Metadata and P2P
optimizing search results in distributed environments

Willem de Bruijn
Leiden University

July 2002

Contents

1 Introduction

The number of people using the internet has grown rapidly over the last couple of years. With this increase in public involvement a shift occurred in networking methodology. Traditionally, there existed a large gap between the suppliers and the consumers of information. In the popular client/server paradigm suppliers would be active all of the time, servicing requests from clients at a fixed location. In an environment where many participants are both consumer and producer of information and no server can be trusted to be active an other way of communicating is necessary.

With the recent growth of available information on the internet and the lack of internal structure linking this data a way of sharing information based not on hierarchy but on equality has to be found. The current situation has renewed interest, both from the scientific community and from the general public, in so called distributed networks. Distributed networks, also referred to as peer-to-peer or p2p networks, supply resources in a decentralized manner, thus acknowledging the fact that no internal hierarchy exists.

Although specific p2p networks have had major attention lately, the general idea is not new. Many proven systems, such as email and internet relay chat, have some form of independence built into their core. The widely popular applications designed specifically for sharing information amongst peers, on the other hand, have only emerged a few years ago. The most widely known implementation is the music sharing network Napster(1).

The strongpoint of Napster was very clear. Many more songs where made available through this network than anywhere else, including live recordings, bootlegs and rare takes. Unfortunately Napster suffered from a severe weakness, namely the chaotic nature of the system. The user had to search through an unordered bag of files.

Napster's irritating shortcomings can be dealt with in many ways. I've started this project about a year ago, when no alternative to the shutdown Napster network had fully emerged yet, with the idea to explore the possible solutions to the problem at hand. Since then many different p2p applications have been created, some of which have also tried to address the problem of ordering their contents. Taking into account both the implementations of others and my own results I'll try to answer the question what methods can be used to improve the usability of large scale p2p networks.

2 Background

2.1 Metadata

When searching for an item one always has to describe certain characteristics that distinguish the desired item from others. These characteristics will then be crosschecked with the characteristics of the possible options to select the best answer. It doesn't matter if a person is searching for a house by asking direc-

tions or if he is trying to find a digital document on the internet, characteristic negotiation is the basic element of searching.

When dealing specifically with data, these characteristics can be seen as data describing the data. For this special type of information the term *metadata* is generally used, where meta stands for 'above'. In everyday situations this information is completely distinct from the object it details. When dealing with digital information, however, it all boils down to numbers. This means that metadata can be seen both as a description of an object and as an object itself.

Traditionally metadata has been separated from objects. The most widely used form of metadata is the filesystem structure used to index digital data on a harddrive. This data is formatted identically for every file and therefore extremely brief. Such an implementation might be sufficient for a local filesystem where a small group of people handle the data, but it is an impractical indexing method for large heterogeneous environments. The large scale on which information is made available on the internet increases the demand for automation of resource identification to a level beyond that of basic filesystems.

Currently a few projects are underway to create a more flexible framework for describing data. Since standardization is essential for any system to become widespread the initiatives backed by international organizations have the highest chance of success. The Resource Description Format (2), backed by the influential World Wide Web consortium, is a semantic layer placed on top of the meaningless XML syntax. In RDF, meaning is given to data by coupling XML formatted content to specialized dictionaries. It is already being used to identify and rebroadcast news messages on the internet and by the Open Directory Project(3) webdirectory. RDF has been designed specifically with the automation of internet resource discovery in mind. The developers expect RDF to become the framework underlining the so called Semantic Web (4), a global information network where computers can autonomously index information.

Despite the effort of various interest groups, metadata is for the larger part currently written in proprietary formats, eliminating interoperability between software systems. Future developers should try to use standardized frameworks as the RDF mentioned above as much as possible if the internet is to remain a global information domain.

2.2 P2P Networking

There are probably as many definitions of what constitutes a p2p network as there are networks themselves. It would be unwise to follow a very narrow definition, since the underlying network isn't our primary concern. Therefore I've chosen the following definition as a basis for our understanding of the term.

> P2P is a class of applications that takes advantage of resources – storage, cycles, content, human presence – available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS

system and have significant or total autonomy from central servers.

(5)

From this definition it becomes clear that using resources at the edges of the Internet gives rise to new problems. Present day Internet protocols were not designed with such a high level of decentralization in mind. Different schemes have been tried to avoid the problems inherent to utilizing edge resources. Since our primary concern is with the optimization of information searches I will restrict my attention to networks that are concerned with content. The most important problem in this field is scaling the bandwidth needs when the network scales. Truly decentralized networks retransmit requests from peer to peer. Since every incoming message is generally rebroadcast to at least two other nodes this leads to an exponential growth in bandwidth demand for a linear growth in connected nodes. After the shutdown of the largely client/server based Napster network many users switched to the fully decentralized gNutella(6) network, resulting in a total breakdown of the network within weeks (7). Since then the network has been altered significantly and continues to grow (8).

Unfortunately little theoretical research has up until now been done in this field. Since network structures can easily be modeled using the formal systems found in fundamental computer science and mathematics one might expect implementations based on such research to be much more efficient than the applications we use today. As a primer, a mathematical analysis of the gNutella network has been undertaken (9). From the results obtained follows that such simple implementations cannot be scaled to potentially millions of nodes. Empirical data obtained by monitoring the gNutella network does give openings for more efficient implementations. One example of such research identifies substructures in the global net (10). Knowledge of a network's structure might lead to search algorithms that exploit the specific substructures to increase overall system throughput and decrease bandwidth usage.

Optimizing search results has a direct connection to the given problem. In an ideal situation all queries are handled in real-time, returning results based on all known resources within a very short timespan. In a fully decentralized network the exponential growth of messages sent between peers can not only seriously degrade overall network performance, it usually leads to a relatively long waiting time before a response can even be generated. Furthermore, it is not certain when a response is obtained whether this response is based on all the resources in the network. This uncertainty follows from measures imposed on the protocol to prevent loops in the message system and the general instability of the network configuration.

Increasing the quality of search results by making extensive references to metadata, as I have implemented, will always induce extra computational overhead. Although future networks might be more efficient in terms of network bandwidth utilization and processing speed, we cannot rely on this for now. How a possibly complex metadata processing layer on top of the network is to be implemented is therefore a second question we will have to ask ourselves. One solution can be caching results as is discussed by (11), but this can seriously

reduce the effectiveness of searches when implemented incorrectly.

2.3 Practical Solutions

Without trying to be exhaustive I will now discuss various real-life implementations of metadata use in peer-to-peer networks. The programs mentioned all contain certain elements that have proven to be useful. At the moment, however, no one program includes all or even a few of these features.

Starting with the most basic functionality, Napster and many of the programs influenced by it allowed only searching on filesystem information. FastTrack(12) is a more advanced descendant of this line of programs, filtering information from the filesystem into a private metadata structure and allowing users to manually add information. The main strongpoint is that metadata seems to be copied together with the main data, creating a persistent label and thereby unifying resources as much as possible. How this is implemented is unfortunately unclear, since no information is made available by the creators.

A completely different approach is taken by the Audiogalaxy application. Inheriting the client/server based query structure from Napster, it completely hides users from one another. The program installed on network edges is a very simple fileserver and -client in one, but all communication and searching is done through a central website. This use of a persistent webinterface not only shields individual users from each other, it also allows for a persistent query infrastructure separated from the client program. A central server remembers all data available on the network over a specified period of time and allows any of these files to be placed in a personal download queue. Only when both a supplier and a consumer of a specified resource become available is a transfer initiated. More importantly, no human interaction is necessary after a request is placed in the queue.

The Audiogalaxy network also features a very detailed structuring of the supplied data. Given the fact that only MP3 sound files are made accessible through the network this task is relatively easy. The high level of knowledge of artists, music styles and connections between them does lead to the assumption that a lot of manual labour has been put into creating and maintaining this information structure. Since no information is made public regarding the client application and the central interface there is no way of knowing to what extent automation in metadata gathering helped to create this system. The client/server approach makes this particular network very easy to use, but also extremely prone to legal attacks, especially since most of the information provided has a questionable legal status.

As mentioned earlier, most p2p applications are based on the Napster approach. Many alterations to the original program have been made, but most advances have gone into the network structure itself instead of in the information retrieval functionality. Taking into account the many problems having to do with the networklayer this cannot come as a surprise. After trying various popular clients the features mentioned above are the only ones that really improve searchability. Especially the Audiogalaxy interface works like a charm. Look-

ing beyond p2p applications we might also find other possible enhancements, though.

The manual addition of metadata, as it exists in the FastTrack software, is not a feasible solution on its own. One technique used primarily in science, but also seen in similar form on public websites, namely peer moderation can improve the quality of metadata, regardless of whether it is manually or automatically gathered. An example I want to explicitly mention is a website harvesting technology related information. This website, Slashdot.org, uses a peer review system that is especially well adapted to categorizing and reviewing large quantities of relatively small messages. I personally know of no other system that has implemented this feature so effectively. One key issue when incorporating peer review is, however, that a heavily involved userbase is necessary.

Adding the techniques mentioned together would be a relatively simple way of increasing search effectiveness. However, there are a few fundamental problems that concerns all mentioned programs. First of all, each program uses a proprietary format for specifying data characteristics. More importantly, no general algorithms is given for finding metadata. Automatic metadata addition basically boils down to reading the filename. Apart from moderation, a case of human intervention, no integrity checking can be applied. As far as automated categorization is concerned, the few algorithms in use today only deal with very specific subsets of information, such as music files in a special format or webpages. Generally these algorithms have been developed behind closed doors, eliminating the possibility of interopting between programs and learning from previous mistakes.

