloadStatusResponse" message="tns:getDownloadStatusResponse" />
  </wsdl:operation>
- <wsdl:operation name="getVersion">
  <wsdl:input name="getVersionRequest" message="tns:getVersionRequest" />
  <wsdl:output name="getVersionResponse" message="tns:getVersionResponse" />
  </wsdl:operation>
- <wsdl:operation name="removeDownloadedObject">
  <wsdl:input name="removeDownloadedObjectRequest" message="tns:removeDownloadedObjectRequest" />
  <wsdl:output name="removeDownloadedObjectResponse" message="tns:removeDownloadedObjectResponse" />
  </wsdl:operation>
- <wsdl:operation name="getDownloadingObjectList">
  <wsdl:input name="getDownloadingObjectListRequest" message="tns:getDownloadingObjectListRequest" />
  <wsdl:output name="getDownloadingObjectListResponse" message="tns:getDownloadingObjectListResponse" />
  </wsdl:operation>
- <wsdl:operation name="downloadObject1">
  <wsdl:input name="downloadObject1Request" message="tns:downloadObject1Request" />
  <wsdl:output name="downloadObject1Response" message="tns:downloadObject1Response" />
  </wsdl:operation>
- <wsdl:operation name="getDownloadedObjectURL">
  <wsdl:input name="getDownloadedObjectURLRequest" message="tns:getDownloadedObjectURLRequest" />
  <wsdl:output name="getDownloadedObjectURLResponse" message="tns:getDownloadedObjectURLResponse" />
  </wsdl:operation>
- <wsdl:operation name="downloadObject">
  <wsdl:input name="downloadObjectRequest" message="tns:downloadObjectRequest" />
  <wsdl:output name="downloadObjectResponse" message="tns:downloadObjectResponse" />
  </wsdl:operation>
- <wsdl:operation name="getPublishedObjectList">
  <wsdl:input name="getPublishedObjectListRequest" message="tns:getPublishedObjectListRequest" />
  <wsdl:output name="getPublishedObjectListResponse" message="tns:getPublishedObjectListResponse" />
  </wsdl:operation>
- <wsdl:operation name="getPublicationStatus">
  <wsdl:input name="getPublicationStatusRequest" message="tns:getPublicationStatusRequest" />
  <wsdl:output name="getPublicationStatusResponse" message="tns:getPublicationStatusResponse" />
  </wsdl:operation>
- <wsdl:operation name="controlDownloadingObject">
  <wsdl:input name="controlDownloadingObjectRequest" message="tns:controlDownloadingObjectRequest" />
  <wsdl:output name="controlDownloadingObjectResponse" message="tns:controlDownloadingObjectResponse" />
  </wsdl:operation>
  </wsdl:portType>
- <wsdl:binding name="P2PMonitoringHttpBinding" type="tns:P2PMonitoringPortType">
  <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="getStatus">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getStatusRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="getStatusResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="removePublishedObject">
```

```
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="removePublishedObjectRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="removePublishedObjectResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="publishObject">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="publishObjectRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="publishObjectResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="downloadObjectURL">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="downloadObjectURLRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="downloadObjectURLResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getDownloadStatus">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getDownloadStatusRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="getDownloadStatusResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getVersion">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getVersionRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="getVersionResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="removeDownloadedObject">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="removeDownloadedObjectRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="removeDownloadedObjectResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getDownloadingObjectList">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getDownloadingObjectListRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="getDownloadingObjectListResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="downloadObject1">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="downloadObject1Request">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="downloadObject1Response">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
```

```
- <wsdl:operation name="getDownloadedObjectURL">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getDownloadedObjectURLRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="getDownloadedObjectURLResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="downloadObject">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="downloadObjectRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="downloadObjectResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getPublishedObjectList">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getPublishedObjectListRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="getPublishedObjectListResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getPublicationStatus">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="getPublicationStatusRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="getPublicationStatusResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="controlDownloadingObject">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="controlDownloadingObjectRequest">
  <wsdlsoap:body use="literal" />
  </wsdl:input>
- <wsdl:output name="controlDownloadingObjectResponse">
  <wsdlsoap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="P2PMonitoring">
- <wsdl:port name="P2PMonitoringHttpPort" binding="tns:P2PMonitoringHttpBinding">
  <wsdlsoap:address location="http://lotar.dsi.unifi.it:7780/WebServices/P2PMonitoring" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

13 Formal description of format BitTorrent MetaInfo

13.1 bencoding

Bencoding is a way to specify and organize data in a terse format. It supports the following types: byte strings, integers, lists, and dictionaries.

13.1.1 byte strings

Byte strings are encoded as follows: <string length encoded in base ten ASCII>**:**<string data>
Note that there is no constant beginning delimiter, and no ending delimiter.

> **Example**: 4**:**spam represents the string "spam"

13.1.2 integers

Integers are encoded as follows: **i**<integer encoded in base ten ASCII>**e**
The initial **i** and trailing **e** are beginning and ending delimiters. You can have negative numbers such as **i**-3**e**. You cannot prefix the number with a zero such as **i**04**e**. However, **i**0**e** is valid.

> **Example**: **i**3**e** represents the integer "3"

- NOTE: The maximum number of bit of this integer is unspecified, but to handle it as a signed 64bit integer is mandatory to handle "large files" aka .torrent for more that 4Gbyte

13.1.3 lists

*Lists are encoded as follows: **l**<bencoded values>**e***
*The initial **l** and trailing **e** are beginning and ending delimiters. Lists may contain any bencoded type, including integers, strings, dictionaries, and other lists.*

> ***Example: l**4:spam4:eggs**e** represents the list of two strings: ["spam", "eggs"]*

13.1.4 dictionaries

*Dictionaries are encoded as follows: **d**<bencoded string><bencoded element>**e***
*The initial **d** and trailing **e** are the beginning and ending delimiters. Note that the keys must be bencoded strings. The values may be any bencoded type, including integers, strings, lists, and other dictionaries. Keys must be strings and appear in sorted order (sorted as raw strings, not alphanumerics). The strings should be compared using a binary comparison, not a culture-specific "natural" comparison.*

> ***Example**: **d**3:cow3:moo4:spam4:eggs**e** represents the dictionary { "cow" => "moo", "spam" => "eggs" }*
> ***Example**: **d**4:spaml1:a1:be**e** represents the dictionary { "spam" => ["a", "b"] }*
> ***Example**: **d**9:publisher3:bob18:publisher.location4:home17:publisher-webpage15:www.example.com**e***

13.2 Metainfo File Structure

All data in a metainfo file is bencoded. The specification for bencoding is defined above.

The content of a metainfo file (the file ending in ".torrent") is a bencoded dictionary, containing the keys listed below. All character string values are UTF-8 encoded.

- **info**: a dictionary that describes the file(s) of the torrent. There are two possible forms: one for the case of a 'single-file' torrent with no directory structure, and one for the case of a 'multi-file' torrent (see below for details)
- **announce**: The announce URL of the tracker (string)
- **announce-list**: (optional) this is an extention to the official specification, which is also backwards compatible. This key is used to implement lists of backup trackers. The full specification can be found *here*.
- **creation date**: (optional) the creation time of the torrent, in standard UNIX epoch format (integer seconds since 1-Jan-1970 00:00:00 UTC)
- **comment**: (optional) free-form textual comments of the author (string)
- **created by**: (optional) name and version of the program used to create the .torrent (string)

13.2.1 Info Dictionary

This section contains the field which are common to both mode, "single file" and "multiple file".

- **piece length**: number of bytes in each piece (integer)
- **pieces**: string consisting of the concatenation of all 20-byte SHA1 hash values, one per piece (byte string)
- **private**: (optional) this field is an integer. If it is set to "1", the client MUST publish its presence ot get other peers ONLY via the trackers explicitly described in the metainfo file. If this field is set to "0" or is not present, the client may obtain peer from other means, e.g. PEX peer exchange, dht. Here, "private" may be read as "no external peer source".
    - **NOTE:** this definition is a stub, use it at your own risk, some disagree with it. fell free to improve it. *http://www.azureuswiki.com/index.php/Secure_Torrents* is the definition from azureus wiki
    - Additionnally it should be noted that even if this field is used in practice, it is not part of the official specification.

13.2.1.1 Info in Single File Mode

For the case of the **single-file** mode, the **info** dictionary contains the following structure:

- **name**: the filename of the file. This is purely advisory. (string)
- **length**: length of the file in bytes (integer)
- **md5sum**: (optional) a 32-character hexadecimal string corresponding to the MD5 sum of the file. This is not used by BitTorrent at all, but it is included by some programs for greater compatibility.

13.2.1.2 Info in Multiple File Mode

For the case of the **multi-file** mode, the **info** dictionary contains the following structure:

- **name**: the filename of the directory in which to store all the files. This is purely advisory. (string)
- **files**: a list of dictionaries, one for each file. Each dictionary in this list contains the following keys:
    - **length**: length of the file in bytes (integer)
    - **md5sum**: (optional) a 32-character hexadecimal string corresponding to the MD5 sum of the file. This is not used by BitTorrent at all, but it is included by some programs for greater compatibility.
    - **path**: a list containing one or more string elements that together represent the path and filename. Each element in the list corresponds to either a directory name or (in the case of the final element) the filename. For example, a the file "dir1/dir2/file.ext" would consist of three string elements: "dir1", "dir2", and "file.ext". This is encoded as a bencoded list of strings such as l"**4**:dir1'**4**:dir2'**8**:file.ext'**e***

13.2.2 Notes

- The **piece length** specifies the nominal piece size, and is usually a power of 2. The piece size is typically chosen based on the total amount of file data in the torrent, constrained by the fact that piece sizes too large cause inefficiency, and too small a piece size will result in a large .torrent metadata file. The conventional wisdom used to be to pick the smallest piece size that results in a .torrent file no greater than approx. 50 - 75 kB (presumably to ease the load on the server hosting the torrent files). However, now that hosting storage and bandwidth are not tightly constrained, it is best to keep the piece size to 512KB or less, at least for torrents under 8-10GB or so, even if that results in a larger torrent file, in order to have a more efficient swarm for sharing files. The most common sizes are 256 kB, 512 kB, and 1 MB. Every piece is of equal length except for the final piece, which is irregular. The number of pieces is thus determined by 'ceil(total length / piece size)'. For the purposes of piece boundaries in the multi-file case, consider the file data as one long continuous stream, composed of the concatenation of each file in the order listed in the files list. The number of pieces and their boundaries are then determined in the same manner as the case of a single file. Pieces may overlap file boundaries.
- Each piece has a corresponding SHA1 hash of the data contained within that piece. These hashes are concatenated to form the pieces **value in the above info** dictionary. Note that this is **not** a list but rather a single string. The length of the string must be a multiple of 20.

14 Formal description of communication protocol BitTorrent Tracker

The tracker is an HTTP/HTTPS service which responds to HTTP GET requests. The requests include metrics from clients that help the tracker keep overall statistics about the torrent. The response includes a peer list that helps the client participate in the torrent. The base URL consists of the "announce URL" as defined in the metadata (.torrent) file. The parameters are then added to this URL, using standard CGI methods (i.e. a '?' after the announce URL, followed by 'param=value' sequences separated by '&').

Note that all binary data in the URL (particularly info_hash and peer_id) must be properly escaped. This means any byte not in the set 0-9, a-z, A-Z, and $-_.+!*'(), must be encoded using the "%nn" format, where nn is the hexadecimal value of the byte. (See RFC1738 for details.)

The parameters used in the client->tracker GET request are as follows:

- **info_hash**: 20-byte SHA1 hash of the value of the info key from the Metainfo file. Note that the value will be a bencoded dictionary, given the definition of the info key above. Note: This string is always urlencoded, as opposed to *peer_id*, which needs to be unencoded.
- *peer_id*: 20-byte string used as a unique ID for the client, generated by the client at startup. This is allowed to be any value, and may be binary data. There are currently no guidelines for generating this peer ID. However, one may rightly presume that it must at least be unique for your local machine, thus should probably incorporate things like process ID and perhaps a timestamp recorded at startup. See *peer_id* below for common client encodings of this field.
- **port**: The port number that the client is listening on. Ports reserved for BitTorrent are typically 6881-6889. Clients may choose to give up if it cannot establish a port within this range.
- **uploaded**: The total amount uploaded (since the client sent the 'started' event to the tracker) in base ten ASCII. While not explicitly stated in the official specification, the concensus is that this should be the total number of bytes uploaded.
- **downloaded**: The total amount downloaded (since the client sent the 'started' event to the tracker) in base ten ASCII. While not explicitly stated in the official specification, the consensus is that this should be the total number of bytes downloaded.
- **left**: The number of bytes this client still has to download, encoded in base ten ASCII.
- **compact**: Indicates the client accepts a compact response. The peers list is replaced by a peers string with 6 bytes per peer. The first four bytes are the host (in network byte order), the last two bytes are the port (again in network byte order). It should be noted that some trackers only support compact responses (for saving bandwidth) and refuse normal requests.
- **event**: If specified, must be one of started, completed, stopped, (or empty which is the same as not being specified). If not specified, then this request is one performed at regular intervals.
    - **started**: The first request to the tracker must include the event key with this value.
    - **stopped**: Must be sent to the tracker if the client is shutting down gracefully.
    - **completed**: Must be sent to the tracker when the download completes. However, must not be sent if the download was already 100% complete when the client started. Presumably, this is to allow the tracker to increment the "completed downloads" metric based soley on this event.
- **ip**: Optional. The true IP address of the client machine, in dotted quad format or rfc3513 defined hexed IPv6 address. Notes: In general this parameter is not necessary as the address of the client can be determined from the IP address from which the HTTP request came. The parameter is only needed in the case where the IP address that the request came in on is not the IP address of the client. This happens if the client is communicating to the tracker through a proxy (or a transparent web proxy/cache.) It also is necessary when both the client and the tracker are on the same local side of a NAT gateway. The reason for this is that otherwise the tracker would give out the internal (RFC1918) address of the client, which is not routeable. Therefore the client must explicitly state its (external, routeable) IP address to be given out to external peers. Various trackers treat this parameter differently. Some only honor it only if the IP address that the request came in on is in RFC1918 space. Others honor

it unconditionally, while others ignore it completely. In case of IPv6 address (e.g.: 2001:db8:1:2::100) it indicates only that client can communicate via IPv6.
- **numwant**: Optional. Number of peers that the client would like to receive from the tracker. This value is permitted to be zero. If omitted, typically defaults to 50 peers.
- **key**: Optional. An additional identification that is not shared with any users. It is intended to allow a client to prove their identity should their IP address change.
- **trackerid**: Optional. If a previous announce contained a tracker id, it should be set here.

The tracker responds with "text/plain" document consisting of a bencoded dictionary with the following keys:

- **failure reason**: If present, then no other keys may be present. The value is a human-readable error message as to why the request failed (string).
- **warning message**: (new) Similar to failure reason, but the response still gets processed normally. The warning message is shown just like an error.
- **interval**: Interval in seconds that the client should wait between sending regular requests to the tracker (mandatory)
- **min interval**: Minimum announce interval. If present clients must not reannounce more frequently than this.
- **tracker id**: A string that the client should send back on its next announcements. If absent and a previous announce sent a tracker id, do not discard the old value; keep using it.
- **complete**: number of peers with the entire file, i.e. seeders (integer)
- **incomplete**: number of non-seeder peers, aka "leechers" (integer)
- **peers**: The value is a list of dictionaries, each with the following keys:
    - **peer id**: peer's self-selected ID, as described above for the tracker request (string)
    - **ip**: peer's IP address (either IPv6 or IPv4) or DNS name (string)
    - **port**: peer's port number (integer)

As mentioned above, the list of peers is length 50 by default. If there are fewer peers in the torrent, then the list will be smaller. Otherwise, the tracker randomly selects peers to include in the response. The tracker may choose to implement a more intelligent mechanism for peer selection when responding to a request. For instance, reporting seeds to other seeders could be avoided.

Clients may send a request to the tracker more often than the specified interval, if an event occurs (i.e. stopped or completed) or if the client needs to learn about more peers. However, it is considered bad practice to "hammer" on a tracker to get multiple peers. If a client wants a large peer list in the response, then it should specify the **numwant** parameter.