I will now suggest a way to automatically add information to a datastructure in a flexible and extendible fashion. How far we can go with automation and whether it can be practically implemented is discussed later on.

3 Testprogram Specification

3.1 Proposal Revisited

The specification of the testprogram was written down in a proposal at the start of this project. Before discussing the final outcome of the tests I'll first outline the original design of the testsystem. Some aspects of this design have changed along the way. Which parts needed changing and why will be dealt with in the next section.

To be sure our program resembles that what was intended I'll first repeat the research goal as it was written down in the design document:

> Eliminating the differences in database and filesystem access may be the key in allowing more useful sharing of information using digital networks such as the internet as well as in making a local computer system more flexible and maintainable.

> In this project, I plan to research the usability of the distributed part of this idea, using SQL based databases as the underlying datasources. The reason for using rDBMSs is the well thought out interface, based on a question and answer scheme. Instead of using binary large object fields containing data inside the database, only links to files are used, thus disabling part of the flexibility of the system. The local computer system is not altered, saving considerable time.

The formulation differs somewhat from that used in the introduction of this document, but I believe the central issue has remained the same throughout the research process.

I will refrain from copying the complete design document here. The full text can be found in the first appendix. To demonstrate how I planned to obtain my test results it is important, however, to identify the original design goals.

The testprogram was designed to consist of four separate modules: The datasource module, a driver accessing the underlying database management system; the networking module, containing all code necessary to communicate with peers; the spidering module, used to index information and add it to the database and the user interface module, handling user requests by calling the other modules' methods.

The module that influences test outcomes the most is of course the indexing module, since all metadata is gathered by this module. From the first stage of design, I believed a plug-in architecture was necessary here. This need comes from the diversity found in possible input data. Separating the general purpose application from the specialized indexing code allows third party modules to extend product functionality and allows competition amongst product developers in real world deployment. Many complex applications allow extensions to their system to accommodate specific user needs. Examples are Adobe Photoshop, Autodesk Autocad and Microsoft Visual Studio.

Data fed into the system has to be processed by the indexing module to create appropriate metadata. This information is then added to the database through calls to the datasource module as an intermediate. The need for intermediate code comes from the idea that it should be possible to change the back-end, the database, without having to alter the main code.

Queries are generated by a user interface module. Keeping in mind the connection with the underlying database, I suggested that SQL should be used as the query language. For testing purposes, both a commandline and a graphical version of the user interface had to be constructed. This way testing could be underway before a user friendly interface was ready.

Queries are then processed and rebroadcast by the networking module. The networking module processes queries by interpreting incoming packages and calling appropriate datasource methods. The network itself would consist of identical peers with equal rights, leading to a gNutella like fully distributed network, with all the problems related to such naively implemented networks. Please remember, though, that network issues are not our primary concern right

now.

All in all, the complete division of the application into modules should allow last minute alterations and extensions to the system. This would enable us to quickly react to obtained test results. How all this should be implemented was deliberately left out of the design document. Only the use of a relational database management system as back-end, the use of TCP/IP as underlying network environment and Microsoft Visual C++ as development environment was specified.

3.2 Alterations

The final version of our testprogram resembles the design as stated in the previous section rather closely. Since the testing started very early in the development cycle, I did find some problems relating to the design of the system rather quickly. In the introduction I mentioned the need to adhere to standards for allowing interoperability between software systems. Especially the communication between peers should be open and standards-based as much as possible. One can argue that relational database management systems based on the SQL query language form a standard, but unfortunately SQL is rarely used as a communication protocol on the internet. The most prominent change from the design stems from this insight. More and more software systems use XML(13) as the grammar for their communication and many development kits exist to accommodate the use of XML. Because of this, I've decided early in the development cycle to focus on using XML and other webstandards based on this, such as RDF, as much as possible.

Using XML and RDF doesn't automatically rule out the use of a rDBMS as back-end data repository. I did remove it, however, in favour of a direct XML connection to speed up the development process.

Another change having to do with the use of XML has to do with the user interface module. The main standard on the internet of showing data is the hypertext markup language, which itself is based the XML grammar. The fact that our data is already encoded using XML gives us the opportunity to quickly create user output in a format recognized by any webbrowser. The interface to the data indexing part of the application is still developed as a $Windows^{tm}$ application. Ideally, administration and file-addition should also be made possible using webtechnology. The necessary added cost of implementing security has kept me from adding this functionality in the current program, however.

The resulting application thus consists of the four modules as detailed, but with the added webinterface used for querying the network.

4 Testprogram Implementation

The main characteristics of the testprogram I've created have already been dealt with. There are many relatively small implementational issues that also

weigh quite heavily on the outcome of the testing and aren't part of the overall design. I'll quickly scan through the most interesting here. Many of the problems I've encountered are, I believe, not only applicable to our problem, but can be encountered in all sorts of programs. On the other hand, not all the choices I've made can be considered wise when seen afterwards. Both the strongpoints and the worst weaknesses can be instructive for future development. I also want note that all sourcecode is made publicly available at *http://atoms.sourceforge.net/* under the GNU General Public License.

4.1 Datasource Module

The datasource module in our test is basically a wrapper around Microsoft's XML Parser(14). The main disadvantage of implementing a possible large datastructure in such a way lies in the poor processing speed. Using a closed source API such as the XML Parser adds the extra problem that it is uncertain where speedups can be implemented. During development different versions of the parser have been released. These all had several speedups over previous releases but just as many slowdowns due to increased standard compliance. More practical issues occurred when doing large scale stresstests such as excessive memory requirements. Looking back at the project I have to say that using a simple document as datastorage is far from efficient. An alternative that is probably the best choice for future use is an XML database. XML databases can probably overcome the practical issues I've encountered while allowing us to keep using XML as our primary communication method.

The format used to write file and plug-in data is modeled to the RDF. An RDF dictionary for standard file data called the Dublin Core is used to give meaning to items. Too few dictionaries currently exist to fully exploit the use of RDF at this moment. Therefore I haven't tried to fully comply to the RDF standard. I believe that developers should try to comply to the standard as soon as specialized development tools and generally accepted dictionaries emerge.

4.2 Indexing Module

The indexing module consists of two relatively separate parts, namely the framework that allows calling of plug-ins and the plug-in architecture itself. I will therefore discuss the two parts separately.

4.2.1 framework

The indexing framework has been altered quite a bit according to test results obtained during development and the incorporation of a plug-in architecture. These recurrent alterations resulted in a final framework that has as small a codebase as possible. Basically the framework consists of two routines. The first routine is run every time the indexing module is loaded and deals with identifying available plug-ins. The second routine consists of a loop where plug-ins are loaded upon use.

The main challenge in designing such a system is finding a reasonable tradeoff between system speed and plug-in utilization. This resulted in a dynamic and - somewhat - intelligent system, where plug-ins are added to the database during the discovery phase as if they were data items. The only difference lies in the identifying tag. Adding an item into the hierarchy at subtree X implies that plug-ins registered at X or at ancestor nodes can find extra information. These plug-ins are called successively, each rewriting the data in the hierarchy. This recursive process stops when no more useful plug-ins are found. To start this process we have to bootstrap the system. Bootstrapping is done by adding a plug-in to the root of the hierarchy. Any data item added to the tree is part of the hierarchy, resulting in the calling of this plug-in. It is elementary that this plug-in should be as general as possible. I've therefore created a plug-in that reads filesystem information only.

I'll illustrate the indexing process with a simple example. Let us consider a .mp3 audio file. The bootstrapping plug-in creates an entry in the filesystem subtree. This activates the content sniffing plug-in that adds an entry in the audio/mpeg subtree. Since we have an mp3 specific plug-in that is registered under this subtree this plug-in will then be called. Another plug-in may then be called using the information obtained during the mp3 specific process. Etcetera.

A problem that may occur with this recursive process is that a loop in the data flow can be created. This problem has not been dealt with explicitly in the testprogram, since too few plug-ins currently exist to create such a loop. Please note that it should of course be dealt with in real life products where more plug-ins are used.

4.2.2 Plug-in architecture

The plug-in architecture used is a very straightforward one. All plug-ins are libraries that can be loaded and released at runtime. After having loaded a library the framework simply executes one of two possible functions. The first function returns an XML datastructure while the second simply returns XML formatted data as a text string. Having separate functions allows the plug-in develop to choose if he wants to work with the Microsoft XML Parser or not.

Since this is a testprogram only a few plug-ins have been created. The most simple one reads data from the filesystem and creates an entry in a subtree that models the filesystem. The second plug-in uses file content to identify the filetype and creates an entry in a MIME based hierarchy. This module is currently a simple wrapper around a function exported by the *Microsoft Internet Explorer*^{tm}. It identifies some 30 different types. A third plug-in is created specifically for reading the proprietary metadata format often used in .mp3 audio files called ID3. This plug-in does not add items to the hierarchy, but simply alters and extends existing entries.

The last plug-in best demonstrates the strength of the system. This plug-in creates keywords from the already added information and with these keywords queries an internet directory. The directory queried is currently the google directory (directory.google.com) that is equivalent to the open directory project.

From the answer a local version of the directory is created containing the processed files. This is where automatic categorizing becomes difficult. The results given by the website is dependent on the keywords we selected and on the results of the earlier work overall. The test results, including this categorizing, is discussed in the next section. Whatever the quality of the results obtained with our system it is probable that it can be improved when domain specific directories are also incorporated for areas where these exists.

4.3 Networking Module

The networking code is implemented as straightforward as possible. An application has to be instructed where it can find one other node by specifying an IP address. After this the node recursively scans the available nodes by requesting the addresses of all neighbours of the already known nodes. This discovery process is implemented in a separate thread that continually runs in the background as long as we are connected. Nodes are selected as direct neighbours indiscriminately until all available slots have been filled with active nodes. The number of slots can be specified as an option by the user.

The first problem with this implementation is that creating direct neighbours from the neighbours of the first node encountered will results in a high degree of locality. In extreme cases this will reduce a network to a large number of distinct islands of subnets. Furthermore, there are far too much edges in the net from any connected node to another, leading to a lot of overhead. The existence of local subnets has been identified by (10).

A few rudimentary safety features have been implemented to reduce the network traffic. First of all, a message received directly from a webclient is given a pseudorandom codestring to identify it across the network. This ensures that a network node will not rebroadcast a message more than once, thus eliminating loops from the network. Second, a time to live (TTL) variable is added that is decremented at every node. The TTL ensures that a message dies out after a certain amount of time. The downside of this is that there exists the change that a query is not being sent across the entire network. Our current implementation uses a TTL of 32, which would allow more than 4 billion nodes to receive the message if each node has only 2 direct neighbours. I therefore expect, although I don't have any proof for this, that the TTL is rarely used to stop a message from being resent.

The nodes basically have one receiving socket, the server, that listens to standard HTTP requests and as much sending sockets, clients, as there are direct neighbours in the active configuration. The clients rebroadcast the original HTTP message together with the added TTL and identifying string if allowed. Using standard HTTP basically creates a webserver. The gNutella network also sends messages using HTTP, but does not actively support webclients. Our testprogram, on the other hand, sends results in XML with XSLt and CSS extensions. XSLt documents contain translation rules. In our case this extra document tells the receiving client how to transform the raw XML data into a user friendly HTML webpage. The other link, a Cascading Style Sheet, adds

some extra customization of the webpage. Because we simply send raw data to the user most processing of data is executed at the webclient's local machine. This should increase the maximum network throughput. A secondary advantage of this scheme over the use of hardcoded parsing rules is the flexibility of data output it entails. The sending of XML allows other programs to reinterpret the data and webmasters to present it to their users in a customized way.

4.4 User Interface

The user interface is divided in an administrator part and an anonymous user part. The interface used for administering the system and adding content to the database is part of the executable. It uses the *Microsoft Foundation Classes* as basis. The codebase for this interface is very small, since all actions occur in the other modules.

The user interface developed for searching and browsing the network is geared especially towards the everyday user. The anonymous surfer will not need any knowledge of the underlying network. Using standard webtechnology gives our network the look and feel of a standard webserver. The added benefit of this is that how data is displayed can be altered by the administrator of the webserver. Any node in the network can thus be seen as a portal, seamlessly coupling its local static website to the dynamic content of the p2p net. This way we mix static and dynamic data, websites and clientnode local files into one seemingly homogeneous environment that can be accessed with everyday technology.

Exporting raw data to an networked environment using XML and thus separating information from visualization is considered the next step in networking. In the last year a de facto standard has emerged called Web Services. This model, based on XML and a transport protocol called SOAP, is being backed by such influential companies as Microsoft and Sun. Web Service communication is in the process of being standardized by the World Wide Web consortium in the Web Services Description Language (15). Since Web Services are only beginning to emerge and few tools currently exist I've use a proprietary format based upon the same principles for my testcase. I do urge future developers to adopt WDSL as their product environment as soon as the details have been worked out.

5 Results

Due to the nature of our research question I believe it is impossible to obtain valid quantitave results. The quality of obtained results depend as much on the question asked as they depend on the tools used to index data. I will therefore avoid creating statistics that appear tentative.

As alternative I'll to try to explain the results obtained by looking more closely at the algorithms implemented. It is relatively easy to create an algorithm that gives good results for questions in a small domain. General purpose

algorithms are a lot harder to come by. By comparing not only the results but also the way we obtained them we should be able to filter out the more general solutions. This will also point us to shortcomings that would remain invisible if we were only to compare results. In the next section potential improvements are presented based on these shortcomings.

5.1 Quality of Results

The results received when testing the network quality where very dependent on the nature of the question. Naturally, queries that deal with filesize, -date or -type were answered very good, since this 'hard' metadata can be learned quite easily. What we want to improve are what I shall call second-level questions, questions that deal with less technical issues. For instance, the question "return all documents that are related to artist X" depends largely on how well the indexer was able to categorize documents according to the original artist.

For the second-level categories I've created two plug-ins. The .mp3 specific plug-in will read any infile tags written in a de facto standard if they are present. The google querying plug-in tries to place files into a directory modeled to the open directory project based on earlier results. I'll handle their results separately.

Quite a few mp3 files found on the internet contain metadata within the file. Because of the specific streamwise encoding of audio in the mp3 format extra information can be added anywhere in the file. This metadata usually contains the original artist, title and album number, giving us plenty of input to use for further automatic indexing. The percentage of successfully categorized entries that contained this formatted metadata approached 100%. Only when the artist info did not reference a single artist or when this artist was not known by the webdirectory did our indexing fail. All in all I think we can say from this that creating plug-ins to read out proprietary metadata formats greatly increases chances of success. This should not come as a surprise.

The google indexer creates keywords from the already created metadata and simply calls the google directory through its webinterface. Whether this returns a correct subtree in the directory is based on two factors: the possible keywords at hand and the selection process we apply to these keywords. The first factor cannot be optimized, since this is dependent on the input from the other modules. We can only make sure that the google plug-in is called after all other (non-web) plug-ins have been called. Finding the correct keywords does make a large difference. Since most raw data is still taken from the filesystem this can be seen as the primary input. When other input is available a spectacular increase of successful results can be noted but this is rarely the case. Finding correct keywords therefore consists mostly of breaking down the filepath and name into subsections and selecting the ones that are most probably of interest to us. The selection algorithm I used is based on some elementary heuristics, for instance that the first part of a filename will describe the name of the author or artist. Using these hardcoded heuristics returns very good results when the user has named his files as expected. Unfortunately this is often not true. Many times

the original artist of a number of files is written in the directoryname. A lot of files cannot even be traced to a single artist, for instance business documents.

I can easily say that the testprogram categorized my audio and video collection very well. Then again, I programmed the system to read filenames in this fashion. To extrapolate these correct results to the general case is a lot harder. One way of doing so would be to ask minimal user input, for instance that the user specifies which convention he uses when naming files and directories. Another solution would be to simply try multiple combinations of keywords with the google directory. I have tried this during testing, but removed it due to the very slow response time. Each lookup cost about 1 second, so doing multiple lookups per file proved to be very impractical. Recently google.com released a web services based API for accessing the system which could speed up the process, making this a viable solution. A third option would be to exploit the locality of files. If one file in a directory is indexed successfully, this information could be used for other's as well. Basically all these small heuristics boil down to the same idea. To successfully extract information from a file's path and name the programmer should somehow reverse engineer the user's naming algorithm. Many schemes can be found that will work for some set of files, but only human interaction will consistently help us select the most useful data. I therefore suggest using a learning algorithm where users can give feedback on the acquired results. This way minimal intervention is needed, but adaption to a personal indexing method remains possible.

5.2 Performance

Although performance was not a very important issue during testing we cannot deny its significance in the real world. The implementation I created was absolutely not fitted for widespread use. Especially the parsing of the XML document proved, even with specialized parsers, to be a very time-consuming task. Large scale networking tests could not be run due to the lack of resources. From other research mentioned in the first section we can conclude that our implementation cannot scale reasonably either. These two deficiencies represent by no means absolute truths, however. It only indicates that a lot of work will have to be put into optimizing network code and improving data throughput.

Increasing data throughput can be achieved relatively easy by using specialized database management tools for storing and retrieving data.

Optimizing network usage is less simple. A client/server architecture is definitely the fastest implementation for any query based system. This model relies on dedicated servers that a distributed architecture lacks, however. Which model gives the best trade off between overall responsiveness and resource utilization spreading is currently unknown. To my knowledge no tests have been done to identify how virtual structures, such as rings, hypercubes and freeform networks layered on top of a heterogeneous physical structure alter the performance of the environment. Clearly, more research need to be done to identify the different strongpoints and weaknesses of possibly useful topologies. For now, we can only look at existing implementations.

6 Potential Improvements

In the previous section many weaknesses to my testsystem have come to light. In an ideal world I would keep on improving the testprogram until all problems were dealt with. This is not an option in practice. I will now give some pointers to how my system might have been more successful. No hard proof exists to back these ideas, naturally.

6.1 Indexing Metadata

One of the issues that became obvious during testing was the lack of metadata persistence. For results to improve we have to be able to learn from our mistakes. The testprogram implementation uses a webclient interface for already specified reasons. A drawback to using this is that metadata has to be recreated at every instance of the network. Other implementations do not have this problem, since they have a proprietary interface. Metadata persistence is easily obtained by placing the metadata somewhere between the core data, but this will cripple many filetypes. Regardless of the advantages a webclient based system may have over platform specific interfaces, I'm afraid the lack of metadata persistence, and therefore the lack of metadata improvement, is such a serious drawback that, until an alternative is found, a specialized interface is necessary after all.