Implementer's Note: Even 30 peers is **plenty**, the official client version 3 in fact only actively forms new connections if it has less than 30 peers and will refuse connections if it has 55. **This value is important to performance**. When a new piece has completed download, HAVE messages (see below) will need to be sent to most active peers. As a result the cost of broadcast traffic grows in direct proportion to the number of peers. Above 25, new peers are highly unlikely to increase download speed. UI designers are **strongly** advised to make this obscure and hard to change as it is very rare to be useful to do so.

14.1 Tracker 'scrape' Convention

By convention most trackers support another form of request, which queries the state of a given torrent (or all torrents) that the tracker is managing. This is referred to as the "scrape page" because it automates the otherwise tedious process of "screen scraping" the tracker's stats page.

The scrape URL is also a HTTP GET method, similar to the one described above. However the base URL is different. To derive the scrape URL use the following steps: Begin with the announce URL. Find the last '/' in it. If the text immediately following that '/' isn't 'announce' it will be taken as a sign that that tracker doesn't support the scrape convention. If it does, substitute 'scrape' for 'announce' to find the scrape page.

Examples: (announce URL -> scrape URL)

```
  ~http://example.com/announce          -> ~http://example.com/scrape
  ~http://example.com/x/announce        -> ~http://example.com/x/scrape
  ~http://example.com/announce.php      -> ~http://example.com/scrape.php
  ~http://example.com/a                 -> (scrape not supported)
  ~http://example.com/announce?x<code>2%0644                                    ->
~http://example.com/scrape?x</code>2%0644
  ~http://example.com/announce?x=2/4    -> (scrape not supported)
  ~http://example.com/x%064announce     -> (scrape not supported)
```

Note especially that entity unquoting is 'not to be done. This standard is documented by Bram in the *BitTorrent* development list archive: *http://groups.yahoo.com/group/BitTorrent/message/3275*

The scrape URL may be supplemented by the optional parameter info_hash, a 20-byte value as described above. This restricts the tracker's report to that particular torrent. Otherwise stats for all torrents that the tracker is managing are returned. Software authors are strongly encouraged to use the info_hash parameter when at all possible, to reduce the load and bandwidth of the tracker.

The response of this HTTP GET method is a "text/plain" document consisting of a bencoded dictionary, containing the following keys:

- **files**: a dictionary containing one key/value pair for each torrent for which there are stats. If infohash* was supplied and was valid, this dictionary will contain a single key/value. Each key consists of a 20-byte binary infohash value. The value of that key is yet another nested dictionary containing the following:
  - **complete**: number of peers with the entire file, i.e. seeders (integer)
  - **downloaded**: total number of times the tracker has registered a completion ("event=complete", i.e. a client finished downloading the torrent)
  - **incomplete**: number of non-seeder peers, aka "leechers" (integer)
  - **name**: (optional) the torrent's internal name, as specified by the "name" file in the info section of the .torrent file

Note that this response has three levels of dictionary nesting. Here's an example:

```
d5:filesd20:....................d8:completei5e10:downloadedi50e10:
incompletei10eeee
```

Where . . . . . is the 20 byte info_hash and there are 5 seeders, 10 leechers, and 50 complete downloads.

14.1.1 Unofficial extensions to scrape

Below are the response keys are being unofficially used. Since they are unnofficial, they are all optional.

- **failure reason**: Human-readable error message as to why the request failed (string). Clients known to handle this key: Azureus.
- **flags**: a dictionary containing miscellaneous flags. The value of the flags key is another nested dictionary, possibly containing the following:
    - **min_request_interval**: The value for this key is an integer specifying how the minimum number of seconds for the client to wait before scraping the tracker again. Trackers known to send this key: BNBT. Clients known to handle this key: Azureus.

15 AXEPTool and AXMEDIA features comparison

The AXMEDIA Tool is the P2P application used in B2C via P2P. More requirements about Client/Server Distribution via PC are available in the specification of WP4 related to "AXMEDIS for Distribution via Internet". The AXMEDIA tool and solution has the above mentioned system and general requirements plus the following.

- Must provide a simple query support that allows simple search queries composition through a simplified GUI, providing results in a simplified format, easy to understand for the final users.

- Must provide complete results exposing AXMEDIS AXOID and metadata.

- Must be easily installable on a wide range of computers and possibly on different platforms such as Windows and MAC.

A stand alone Java based P2P tool without Web Service server and Web Server. AXMEDIA must not include the Web Service and Web Server code, neither in a deactivated form. Both AXEPTool and AXMEDIA must be created with conditional compilation and AXMEDIA project must not be a duplication of AXEPTool's one.

It has to be capable to access to the AXMEDIS P2P Query Service Server by means of a Web Service client.

What has to be done:

- It has to work as the AXEPTool on AXOID
- It has to allow making simple queries on the AXMEDIS P2P Query Service Server by means of a Web Service client. The query has to expose that is performed by an AXMEDIA tool.
- It has to allow to provide results of the query, direct selection of the file to be downloaded, selecting the line in the query result

The AXMEDIA tools presents:

- a nicer user interface
- a simplified query user interface
- a simplified monitoring tool and user interface

In the following sections an AXETPool/AXMedia feature comparison is presented.

15.1 Menu on left and window tabs

ID	AXEPTool Feature	AXMEDIA Feature
1	Menu on left with *Homepage*, it leads to open the Home Page Tab, reporting the AXMEDIS home page as defined into the configuration file	Same as in the AXEPTool
2	Menu on left with *Search*, it leads to open the Search Tab, reporting the Query Portal page as defined into the configuration file	Same as in the AXEPTool
3	Menu on left with *Catalogue*, it leads to open the Catalogue Tab, reporting the AXTrack Catalogue page as defined in the configuration file	Same as in the AXEPTool, it is probable that this feature is tooa complex andity not simple to be understood by the users.
4	Menu on left with *Downloads*, it leads to open the Downloads Tab, reporting the objects' transfer list	Same as in the AXEPTool
5	Menu on left with *Settings*, it leads to open the Settings Tab, that allows to edit the Settings reported in the configuration file too	Feature not requested in AXMEDIA, the form to change the setting has a sense only if some setting are still needed in AXMEDIA tool
6	Menu on left with *Publish*, it allows to select and publish an object through a file selector dialog window	Same as in the AXEPToolFeature not requested in AXMEDIA

15.2 Menu on top

ID	AXEPTool Feature	AXMEDIA Feature
7	Menu on top with *Help*, it leads to open a temporary popup which reports the AXMEDIS portal link:	Same as in the AXEPTool but the link has to be clickable
8	Menu on top with *About*, it leads to open a temporary popup which reports the AXMEDIS portal link and the current application version.	Same as in the AXEPToolSame as in the AXEPTool but the link has to be clickable

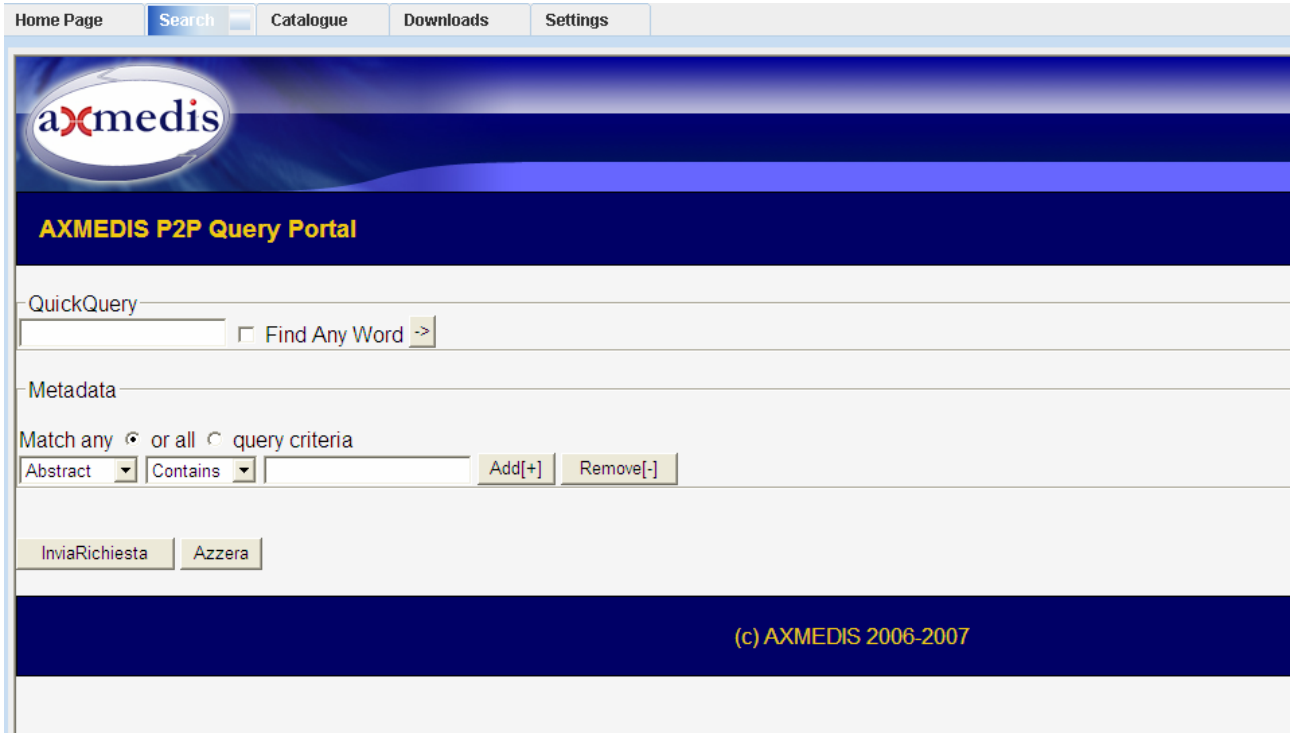
15.3 Browsing buttons

ID	AXEPTool Feature	AXMEDIA Feature
9	Browsing Left Button *Previous* under top menu, it backs to the previous cached page if clicked when the Home Page tab is selected. BUAG the link has to be clickable added to the wiki	Same as in the AXEPToolSame as in the AXEPTool but the button has to work with the Search page too
10	Browsing Right Button *Next* under top menu, it forwards to the previous cached page if clicked when the Home Page tab is selected BUAG the link has to be clickable added to the wiki	Same as in the AXEPToolSame as in the AXEPTool but the button has to work with the Search page too

15.4 Tracker URL

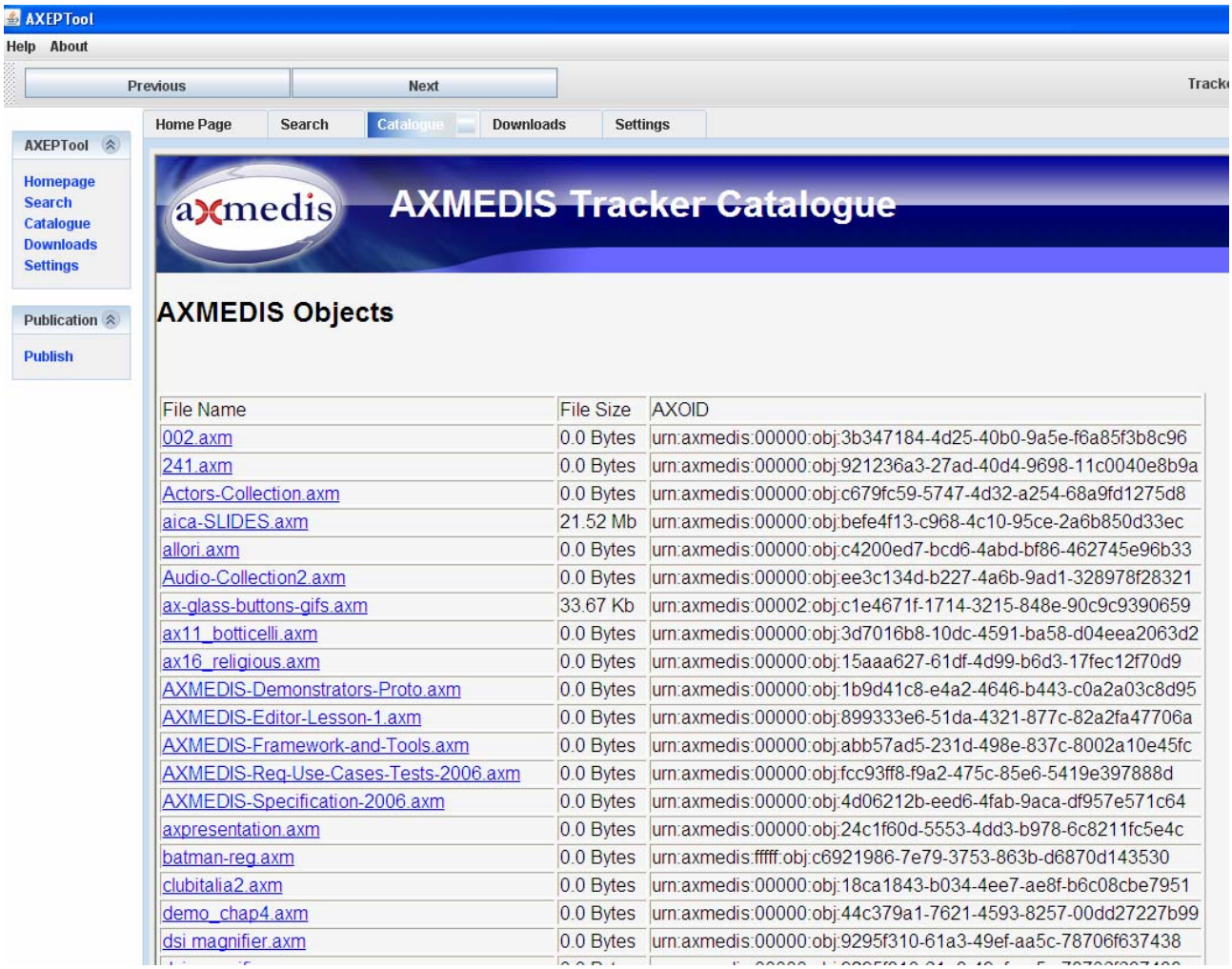
ID	AXEPTool Feature	AXMEDIA Feature
11	Tracker URL reported on the top right of the AXEPTool window	Feature not requested in AXMEDIA. This means that the AXMEDIA does not have t present the tracker url

15.5 Search Page



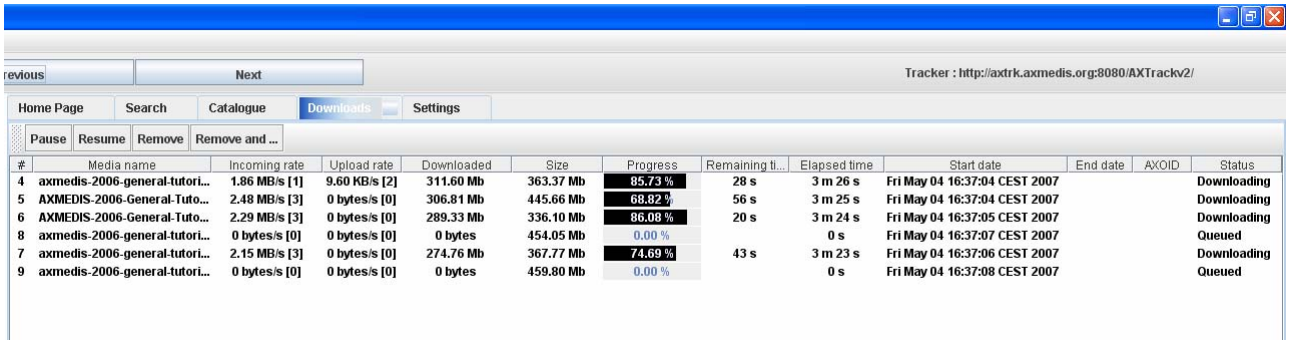
ID	AXEPTool Feature	AXMEDIA Feature
12	It show the Query support portal page which allows to make queries for objects metadata	Same as in the AXEPTool