Unless many specialized plug-ins are created, metadata has to be gathered from filenames. In order to receive useful results without a plug-in for every type of file it is necessary to optimize the keyword selection algorithm. I expect the most promising idea will be based on a learning algorithm in some way. How this should be implemented is a question that can't be answered so easily. I believe this is a subject that should be examined more closely.

No indexing method can be proven to be successful, since input is always human created data and therefore inherently unreliable. Finding heuristics based on human computer interaction is, I think, the only way of solving this problem. I will give two such heuristics as example. First, exploit the fact that people order their data in a to us unknown yet most probably rational manner. Look at the directory structure and try to find congruent features of elements within certain subsets. The less elements in a directory, the more they usually will fit together in some way or another. Second, many people create, somewhat unconsciously perhaps, a personal encoding scheme for any subset of data. For instance all work related files are numbered with their project code, all music files start with the name of the artist or all files in the favorites directory reference webresources. These rules of thumb I've just given are by no means revolutionary nor are they the only ones to be found. They are simply examples of everyday practice one can learn from personal experience that can be used to broaden the way metadata input can be dealt with.

6.2 Distributed Networking

I've mentioned that little empirical research has been undertaken to gain knowledge of the connection between network topology and system behaviour. The network I'm going to suggest here is only based on existing implementations that proved to be successful in present day applications.

First of all a tradeoff between a fully distributed and a client/server architecture has to be found. Both systems have advantages and disadvantages, so a crossover that minimizes the weaknesses and maximizes strongpoints is what we should be looking for. I suggest an adaptation of the FastTrack network. This network consists of a relatively small set of highly connected nodes that cache results from other less responsive systems. The caching scheme can seriously reduce overall network traffic and increase responsiveness, while the segregation of the nodes according to the quality of the resources will ensure no unnecessary bottlenecks in the network occur. Frequently reviewing the state of the network increases robustness and allows for an automatic stabilization in the dynamic environment. A generalization of the 2-level hierarchy to more illusive structures based on the same dynamic principle could reduce overhead even further.

Alternatively, the system can be modeled after the Alpine(16) network. This implementation does not rely on rebroadcasting by peers. Instead, a node broadcasts (or multicasts if possible) its message using the unreliable UDP protocol to all known nodes at once. Minimizing the size of these initial messages together with the lack of indirection could very well make this a viable method of communication initiation. When a node has an answer it can then set up a reliable direct link using for instance TCP. This system might work especially well when reducing the number of nodes that may respond to the initial message. For this cause a caching scheme could be implemented. It is then basically reduced to the previous system, but without the complicated bottom up query forwarding.

6.3 User Interface

Most modern operating systems export hooks to add information directly to the filesystem view. Under Unix the
proc filesystem is a good example. Under Windows shell extensions can be used. Exporting the browseable directory of a local system or even a network as a standard directory view structure is a simple way to even further incorporate intelligent data sorting into everyday computing.

7 Putting it together

I've presented a selection of applications and created a testprogram to further research possible inclusions of metadata into p2p applications. The ideas incorporated in these various systems may very well help us tame the enormous amount of data p2p networks are expected to make publicly available. The main problem with current implementations lies in the fact that interoperability between systems is seriously neglected. Standardizing work underway will

hopefully solve this problem, allowing us to fully exploit the power of these technologies within the next few years. I propose to combine a number of the discussed technologies. The integration of these methods is based on the test-program's modular architecture.

Gathering metadata is based on domain specific heuristics, since many unrelated types of data have to be processed. A software system should therefore be flexible enough to allow the addition and alteration of algorithms. This can be accomplished by refining the indexing engine presented in the tests. The initial metadata found can be extended by referencing publicly available databases. One example given was the google enabled plug-in. Further automatic refinements may come from crossreferencing items, as presumably used in the Audiogalaxy network. Allowing both manual alteration of metadata as found in KaZaA and moderation techniques as used on Slashdot can to some degree remove the errors created by the automatic process. Refining preliminary results will, however, only be of use when metadata can be made persistent across the network. So far, only the KaZaA and Morpheus applications employ such features.

To allow rapid interoperability between networks, it is imperative that already widespread communication standards are followed. Formatting data as XML or even RDF is a start. The use of a web-based client has a similar advantage over OS specific user interfaces. The lack of roaming metadata features severely reduces the strength of this option for now, unfortunately.

8 General issues

The problems I've encountered and solutions I proposed only deal with a subset of the problems one can encounter when creating a P2P application. To create a usable and scalable system many other issues will also have to be resolved. I will briefly touch upon these issues to show the complexity of the field. It must be said, however, that this section is slightly outside the scope of our research.

These problems mainly deal with the infrastructure of P2P networks. While early networks used excessive amounts of bandwidth for network management alone this problem is slowly being dealt with (17). However, the current situation is still far from ideal and can seriously harm future growth (18).

Luckily, P2P technology has seen a new enthusiasm in the last few years from the academic society. If anywhere it is here that we can expect well-founded solutions to emerge. Counting not only on intuition, but increasingly on mathematical proofs, researchers have so far been looking into network topology and ways of exploiting this structure. Examples of this can be found in papers proposing slight adjustments to existing protocols (**?**) and others that detail new protocols based on other topologies (19).

A field of research I believe interesting solutions can be found is that of ad-hoc networking. Ad-hoc networks are wireless networks lacking a centralized backbone infrastructure. They therefore exhibit the same dynamic behaviour of P2P nets. Especially solutions dealing with node discovery and intelligent

routing can be transferred to P2P applications. Routing protocols geared especially towards a dynamic environment that can be used in P2P applications are for instance the Ad-hoc On-Demand Distance Vector Routing protocol (AOVD) (20) and the Dynamic Source Routing Protocol (DSR) (21). A comparison of a number of protocols has also been undertaken (22).

9 Conclusion

A development path has been presented for the near future. Due to the nature of our problem no hard evidence can be presented to prove the success of such a system. The main goals mentioned, namely interoperability and extensibility, are the key of the suggested system and should always be followed, however. This practice is already common in established fields of computer science and application development. It is widely accepted that such goals are the principles of engineering. The lack of it therefore shows the infancy p2p and metadata are in. When these guidelines are followed the rapidly spreading p2p networks will not only remain as searchable as they are. They can become the key to narrowing the gap between local and networked resources.

10 Summary

Peer to peer networking is currently in its infancy. It is, however, expected to become widespread in the near future. The problem facing us is that these applications will become unusable as their popularity increases. I present a selection of technologies that control the flow of information by utilizing metadata. The characteristics of these methods and what problems a developer can expect during implementation are discussed in depth. Finally, a combination of methods is suggested to maximize usability in a practical manner.

11 Bibliography

References

[1] "Napster website.". Available from World Wide Web: `http://www.napster.com/`.

[2] O. Lassila and R. R. Swick, "Resource description framework(rdf) model and syntax specification," tech. rep., World Wide Web Consortium, February 1999. Available from World Wide Web: `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`.

[3] "Open directory project website.". Available from World Wide Web: `http://www.dmoz.org/`.

[4] T. Berners Lee, O. Lassila, and R. R. Swick, "The semantic web," *Scientific American*, vol. 5, 2001. Available from World Wide Web: `http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html`.

[5] C. Shirky, "What is p2p... and what isn't," November 2000. Available from World Wide Web: `http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html`.

[6] "Unofficial gnutella website.". Available from World Wide Web: `http://gnutella.wego.com/`.

[7] A. Pava, "Gnutella is going down in flames," September 2000. Available from World Wide Web: `http://zdnet.com.com/2100-1107-524101.html`.

[8] K. Truelove, "Gnutella: Alive, well, and changing fast." January 2001.

[9] J. Ritter, "Why gnutella can't scale. no, really.." February 2001.

[10] M. A. Jovanovic, F. S. Annexstein, and K. A. Berman, "Scalability issues in large peer-to-peer networks - a case study of gnutella," tech. rep., University of Cincinnati, 2001.

[11] K. Sripanidkulchai, "The popularity of gnutella queries and its implications on scalability,". Available from World Wide Web: `http://www-2.cs.cmu.edu/~kunwadee/research/p2p/gnutella.html`.

[12] "Fasttrack website.". Available from World Wide Web: `http://www.fasttrack.nu`.

[13] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, *Extensible Markup Language (XML) 1.0 W3C Recommendation*. World Wide Web Consortium, 2 ed., October 2000.

[14] "Microsoft xml parser website.". Available from World Wide Web: `http://msdn.microsoft.com/xml/`.

[15] J. C. Schlimmer, *Web Service Description Requirements*. World Wide Web Consortium, draft ed., April 2002. Available from World Wide Web: `http://www.w3.org/TR/2002/WD-ws-desc-reqs-20020429/`.

[16] "Alpine network website.". Available from World Wide Web: `http://www.cubicmetercrystal.com/alpine/`.

[17] E. Adar and B. A. Huberman, "Free riding on gnutella," *First Monday*, Sept. 2000. Available from World Wide Web: `citeseer.nj.nec.com/article/adar00free.html`.

[18] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," 2002.

[19] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proceedings of the 2001 ACM SIGCOMM Conference*, pp. 149–160, 2001. Available from World Wide Web: `citeseer.nj.nec.com/stoica01chord.html`.