15.6 Catalog Page



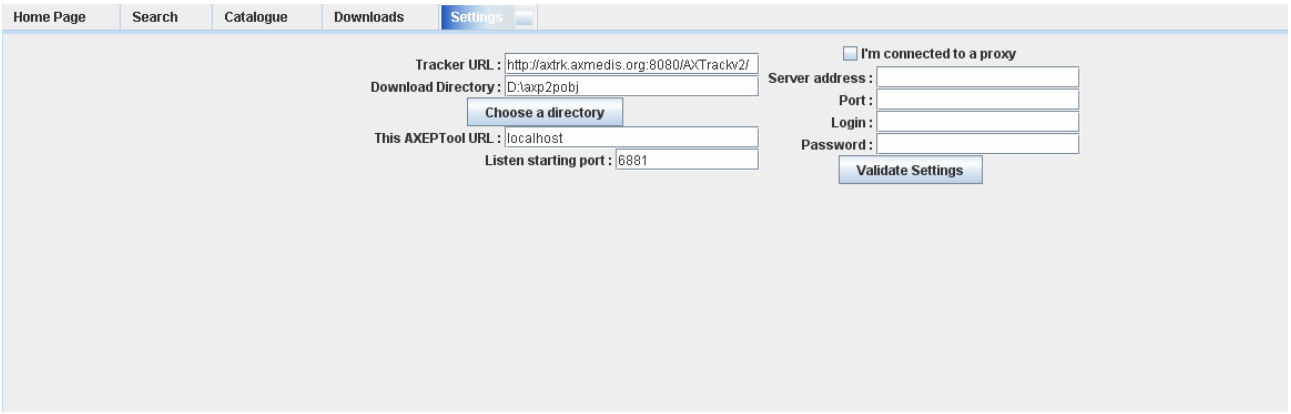
ID	AXEPTool Feature	AXMEDIA Feature
13	It shows the list of AXMEDIS and NON AXMEDIS objects published on the AXTrack, each of them with its file name, file size and AXOID (if present)	Same as in the AXEPTool but it has to list only AXMEDIS objects and doesn't have to report the AXOID field, probably the access to catalogue is not needed.

15.7 Downloads Page



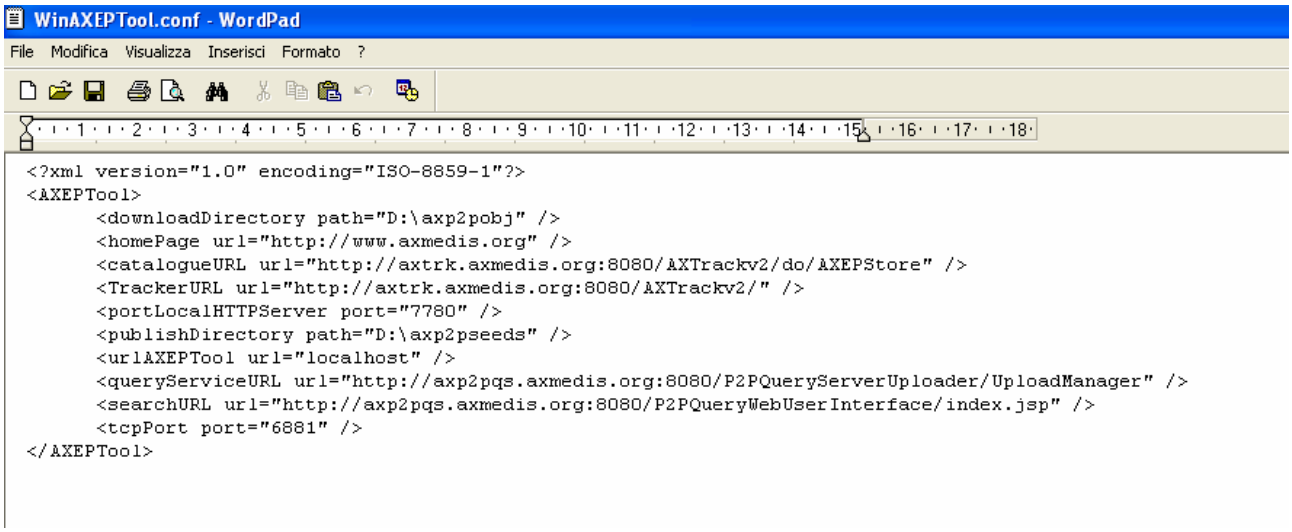
ID	AXEPTool Feature	AXMEDIA Feature
14	It shows the list of downloading/downloaded AXMEDIS and NON AXMEDIS objects with their files name, incoming and upload rate, downloaded size, file size, progress percentage, remaining time to completion, elapsed time, start date and time, end date and time, AXOID (if present) and current status	Same as in the AXEPTool but it doesn't have to report the AXOID field.
	The AXEPTool presents a quite sophisticated solution for the removal of bittorrent files and downloaded file.	This feature has to be probably be removed in the AXMEDIA in which the user should be forced to keep visible the files for sharing them among the people.

15.8 Settings Page



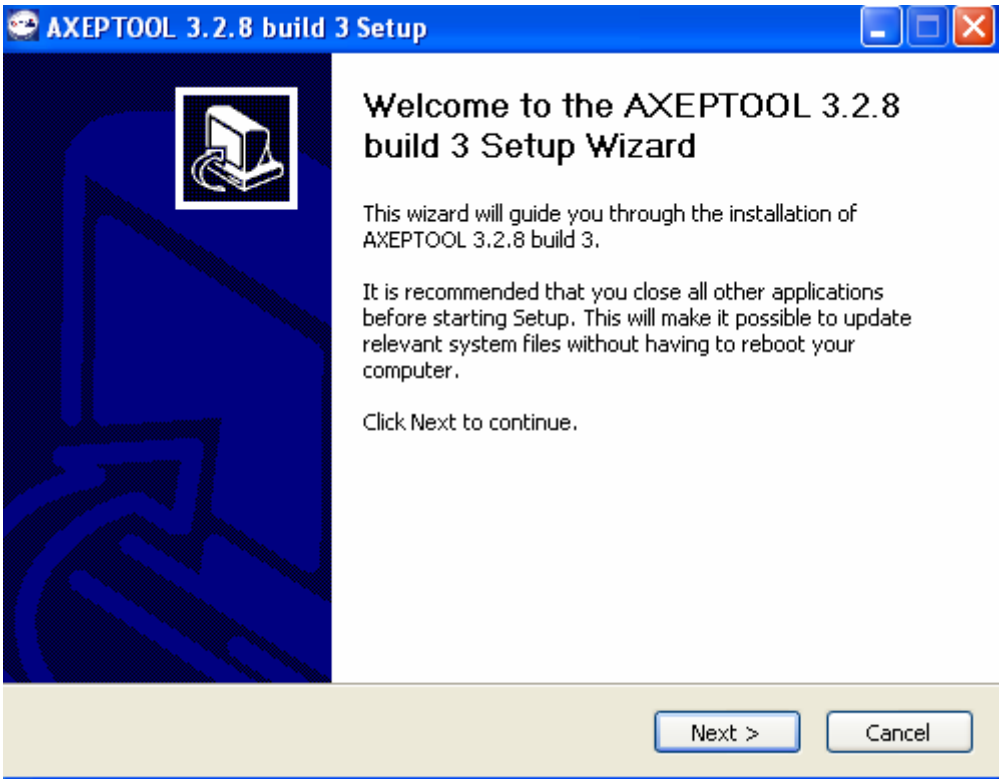
ID	AXEPTool Feature	AXMEDIA Feature
15	It shows and allows to edit the following required settings: *Tracker URL*, *Download Directory*, *AXEPTool URL*, *Listening starting port*, It shows and allows to edit the following optional proxy settings: *Server address*, *Port*, *Login*, *Password*	Features not requested in AXMEDIA
	It shows and allows to edit the following optional proxy settings: *Server address*, *Port*, *Login*, *Password*	NESI: It is not clear for me if these feature work, and what is their purpose. Once clarified a decision can be taken