[20] C. Perkins, "Ad-hoc on-demand distance vector routing," 1997. Available from World Wide Web: `citeseer.nj.nec.com/article/perkins97adhoc.html`.

[21] D. Johnson, D. Maltz, and J. Broch, "Dsr the dynamic source routing protocol for multihop wireless ad hoc networks," 2001. Available from World Wide Web: `citeseer.nj.nec.com/johnson01dsr.html`.

[22] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Mobile Computing and Networking*, pp. 85–97, 1998. Available from World Wide Web: `citeseer.nj.nec.com/broch98performance.html`.

[23] "World wide web consortium (w3) website.". Available from World Wide Web: `http://www.w3.org/`.

A Design Document

The design goals for the atomsnet project have been laid down in a proposal document in December of 2000. I've added this proposal below.

A.1 Proposal for a college project

- Introduction

For a college project of 10 dutch 'studypoints', I’d like to build an information retrieval system which combines the distributive approach of file-sharing utilities and the logical framework built into relational database management systems (rDBMS’s).

With the combination of these two users can do more than simply swap large quantities of static data (files) by searching in their filename. The use of a DBMS enables anyone to include a lot of information about a piece of data. It can, for instance, be linked with other items in one or more ‘equivalence’ groups or specific commentary and keywords might be included to increase the hitcount. Textual data might be indexed by a spider for retrieval and other ‘intelligent’ features could be included to create a system which is much more flexible than the current file-based distributed sharing systems.

- Background

Since the line between the internet and local data is becoming ever more fuzzy, the difference between finding data on a local system and the network is also disappearing. For about half a year I’ve been working on an document for building this ‘bridge’ at http://atoms.htmlplanet.com[1].

On the user’s local computer information is seen as static packages (files) that are accessed in a hierarchical manner. The internet, on the other hand, is built out of websites, consisting of many files and even information from databases. Although some information on local computer systems are also kept in databases, most of it is not.

Eliminating the differences in database and filesystem access may be the key in allowing more useful sharing of information using digital networks such as the internet as well as in making a local computer system more flexible and maintainable.

In this project, I plan to research the useability of the distributed part of this idea, using SQL based databases as the underlying datasources. The reason for using rDBMS’s is the well thought out interface, based on a question and answer scheme. Instead of using binary large object fields containing data inside the database, only links to files are used, thus disabling part of the flexibility of the system. The local computer system is not altered, saving considerable time.

- Current development in this field

[1] See URL http://atoms.htmlplanet.com/

The project I propose to develop is not unique in this field. Although I know of no other project with exactly the same goals, there is some quite interesting work going on in using relational databases to index a harddrive and in distributive information sharing. In the first field the next projects might be interesting, although rDBMS manufacturers are also working on projects like these:

- TUNES[2]"[..] we want to replace filesystems with orthogonal persistence of objects, compilation and administration of program binaries with automatic consistency management of dynamically optimized code, interface-driven programming with structure-driven interfacing, explicit manual networking with implicit automatic distribution. [..]"

- Oracle iFS[3]"[..]From the end user’s standpoint, Oracle *i*FS appears as any other file system accessible through Windows networking, a web browser, FTP, or an e-mail client. However, unlike other file systems, Oracle *i*FS stores all your files from web pages to e-mail, from spreadsheets to XML files in the same file system. [..]"

The second part of the project deals with developing a distributive persistent ‘information cloud’ in which people can find the information they want in by querying this system. The term under which these applications are known is peer-to-peer networking. Some well known projects are Napster[4]and gNutella[5].

- Modularity

Since it is wise to modularize a programming exercise of such size I devided the entire system in a couple separate parts. These will be discussed individually below.

The main core of the system is the datasource each user has. This will be built out of standard DBMS technology, so that SQL queries can be used. For the windows platform a Microsoft Access database would probably be preferable, although some work on choosing the right rDBMS should still be done.

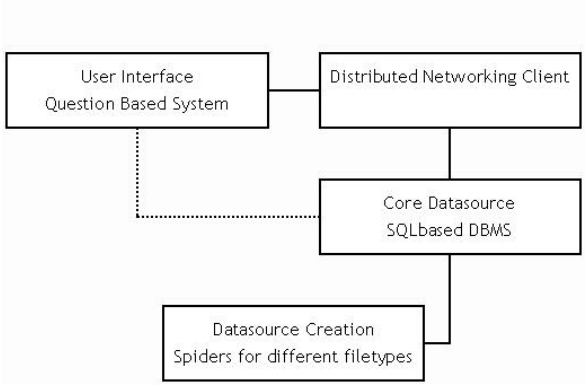
To fill the database with useful information, spiders can be used (the term ‘spider’ is used in internet searchengine technology) which can crawl through local files. For each specific filetype a different spider should be developed. This makes it necessary to use a plug-in portal for these tools.

The networking should rely on the TCP/IP protocol, since this is the most common networking protocol today. A lot of data-swapping and information synchronisation techniques I would like to include can already be found in file-sharing utilities such as Napster, iMesh and gNutella. Since the last of these is open-source I will probably use a lot of gNutella code for creating the data-sharing layer.

[2]See URL http://www.tunes.org/
[3]See URL http://technet.oracle.com/products/ifs/htdocs/iFSFO.html
[4]See URL http://www.napster.com/
[5]See URL http://gnutella.wego.com/

On top of the simple file-sharing layer an interface will be added which allows for detailed questioning of the current ‘datacloud’ based on the SQL used in the core.

You can see the way these modules connect below:



- Module 1: Datasource Module

The datasource module can be seen as the core of the system. Containing all the information the system knows of, it is the central part of the information. The datasource can be built out of a relational DBMS and a port to this system which acts as an information server. The port queries the individual database using standard SQL for information and acts on behalf of the local user, a remote user and based on programmed synchronisation of information in the information cloud.

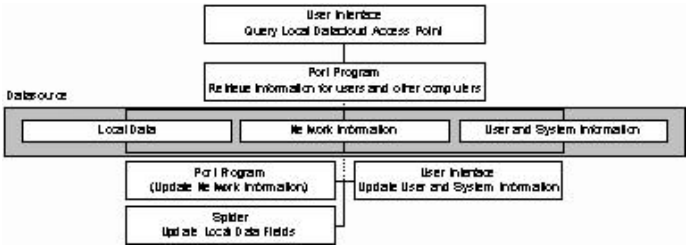
The database should at least consist of these parts:

- One or more tables defining the local system and configuration options. This allows for communication with end-users and other connected databases. This data is set by the user interface program.
- One or more tables defining the entire information cloud. This realtime data is set by the port program.
- Tables defining the local filesystem and all indexed files. Which parts of the local filesystem are shared publicly or protected should be known to the system by special columns in the tables. This is the part the filesystem comes in handy. Every file could get it’s own table with predefined colums such as size, name, type, creation date, sharing status, keywords and description. User defined columns should also be allowed here, making a flexible system based on it’s spiders.
- Views or tables connecting the data above into logical groups, based on the above columns or a user-specified grouping. These user specified sets

may be a project such as this, a cd made out of mp3’s or other to the user logical groups. Every group should have extra information based on the scheme depicted at the filetables above.

- Views or tables consisting of frequently used information in the datacloud by local users. This creates a view based on the information cloud per user. Frequent reindexing of this information is required to keep the local and the remote system up-to-date.

Graphically, the datasource is internally and externally connected as shown below:



- Module 2: Distributed Networking Module

The Distributed Networking Module is described above as the port program. It will be the port between the local and the remote information and should be invisible from the user’s point of view. Running as a process in the back, this program allows or denies computers trying to connect to the network by connecting to this system, frequently updates values such as number of connected computers and items of information online. On behalf of incoming request from other programs it can query the database and return information based on the results or bounce the request to another known system.

The use of SQL is especially important for this part of the program. Since SQL can be ‘understood’ by a computer program, we can base decisions on incoming queries, and deny, allow or alter these queries before actually running them. The port program should also have security features installed, such as blocking of requests from users with too little access priviliges and perhaps a blacklist based on pryor behaviour.

Furthermore, the port-program has to create standardized output based on the results it receives from the database.

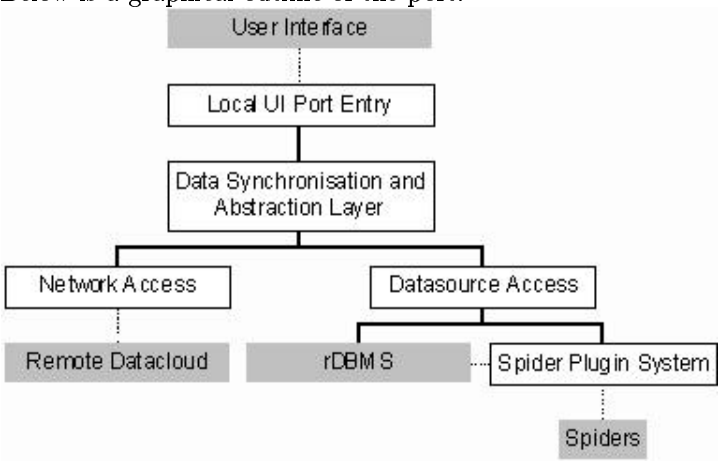
Although not needed, it might be a good idea of adding a grouping feature based on unique numbering. This way a port can be member to specific groups of friends, such as ports in the same building or speciality interests of users

controlling the port. Belonging to a group will allow for sharing of protected data, while otherwise only public data will be shared.