15.9 WinAXEPTool.conf configuration file

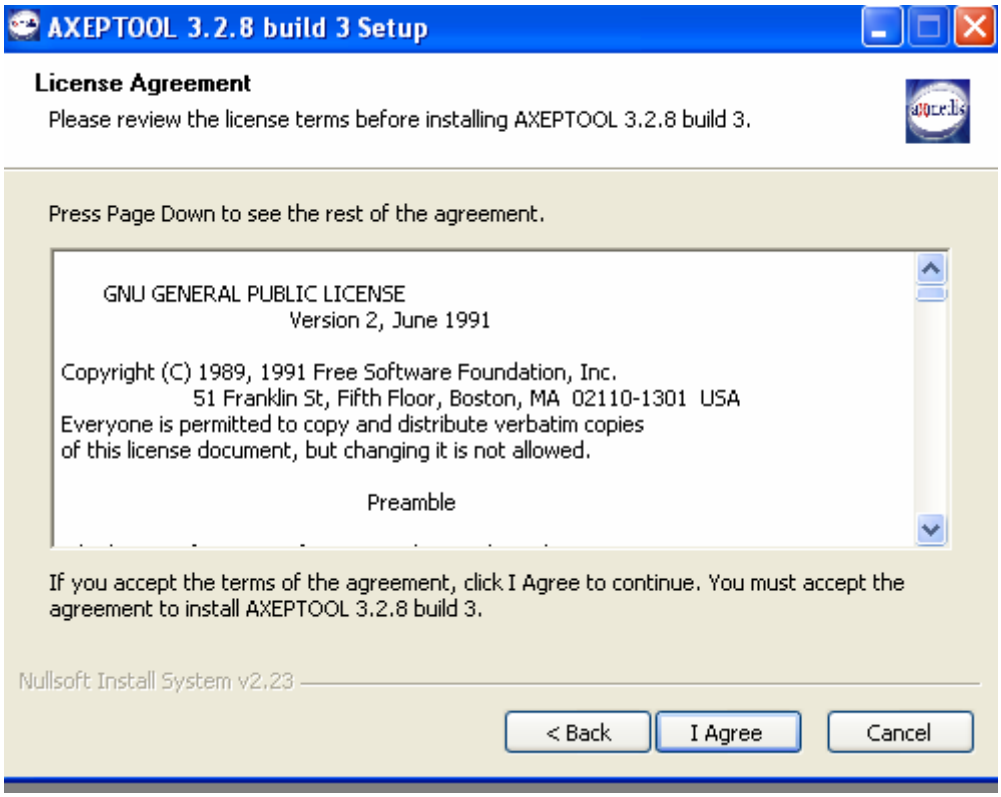


ID	AXEPTool Feature	AXMEDIA Feature
16	It reports the following editable settings: *downloadDirectory* path *homepage* url *catalogueURL* url *TrackerURL* url *portLocalHTTPServer* port *publishDirectory* path *urlAXPETool* url *queryServiceURL* url *searchURL* url *tcpPort* port	Same as in the AXEPTool, but it has to be a hidden file on the user's disk, but the following fields does not have to work any more in the AXMEDIA: *downloadDirectory* path *homepage* url *catalogueURL* url *TrackerURL* url *portLocalHTTPServer* port *publishDirectory* path *urlAXPETool* url *queryServiceURL* url *searchURL* url *tcpPort* port In the sense that corresponding values have to be coded into the java code.

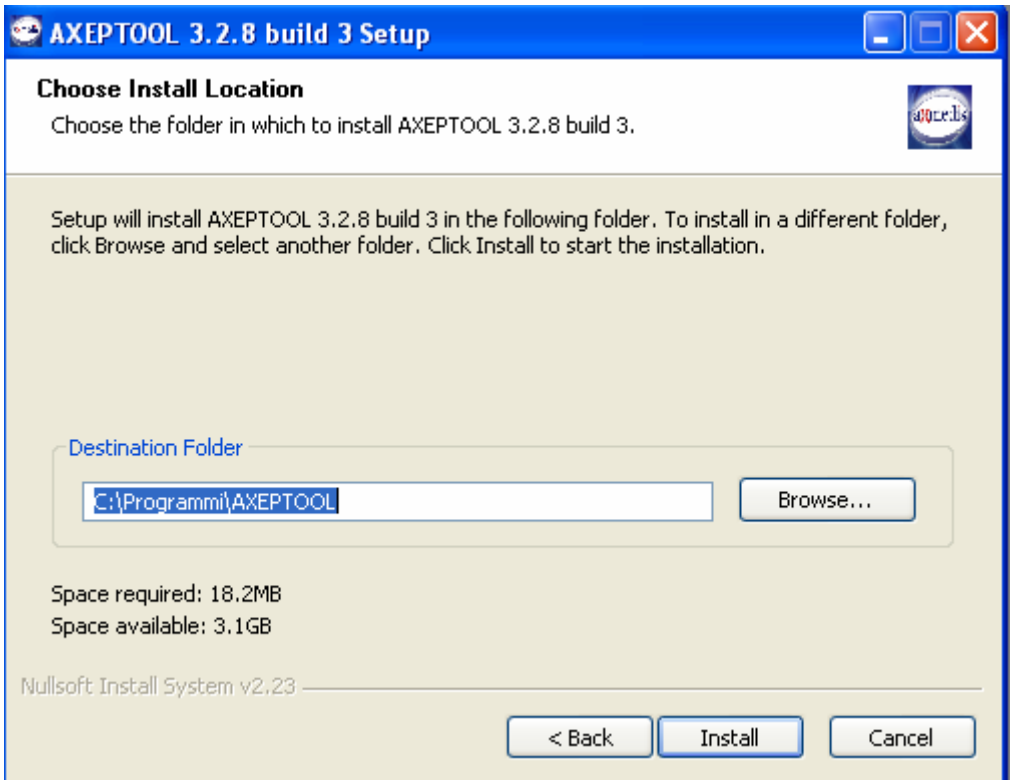
15.10 Tool's Installer



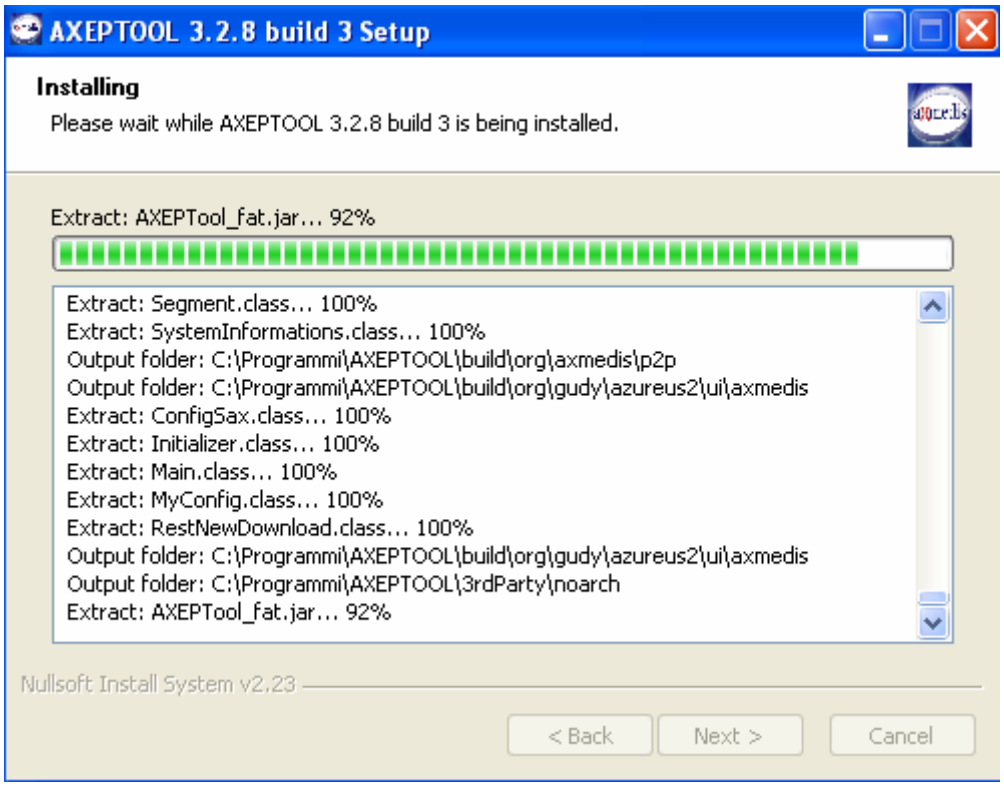
ID	AXEPTool Feature	AXMEDIA Feature
17	It's a NSIS generated installer that provides an interface for the AXEPTool installation, starting with a welcome screen reporting a welcome message, AXEPTool version, some recommendations, and asking the user to click a button to continue the installation process	Same as in the AXEPTool, but the tool has to be called AXMEDIA, etc…