On the networking level the program should consist of TCP/IP protocol access with a HTTP layer. Creating an efficient data-exchange language on top of TCP is too much work to my opinion and the HTTP layer allows for standardized and reliable networking.

The last part of the port consists of access points to spiders. The seperation of port and spiders allows for competition in spider-design and expansion of the versatility of the system without having to change the port itself. I will have to look through WinAmp, Netscape or Internet Explorer documentation for designing such a plug-in system.

Below is a graphical outline of the port:



- Module 3: Spiderplatform Module

Although the spiders connect to the port through a plug-in system the spiders themselves are part of the port. The separate spiderplatform is used because third-party users might want to create spiders specific to their tasks if this project gets to the point of people playing with it or using it.

Because of this design issue, separate standardization on these plugins should exist. For now, I’m thinking of dynamically loaded libraries whose member function executing the indexing is called by the port-program on a user-defined basis. The output they generate is of course based on the datalayout of the database and therefore I believe it is too early to discuss it here.

- Module 4: User Interface Module

The last module deals with the human-computer interface and should be a separate program that calls on the port program to get information. Seperating the UI from the core leaves for many different implementations based on user needs. For this program I propose the following:

Two UI programs, showing the diversity of the system, serving two different groups of users. The first program, which can be used for debugging of the port

program will be a simple SQL front-end. The user has to type in it’s queries and retrieves views based on these queries, plain and simple.

The second program can then be a user friendly front-end, showing the user which standard queries the system usually answer, such as a search based on filetype and name. The direct SQL is thus made invisible to the user. Specific implementation of this system is unclear at this point and further research should be done on this point (I believe a course called ‘Interfaces’ is given at the LIACS this semester which I intend to follow if it deals with this subject).

- Development Platform

The platform on which I’d like to develop this project is Windows NT or Windows 2000. The reason for this is the following. I have no knowledge about other computer platforms than Windows and Unix and of these two I find the first both developer- and user-friendlier.

The development package I would prefer is Microsoft Visual C++, based on previous experience with this system.

- Timespan and planning

For a project of this size some planning might be useful. Below is the timeline of the program I propose:

Startdate : Enddate (DD/MM)

Workplan for this timespan

08/01 : 19/01

Writing Project Proposal

20/01 : 02/02

Requirement analysis and more precise specification of the different modules based on this research.

03/02 : 09/02

Creating and documenting the Database model, with standardized table and view-definitions.

10/02 : 02/03

Creation of a local port with spiderplug-in system and datasource connection in place.

03/03 : 09/03

Creation of a very basic user interface to test the local port

10/03 : 30/03

Updating of the local port and adding of remote access facilities.

31/03 : 12/04

Creation of spiders for text and webpage indexing and for simple filesystem indexing based on filename.

April 12^{th}

First demonstration version should be finished.

13/04 : 11/05

Documentating the system so far and indexing of shortcomings. Altering the system based on this documentation.

12/05 : 01/06

Creation of a second User Interface, with SQL abstraction.

02/06 : 15/06

Reserved for delays and final documentation updates.

June 15^{th}

Demonstration of final program.

B User Manual

The atomsnet application has been designed to be easy to use for beginners, yet extensible enough to be interesting for more advanced computer users. In the introduction I will give an overview of the most basic functionality and how one can use the system 'out of the box'. More in-depth information on extending and personalizing the system is discussed in the other sections. That said, the introduction alone should suffice to help most people on the way.

Many different versions of the application have been released in the public domain during the development phase. Since many alterations have been made along the way I cannot describe every feature of every version. I will therefore restrict myself to the versions regarded stable. This manual therefore discusses solely atomsnet versions 1.0x, where the x can be any number.

B.1 Introduction

B.1.1 Prerequisites

Atomsnet relies on the Microsoft XML Parser version 3 or higher. This means that you need to have this package installed on your computer prior to running atomsnet. If atomsnet cannot open the XML parser it will show a message in the message window. Newer computers will have no problem, since the MSXML parser is part of Internet Explorer 6. If you do have an earlier version of internet explorer installed you should either upgrade the entire internet explorer package, or you can download the parser itself from microsoft.com

Atomsnet will only run on the Microsoft Windows operating system. It has been tested with Windows 98SE, Windows NT4 and Windows 2000 on Intel based computers.

B.1.2 Installation

Installing the application is very straightforward. There exist both a sourcecode download and a binary version. Application users should choose the latter package. This version of the application comes with a complete installer/uninstaller and can be installed without hassle within a few minutes.

After downloading the package, simply start the installer. During installation a few questions regarding package placement are asked after which the

atomsnet application is installed on the user's computer. After this process a program link should have been added in the user's start menu.

In most cases no configuration is necessary to start working with the program. For safety the application will ask the user to check the settings upon starting the program for the first time, but it is generally safe to leave most settings as they are. The only setting that might be of interest is the 'remote server'. Atomsnet isn't preconfigured to connect to a network of active servers. If you do want to become part of an active p2p network you will have to add the IP address and portnumber of a running server here. A good source for server addresses is the discussion board at the atomsnet website.

Uninstalling the program can be accomplished by clicking on the start menu link below the application link. The automated uninstaller will then completely remove all references to atomsnet on the user's computer. Only when a user has added new files such atomsnet databases or webpages in the atomsnet directory will this location exist after uninstallation. Removing these files and deleting the directory has to be done by hand in that case. It is important, however, to run the uninstaller prior to deleting the directory by hand. Otherwise some references to the applications will remain on the local computer.

B.1.3 The Menus

Contrary to many Windows applications, the atomsnet user interface is split into separate menus. Upon starting the application the main menu is shown. The menu is shown in the next image.

Apart from the exit button, all of these buttons open a separate window.

The top botton, 'organize my data' allows a user to create an new database or open an existing one. When a database is opened files can simply be dragged from the windows explorer to the menu after which they will be added to the database. Be sure to save the database before you leave this menu. More detailed information on indexing data can be found in the 'indexing data' section.

The next button opens the network window. Here you can start and stop the network server. The menu has the same options for opening and closing database files as the indexing menu described above. This allows you to keep multiple databases for different uses. When closing this menu, the network is automatically shut down. Due to some network management done in the background it may take a while before the network is shut down, and therefore before the menu closes.

For debugging purposes all modules send human readable messages to the main applications. These can be viewed in a separate menu by selecting the 'show messages' button. In many cases this information will not be of interest

to the user and therefore the button can be left unset. If you are experiencing problems or want to follow what's going on in the background this is an essential tool.

Finally, the settings dialog allows you to alter atomsnet's configuration. Be careful, however, in most cases these settings are correct and don't need to be altered.

B.1.4 Configuration Options

The configuration options can be set from the settings dialog. I will explain them in the order in which they are displayed.

First of all the location of the atomsnet homedirectory has to be given. This is initialized to the directory atomsnet was installed in. Changing this value can make atomsnet unusable, so please make sure that it is always set to the location of the application. Again, under normal circumstances this option can be left alone.

Second, the local IP address and portnumber can be chosen. Under normal circumstances the IP address is guessed by the application. It should only be altered if a user finds himself shielded from the rest of the network by a so called firewall. Configuring atomsnet for use behind a firewall is discussed in the 'sharing data' section below. The local portnumber is set to 80 by default. This is the normal webserver port and it is advised to use this port. In case another application is already running on port 80 this setting can be changed to the user's liking. Please note that atomsnet will show a message in the message window if it is unable to claim port 80.

The next option regards connecting to an active network of atomsnet servers. If you want to become part of a network you should add the address of a known server, together with its portnumber (usually 80) here.

Finally, you can set the maximum number of simultaneous connections. By default this is set to 5, but it can be any number from 0 to 25. The number of simultaneous connections has nothing to do with the users that access the computer from their webbrowser. It deals with the number of other atomsnet servers your server can contact directly. Since increasing this value means more background communication is allowed it can seriously degrade both your computer's responsiveness and your available network bandwidth. Setting this value to anything higher than 5 can be considered unwise in a highly active network. It is however useful in private networks that generate little activity, such as company intranets or in-home networks. Please note that setting this value to 0 disallows any connections to other servers, thus making your atomsnet application a stand-alone webserver.

B.2 Finding an item

To clients the network is displayed as a website. This means that although the atomsnet application can only run on Windows computers anyone can access

the shared information. To search across a network you have to enter the address of a running server that is active within this network. This computer can have a hostname (e.g.: http://www.myserver.net) or an IP address (e.g. http://10.0.0.1). Webbrowsers guess that they should connect to port 80. This is why atomsnet servers are configured by default to listen on this port. In case the server has been reconfigured to listen at another port this has to be told to the webbrowser by appending the following:
:X
, where X is the portnumber. For instance, a server running on a computer with IP 192.167.12.14 and port 2000 can be accessed by typing http://192.167.12.14:2000 in the webbrowser's address bar.

When connected, you will see a webpage that can be either the standard atomsnet entrypage or a page created by the server's administrator. In the general case you will find both a search and multiple browse options. Searching the network works similar to other searchengines. Browsing is modeled to so called directories, such as Yahoo! (http://www.yahoo.com/) and the Open Directory Project (http://www.dmoz.org/ and http://directory.google.com/). Anyone accustomed to searching the web will be able to use the atomsnet. Because many computers have to be accessed for a request the waiting time can be longer than with regular searchengines, however. How long you have to wait before results are displayed is linked directly to the speed of the server, the size of the network and the bandwidth between the participating computers.

Although the program's functionality is extendible the basic browsing options are the following. Firstly, you can find files according to their type, this means all files in the network are grouped according to their nature. Searching by type (or more precisely MIME-type) uses a two level structure. You start by selecting the broad type, for instance video. After that a selection has to be made according to the precise filetype, for instance MPEG. It is, however, also possible to simply select video files and from there start a search, so that all video files regardless of their subtype that are of interest will be presented.

Another browse option is 'by category'. Categories are directly copied from the Open Directory Project format and display files not according to their type, but according to their content. This way it is for instance possible to find all files that concern football. Thousands of subcategories exits, making this a very finegrained browsetree. It is important to note, however, that this feature is experimental and not all files are categorized correctly. Hopefully the successrate will go up when new and improved plug-ins become available, but currently it is mainly useful for files where the filename contains the subject, or for mp3 audio (browse by artist).

The last browse option is 'by directory'. This option mimics the physical directory structure of the server you are accessing. It is added with the idea that people normally create their directory structure in a logical way. Using this option means relying solely on the archiving techniques of another person. It can be useful as an alternative to the browsing options discussed earlier, since many people index their files either by filetype (for instance a 'song' directory containing all mp3 files) or by category (for instance a directory containing all

files belonging to a certain project).

When a file is selected, all characteristics can be shown by clicking on the filename. This way the mimetype location, the category and the directory location of the item are exposed, allowing you to browse similar files from this starting point. Furthermore, file size, date and additional information is made visible.

Since the network user interface is modeled closely to existing web resources I believe using the system is straightforward for most internet users. It is however possible for anyone that administers a server to alter the look and the working of his server to his personal liking. This means that certain features described above can be turned off, others can be added and the overall navigation can be rewritten completely. The usability of the network is completely in the hands of the server administrator. If you do not like a particular layout you can choose to 'hop' to another server until you find one that suits you. As a last resort, it is possible to create a personal entrypoint by installing the application on your local computer and connecting to the existing network.

B.3 Indexing data

Data can be indexed by accessing the 'organize my data' menu from the main menu. The first thing you need to do is open a file. This can be done in several ways. You can either open a previously created file or start with a blank file. When an older database is opened the system starts by checking the consistency of the database. This might take a while if you are loading a database containing many files.

When the database is loaded files can be added by simply selecting them in windows explorer or a similar program and dragging them to the menu. As soon as a file or selection of files (and/or directories) are dropped on the menu the applications starts to find information about the files through several options. The application relies on plug-ins, small separate programs, to find this information. The amount of information found and the time it takes to process depends largely on the plug-ins used. If you want to know what plug-ins the system uses open up the messages window before loading the database. Especially plug-ins that connect to other computers, for instance the google indexing plug-in that is part of the base package, will be active up to a few seconds per file. This makes the indexing process quite slow. It is therefore best to add up to a maximum of tens of files at once. Another reason to not overload the system is that the addition of data entries will take up quite a lot of memory space. As a reference, adding a thousand files can take up approximately 100 mega bytes of main memory. When all files have been added to the database make sure you save the file before exiting the menu, otherwise all new information will be lost.

B.4 Sharing data

Sharing the data can be accomplished by opening the 'share my data' menu. From that menu you can select to open a database. As soon as the file is successfully loaded the webserver is started. Only if the 'open file' buttons become grayed is the network active. Please open the message window and try again if the network is unable to load. This usually means that the local port is already occupied or the database cannot be loaded correctly.

As soon as the webserver is active a background process starts as well. This process tries to find other servers on the network and remains active as long as the webserver is active. Only when no remote server is given in the settings or when the maximum number of simultaneous connections is set to zero will this process stop immediately. How many connections are active can be seen at the bottom of the menu. Here a message can be found that usually reads 'no connections' upon network activation. During the active period other servers can be found, increasing the number of connections. This number will never exceed the maximum number of simultaneous connections as given in the settings dialog.

Leaving the network is accomplished by clicking the the 'close and return' button. It might take a while before the menu is actually closed. This is not a bug. During the waiting time the server is gracefully closing all active connections and closing the webserver. Since the system might be busy we have to wait a little while. Please be patient and wait for the system to return. Shutting down the application (for instance with the Windows Alt-F4 combination) will shutdown the network much faster, but can result in data inconsistencies.

B.4.1 Firewalls

If you do not know what a firewall is it is probably safe to skip this section. If you are aware that your computer is shielded from the rest of the active network, usually the internet, by a firewall some extra steps have to be taken to be able to run the atomsnet server.

There are two possible problem when a server is started from behind a firewall. Both problems can be dealt with and more often than not they exist simultaneously.

Firstly, it is possible that your computer does not have an internet wide IP address. This is made up for by having a firewall that acts as a masquerading server. Explaining how masquerading works is outside the scope of this document, but many good information sources can be found on the web. In short, masquerading makes your computer look like the masquerading server to everyone at the other side of the firewall. Since your IP address is unknown to people on the wide network they cannot access your computer. To circumvent this your server should make himself known by supplying the closest point on the wide network: the masquerading server. This is precisely where the 'local IP' option is for in the settings dialog.

Let me give a small example. Say your computer is behind a masquerading

server. Your IP address is 10.0.0.2 and the server has address 100.0.0.1 on the wide network and 10.0.0.1 on the local network. While the addresses seem alike, 100.0.0.1 is a valid internet address, while everything in the range 10.X.X.X is reserved for local use. Anyone try to access 10.0.0.2 will therefore fail to reach you. If, however, you supply the internet address 100.0.0.1 others can find your masquerading server and all that needs to be done by this computer is to resend the data to your computer. This leads to the second problem.

Firewalls allow users behind the firewall to access computers outside their local network, but restrict strangers to access local computers. To allow users to access your atomsnet server you need to alter the firewall so that it allows others to access part of your computer, namely the atomsnet server. This can result in a serious security breach and has to be done with the utmost care. Ask permission from the firewall administrator before changing anything in the firewall! Since atomsnet listens on one port, namely the one configured in the settings window, you will only have to allow the firewall to forward data destined to this port to go through. Everything else should stay blocked.

In case of a masquerading firewall, remember that data is actually sent to the firewall, not to your computer. In this case the data should not only be allowed to travel through the server. You have to explicitly tell it that data destined for the firewall (in our example 100.0.0.1) on the same port as the one your computer is expecting it (for instance port 80) should be redirected to your computer. This means redirecting all requests going to 100.0.0.1:80 to 10.0.0.2:80 .

The portnumbers on the firewall and the atomsnet server have to be the same. This means that you should set your local port to a value acceptable for the firewall. Again, ask your firewall administrator for a suitable port, usually this will be a number greater than 2000.

Allowing external accesses to your computer can cause security breaches. Especially the configuration of the firewall is very important. Since atomsnet can only deal with security on its designated port all other open connections are the user's own responsibility. How atomsnet deals with network security is dealt with in the next section.

B.4.2 Security

For every program that has access to network resources security is a key concern. Atomsnet has been created as an easy to use program, meaning that configuration of the system is largely unnecessary. Therefore no human errors can make the system less secure (aside from the possibly needed firewall configuration discussed in the previous section).

The simplicity of the application is the strongest advantage of the atomsnet application. Since external users are only allowed basically four actions and by default denied others I believe the system to be very secure. All actions allowed use the HTTP GET protocol. Any other network requests are automatically declined. All requests result in read-only operations. I will now deal with safety regarding each allowed action.

Accessing the database Most user actions will concern retrieving information from the database. This information resides in main memory, so no disc access has to be given to the user. Only requests for data based on database item identifiers is accepted, therefore governing that no access is given to any information outside of the database. If you want to know what information is accessible please inspect a database by using an XML editor or text editor. In any case, no personal information is given.

Accessing the database should therefore not cause a direct security breach. Users should take notice that the following data is exposed: the directory layout as far as the files added to the database is concerned and the IP address(es) of the user's computer. This information can be considered of limited value to anyone trying to access your computer.

Accessing webpages Apart from querying the database, users can also access all files in the directory specified as the homedirectory in the settings window as well as all of its subdirectories. This is commonly the place where the atomsnet application and webfiles reside. The reason for this is that the atomsnet application has to be able to respond to requests for webpages, in the very least index.html . Since no access is given to locations using the 'parent directory' (..) locator, other parts of the computer are inaccessible to clients.

Retrieving files Shared files are made accessible as well, but these can reside anywhere on the user's harddrive. To make sure no other files can be viewed these shared files are only made available through their database identifiers. Since no direct filename based requests are allowed access is restricted to the files explicitly added to the database by the user.

Finding network resources The last action the server can be requested to undertake is to send a list of known servers. Replies to this request are simply lists of IP addresses. This should not lead to any security issues.

Stability The only possible security breach to my knowledge that can come from using the atomsnet application lies in the flooding approach. Flooding is generally used to get the computer in an unstable state. This can sometimes allow a user to get access to low level networking features *outside atomsnet*. Having an atomsnet server running does incur processor overhead. Misusing the system by sending a massive amount of simultaneous requests can degrade system performance to the point that the system becomes unresponsive and even unstable. In how far this can lead to security risks is dependent on the operating system. When viewed as such, any network server running on your computer can pose a threat, but a minor one at most.

B.4.3 Personalization

The atomsnet package comes with a standard built in webserver and some accompanying files. The webserver cannot be altered (in the binary distribution) for security reasons. You can change and replace the webfiles given with the main package to create your personal look and behaviour. Understanding of webtechnologies is necessary to change this data.