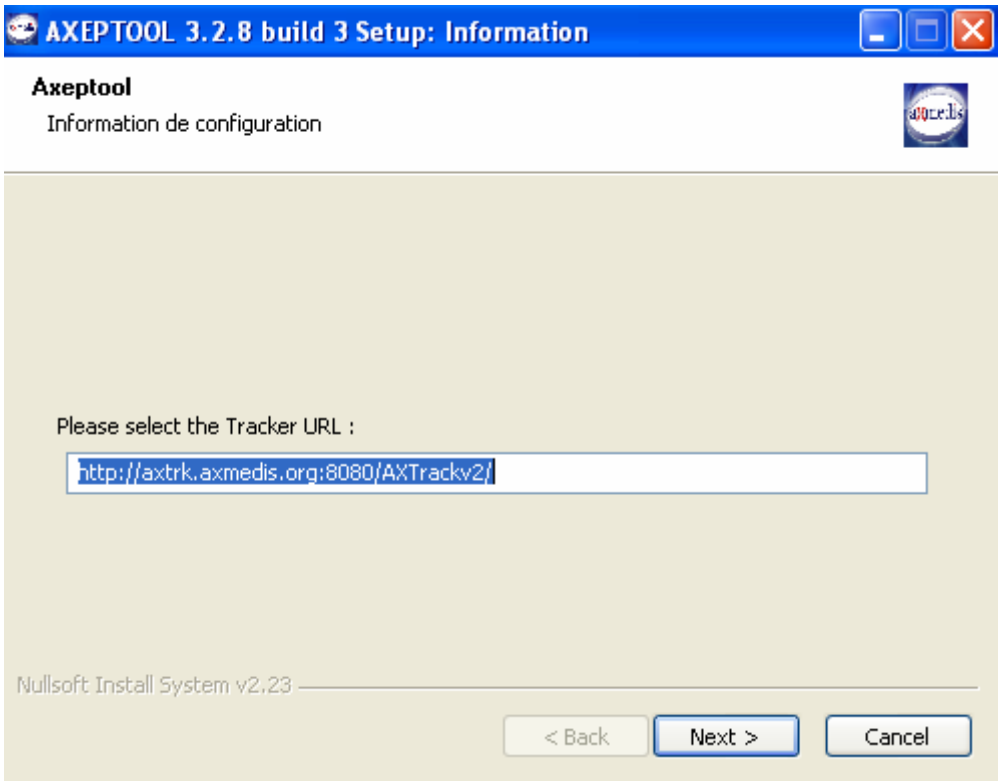
ID	AXEPTool Feature	AXMEDIA Feature
18	It asks the user to accept the general GNU PUBLIC LICENSE before continuing the installation process	Same as in the AXEPTool, text has to refer to AXMEDIA not to AXEPTool



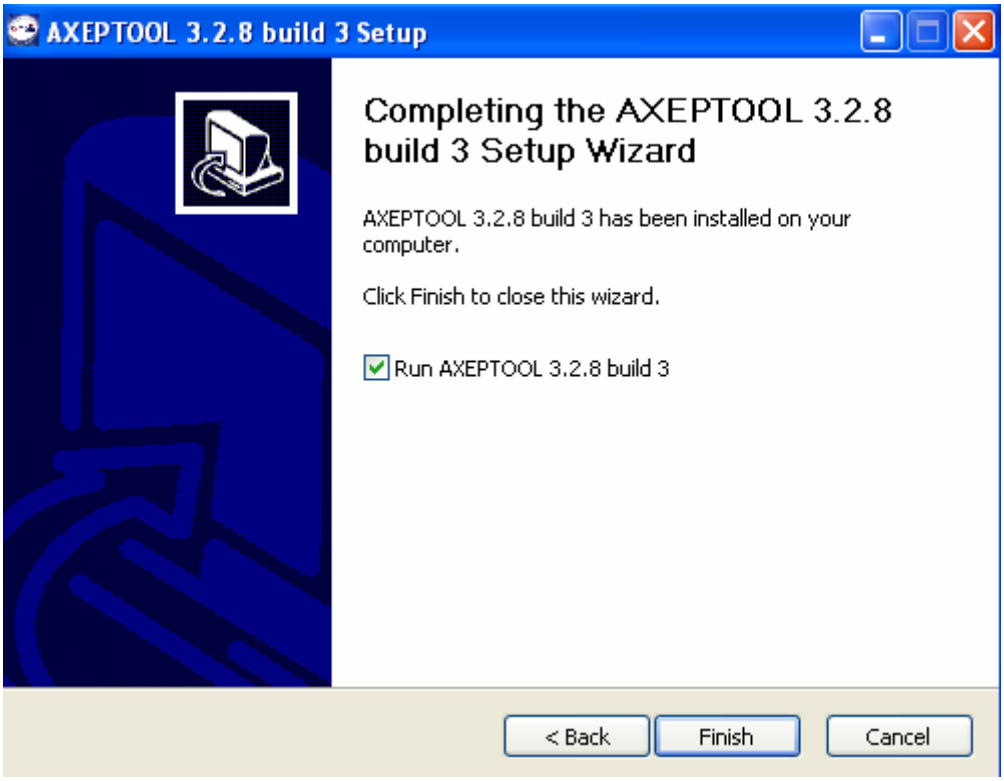
ID	AXEPTool Feature	AXMEDIA Feature
19	It asks the user to choose the installation folder before continuing the setup process	Same as in the AXEPTool, but the directory has to be c:\.....\AXMEDIA, The same for the information into the c:\document and setting\.....\AXMEDIA and not AXEPTool.



ID	AXEPTool Feature	AXMEDIA Feature
20	It starts copying the files	Same as in the AXEPTool



ID	AXEPTool Feature	AXMEDIA Feature
21	It asks the user to choose the default BitTorrent tracker before continuing the setup process	Feature not requested in AXMEDIA, this information has not be requested to the user



ID	AXEPTool Feature	AXMEDIA Feature
22	It confirms the installation completion and asks the user to start or not the AXEPTool before closing the installer	Same as in the AXEPTool but it has to ask the user if he/she wants to start AXMEDIA at boot too.
23	At installation the AXEPTool is automatically included into the tools to be executed at the start up.	the AXMEDIA installation has to ask at the User if he wants to have the AXMEDIA tools executed at the boot or NOT
24	The source code of AXEPTool and AXMEDIA has to be a unique project with two different manners of deploying the tools: AXEPTool and AXMEDIA.	The code and the AXMEDIA tool has to be obtained from the AXEPTool with a conditional compilation without duplicating the code.
25		The code that is not needed into AXMEDIA does not have to be left as hidden into the AXMEDIA version.

16 Basic Javascript Rules functionalities

THIS SECTION IS PRESENT ONLY IN THE CONFIDENTIAL VERSION OF THISDOCUMENT THAT CAN BE ACCESSED BY AFFILIATED PARTNERS OF AXMEDIS ONLY.

17 Javascript Rules examples

In this section some javascript rules examples will be provided to show how an AXMEDIS P2P Network can be monitored and controlled through the use of Web Services.

THIS SECTION IS PRESENT ONLY IN THE CONFIDENTIAL VERSION OF THISDOCUMENT THAT CAN BE ACCESSED BY AFFILIATED PARTNERS OF AXMEDIS ONLY.

18 AXTrack user Manual

18.1 Prerequisites

Axmedis Tracker (Axtrack later in this document), requires the following in order to install properly :

- A Java 5 compliant virtual machine. (IBM or BEA virtual machine is recommanded but not mandatory).
- An Apache Tomcat 5.5 (Tested on 5.5.20) running under that virtual machine.
- A working mysql installation.

If you do not have an IBM virtual machine, please download it from there :
http://www-128.ibm.com/developerworks/java/jdk/

And restart tomcat with the JAVA_HOME variable pointing to it. This JVM is fully compatible with Sun's one but has several bug removed (especially some memory leaks in the heap) which make it more suitable for use in a production environement.

If you plan to use Sun's virtual machine, it is recommanded to increase the Java heap size as much as you can. Here is an example (please take into account the maximum amount of memory you have in order to avoid swapping) :

JAVA_OPTS="-Xms256M -Xmx512M"

If you intend to build the application from the sourcecode, you'll need the following :
JDK 1.5 or later is needed either from Sun or IBM. (Both are compatible)
Apache Ant version 1.6

Please note that all necessary dependencies are provided in the 3rdParty directory of the build or source files. You can replace those dependencies by later versions if you like but do so at your own risks.

18.2 Installation

18.2.1 Building from source

You can build the whole application from sourcecode :

- Download the file from the SVN or get a release tarball.
- If you got a tarball, unpack it.
- CD to AXTrack's root directory
- Type “ant” on the command prompt, then enter.
- The application should build without errors.
- CD to dist/ directory
- The AXTrackv2 WAR file is the binary you can distribute.

18.2.2 Preparing the system

The first step is to prepare the database system :

Create the database as follows :

Log on to Mysql, type on your shell : “mysql -u root -p”
Give the root password at prompt
Issue : “CREATE DATABASE hexatrack;” at the MySQL prompt.

Create the database schema with the following SQL :

DROP TABLE IF EXISTS `ConfigurationAxtrack`;
CREATE TABLE `ConfigurationAxtrack` (

```
  `Id` bigint(20) default NULL,
  `AcceptAxmedis` varchar(10) default NULL
) DEFAULT CHARSET=latin1;

INSERT INTO `ConfigurationAxtrack` VALUES (1,'YES');

DROP TABLE IF EXISTS `Peers`;
CREATE TABLE `Peers` (
  `Id` bigint(20) NOT NULL auto_increment,
  `Idt` bigint(20) default NULL,
  `PeerIp` bigint(20) default NULL,
  `LastUpdate` bigint(20) default NULL,
  `PeerPort` int(11) default NULL,
  `Dl_sec` bigint(11) default NULL,
  `Status` varchar(20) default NULL,
  `Completed` tinyint(1) default NULL,
  `StartingTime` biDROP TABLE IF EXISTS `Torrents`;
CREATE TABLE `Torrents` (
  `Id` bigint(20) NOT NULL auto_increment,
  `Info_Hash` varchar(80) default NULL,
  `Active` tinyint(1) default NULL,
  `Banned` tinyint(1) default NULL,
  `Description` varchar(255) default NULL,
  `NbDownload` bigint(20) default NULL,
  `Axoid` varchar(80) default NULL,
  `FileSize` bigint(20) default NULL,
  `Protected` tinyint(1) default NULL,
  PRIMARY KEY  (`Id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;gint(11) default NULL,
  PRIMARY KEY  (`Id`)
) DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `Torrents`;
CREATE TABLE `Torrents` (
  `Id` bigint(20) NOT NULL auto_increment,
  `Info_Hash` varchar(80) default NULL,
  `Active` tinyint(1) default NULL,
  `Banned` tinyint(1) default NULL,
  `Description` varchar(255) default NULL,
  `NbDownload` bigint(20) default NULL,
  `Axoid` varchar(80) default NULL,
  `FileSize` bigint(20) default NULL,
  `Protected` tinyint(1) default NULL,
  PRIMARY KEY  (`Id`)
) DEFAULT CHARSET=latin1;
```

A file named hexatrack.sql has been provided in the SVN repository. You can use it with a redirection to fasttrack the schema creation.

Now, create an user and a password as follows :

GRANT ALL PRIVILEGES ON hexatrack. TO hexatrack IDENTIFIED BY 'yourpassword'*

Your database system is ready ! If you have any problem with this procedure, please contact your database administrator or check MySQL documentation.

Next step, copy the seedingpool (available on your source distribution), anywhere you like on your server for example :

 # cp -r seedingpool/ /AXTrack

Do NOT forget to give appropriate rights for the tomcat process to read and write to this directory. (Do the appropriate chmod). Failing to do so will result in an exception. If you want to diagnose and see if this exception is linked to a file rights problem, you can make the /AXTrack directory world readable/writable by doing a chmod 777. (But please correct the problem rightafter, chmod 777 is a big security risk in a production environement).

18.2.3 Installing WAR on a production server

Please ensure that your Tomcat is running under an IBM or BEA virtual machine if you intend to use the system in production. Also please note that AXTrack is server software. As a result, we recommand a Linux or Unix-like system for production use. Usage under Windows should work but is not recommanded.

Once your tomcat environement is setup:

- Login to the tomcat manager.
- Check that no AXTrackv2 application is already loaded. If one is already loaded, unload it now.
- Deploy the WAR file you just generated. You should connect it to the /AXTrackv2 URI. If you do otherwise, please update the configuration of all client software.

WARNING : If you use the application under Sun's JVM against our recommandation, please note that frequent loading and unloading of the application will create a memory leak. A full description of the problem is available here :
http://www.jroller.com/page/agileanswers?entry=preventing_java_s_java_lang

The next step is to edit the configuration file :

cd $TOMCAT_HOME/webapps/AXTrackv2/WEB-INF/classes/

Once here, edit the AXTrack.properties file :

 # RessourceBunble for AXTrack

axtrack.host=localhost:8080

axtrack.url=AXTrackv2
axtrack.linux.path=/AXTrack/
axtrack.linux.maketorrentcommand=btmakemetafile
axtrack.windows.path=c:/AXTrack/
axtrack.windows.maketorrentcommand=python c:/BitTorrent/maketorrent-console.py
axtrack.externalipadress=150.217.15.236
axtrack.localnetwork=192.168.
axtrack.localipadress=192.168.0.102
axtrack.seedtorrents=false
axtrack.debugLevel=1
axtrack.fulltrackerurl=http://tracker.hexaglobe.com:8084/AXTrackv2/do/Announce

You should need to modify only ip addresses, and the fulltrackerurl. If you want to activate the torrent seeding functionality (warning this will consume a lot of disk space), set it to true. If you changed the path to the seeding pool, change the axtrack.linux.path.

Then edit the hibernate.cfg.xml file and adjust it for your database parameters.

Your AXTrack server should be ready by now. Change directory to your $TOMCAT_HOME and issue the following commands :

./catalina.sh stop
< a few seconds wait >
./catalina.sh start

Connect to the following URL :

http://mytracker.mydomain.com:8080/AXTrackv2/

18.3 Troubleshooting

Your AXTrack software can receive a very high number of requests. As a result, you should be careful when configuring your Tomcat and MySQL server. Here are some troubleshootings tips :

When I connect to the tracker URL, I get an error 404 :
Your AXTrack war file has not been correctly loaded into tomcat or is not started. Please login to your tomcat manager, and check if the AXTrackv2 application is currently running. If it's present but not running, try to start it. If it's not present, upload the application.

When I connect to the tracker URL, i get the menu but nothing else works. I Keep getting Error 500. I tried to restart the application but this did not solved the problem :
First thing, try the following :

4) Stop the tomcat server and wait a few seconds.
5) Do a “ps aux” on the prompt and check if the tomcat process has really been killed. If not, kill it manually.
6) Start the tomcat server again.

If it solved your problem, then you are done. If not, it's probably a problem with your database server configuration, In this case, look at the database configuration you entered and try to login manually to MySQL with those credentials. If it does not work, then correct your configuration with correct credentials. If it does work, check tomcat's catalina.out logs and see if the exceptions generated contains a clue about your database problem. This is most likely due to a wrong configuration like attempting to connect to MySQL using a UNIX socket.

The tracker seems completely stucks and irresponsive and I want to report that to developpers :
Please ensure you are using an IBM virtual machine then do a “ps aux” to get the PID of your tomcat process. Once known, issue a KILL -QUIT <the_pid_you_just_got>, and check the catalina.out. It should contain the location of a /tmp log files containing a stack trace (and various informations) for all tomcat threads. Just send this file to the developpers along with a detailled description of what happened and what you were doing when it happened.

19 AXEPTool Javascript Rules User Guide

In order to use WS functionalities you have to create an instance of AXP2PManager in your Javascript code.

THIS SECTION IS PRESENT ONLY IN THE CONFIDENTIAL VERSION OF THISDOCUMENT THAT CAN BE ACCESSED BY AFFILIATED PARTNERS OF AXMEDIS ONLY.

20 Bibliography

- AXMEDIS for All document
  - http://www.axmedis.org/documenti/view_documenti.php?doc_id=1728
- Requirements document, The specific requirements are from page 66 to 70 of document http://www.axmedis.org/documenti/view_documenti.php?doc_id=1712
- Use cases document, The Use cases are from page 93 to 102 of document: http://www.axmedis.org/documenti/view_documenti.php?doc_id=1824