The standard package consists of four important files. These files can all be found in the installation directory of the application. Index.html is the file shown to a client when he accesses your server directly at http://www.myserver.net/ (exchange this hostname for your own hostname or address). This file basically explains how to use the system and provides links to the search and browse facilities. You can completely restyle your entrypoint by altering or exchanging this page with your own webpage. To alter the look of the main website you need to have a basic understanding of the Hypertext Markup Language (HTML). Information about HTML is publicly available on the web.

Apart from the index page, you can also alter the look and feel of the entire website, including the results the database send back by changing the file atomsnet.css . This file is written in the Cascading Style Sheets language. Please visit a website to learn more about CSS before altering this file. Basically, CSS tells the users webbrowser how a data item should be shown. For instance, the color, font and size of a text element can be chosen. By altering the data in atomsnet.css these and more layout features can be altered. For instance, changing
background : *white*
into
background : *black*
will make all pages have a back background. Of course, more interesting aspects can be changed.

If you want to change not how, but what information is displayed another file has to be changed. The file ane_html.xsl is is used to translate the raw database entry into a human readable format. This file is written in yet another language, XSLt. Again, please visit a website to learn more about this technology. In short, XSLt translates data from the raw XML form in which it is sent to the client into HTML document code. In our standard distribution, for instance, each browse result is shown to the user in a format resembling the Open Directory Project's style. You can display the data in a completely other format by simply changing this file.

The fourth file added to the distribution, corner.gif, is simply a picture containing the atomsnet logo. It is referenced from atomsnet.css . So, if you remove the reference to this image in atomsnet.css it is safe to remove the image.

Summarizing, the look and behaviour of the webserver is governed by standardized style documents. You are free to alter or exchange these files. Doing so does rely on understanding of the technologies used and can be a bit hard at first. Especially XSLt is a new technology and might pose a few problems.

B.5 Further information

I hope this manual has helped to understand the features of the atomsnet application. If you want to know more there are other resources publicly available. Apart from this document there exist a developer manual and a research document based on the application. Furthermore, the application is also being made available as sourcecode, meaning that you can view precisely how it works. For this an understanding of C++, Windows API's and webstandards is very useful.

If you have specific questions you can also ask them directly by placing a message in the discussion board at the atomsnet website. The address is http://atoms.sourceforge.net/ . Remarks and bug-reports can also be placed there.

C Developer Manual

The atomsnet application has been developed as an open source project. All code is made freely available under the GNU General Public License. As such, I want to encourage anyone to learn from, alter and reuse any code. To allow program alterations the application has been designed as modularized as possible. How these modules interact is discussed in detail in later sections.

I've tried to write useful comments in between the sourcecode wherever applicable. This should allow anyone with a basic knowledge of programming to start coding right away. Therefore this document will only describe the architecture from a bird's eye perspective. Detailed information about the application's functionality can be found in the User's Manual. This information will not be repeated here, so please read that document before continuing.

Environment Atomsnet has been written as a Windows application using standard C++. I've used the Microsoft Visual C++ 6 integrated development environment for creating the system. Apart from the standard Windows hooks the system also depends on the Microsoft XML parser, version 3 or higher. Information about all things microsoft can be found at their developer's website, http://msdn.microsoft.com/ .

Porting Reusing code in another environment has been made relatively easy by shielding most operating system specific calls from the general code. All library loading, networking and window manager specific calls have been placed in specialized functions that can be overridden with limited effort. The XML parser calls exist solely in the datalibrary module. Unfortunately they can be found throughout this module, so it will be harder to replace them.

C.1 Plug-in Architecture

Metadata indexing is based largely on file-type specific heuristics. Therefore I've chosen to remove all indexing code from the application codebase. The actual

process of finding metadata has been placed in separate modules that follow a very basic design. These modules shall from now on be called plug-ins.

The application's plug-in architecture consists of two parts: the module loader framework and a selection of representative example modules.

C.1.1 Framework

The framework is responsible for:

finding modules
verifying module integrity
loading modules
running modules
integrating module results in the database
and
safely unloading modules

The functionality is encapsulated in two functions. Upon opening a database, the framework is loaded for the first time and the discovery loop is entered. This loop reads all files in the plug-in subdirectory of the homedirectory. Then it tries to load each file and verifies its integrity by calling a standard function. This somewhat rudimentary testing should suffice to accept or deny any file encountered. More importantly, crashing the system by loading incorrect files should not be possible. Each file that has been identified as atomsnet plug-in is added to the opened database under a special key.

When indexed, files are normally added to the document as xml elements called resources. These elements are placed in the tree-like structure according to their metadata. Plug-ins are added in largely the same way, but are distinguished by their element's name. This mixing of data and code will be discussed in the next paragraph. After initialization all plug-ins are released and unloaded from the execution environment.

When a user drops a file the actual indexing takes place. This is where the plug-in/resource mixing of the datatree becomes useful. Upon starting the indexing process no previous data exists so a general plug-in is called that resides at the root of the datatree. This plug-in is guaranteed to return a result, since it relies solely on filesystem information. After this bootstrap process a recursive process starts where the framework searches through the tree for an applicable plug-in. The search is started from the location where a resource has last been added or altered and continues up the tree. This way all ancestor nodes are travelled. When a node is encountered containing a plug-in this plug-in is called for the current file.

A problem that may occur with this recursive process is that a loop in the dataflow can be created. This problem has not been dealt with explicitly in the testprogram, since too few plug-ins currently exist to create such a loop. Please note that it should ofcourse be dealt with in reallife products where more plug-ins are used.

The rationale behind the recursive indexing process is that recently discovered information can help us select new appropriate plug-ins. This is obviously at least as efficient as calling all plug-ins, while it gives at least as good results as calling a preselected subset.

Let me give a small example. A user wants to index an mp3 file. In the standard package the following process would take place. The bootstrap plug-in writes filesystem data and then calls the MIME-type plug-in, because it is entered at the root of the filesystem subtree. The filesystem plug-in is not called again since recursive plug-in searching stops one level above this node (a somewhat dirty hack). The mime-type plug-in enters information under the audio/mpeg entry of the file-type subtree from where the mp3 specific plug-in is called. Finally, the google plug-in is called because it is entered at the root of the MIME-type subtree. It is obvious that new subtrees can thus be added to the system by creating an appropriate plug-in. Similarly, subtrees can be removed by removing specific plug-ins. This flexibility is the main strong-point of the plug-in based system. The next section deals with coding a plug-in.

C.1.2 module layout

The framework uses a very simple scheme in communicating with the plug-ins. The rules have to be followed precisely, otherwise the system might either not accept a plug-in or show unexpected behaviour. The precise naming of the functions can be seen in the example plug-ins source. I will now only describe the general functionality required. It is probably easier to alter the example plug-ins then to create a new plug-in from scratch. There exist two different versions, however. These two versions have identical hooks when it comes to displaying their name, version and preferred locations. The difference lies in the actual processing hook. The more advanced plug-ins adds XML data using the MS XML API. Since plug-in developers might not want to use the MS XML parser a plug-in is also accepted if it listens to a second function, but not to the first. This second functions is expected to return a simple text string containing the xml data.

The framework starts by calling the simple text function. If this function returns an empty string or the NULL pointer, the more advanced XML enabled function is called.

It is completely up to the developer whether he wants to create XML enabled plug-ins. The main advantage for the developer is that he can view and alter the open database from within the plug-in and thus has more options than with the the text based plug-in. If this extra functionality is not needed I would suggest creating pure C++ text based plug-ins, since these are quicker to write, smaller in code and easier to port.

C.1.3 example plug-ins

Since this is a testprogram only a few plug-ins have been created. The most simple one reads data from the filesystem and creates an entry in a subtree

that models the filesystem. The second plug-in uses file content to identify the filetype and creates an entry in a MIME based hierarchy. This module is currently a simple wrapper around a function exported by the *Microsoft Internet Explorer*^{tm}. It identifies some 30 different types. A third plug-in is created specifically for reading the proprietary metadata format often used in .mp3 audio files called ID3. This plug-in does not add items to the hierarchy, but simply alters and extends existing entries.

The last plug-in best demonstrates the strength of the system. This plug-in creates keywords from the already added information and with these keywords queries an internet directory. The directory queried is currently the google directory (directory.google.com) that is equivalent to the open directory project. From the answer a local version of the directory is created containing the processed files.

C.2 Reusing Modules

Besides the fine-grained plug-in architecture, the application itself has also been subdivised into larger modules. All user interface code can be found in the executable, while the database and networking code has been placed in separate libraries. These are called datalibrary.dll and netlibrary.dll .

Modularizing the entire application allows for relatively easy replacement of the interface, the networking approach or the database back-end. It also helps in keeping the interdependency of code relatively small. The data- and netlibrary are called by the user-interface through special hooks to start and stop their specific routines, such as the webserver and the indexing engine. Furthermore, the database library contains a class that is used by the webserver to query the database.

Which functions have to be exported can be seen in the .def files accompanying the module sourcecode. All interfacing is done through standard C++ function calls, so portability should not be to great an issue.

C.3 Database layout

The databases used by the atomsnet applications are standard XML documents. This reduces system complexity, since data can be sent to a client without conversion. Formatting this data into webpages is done on the client's computer through the use of XSLt transformations. More about this client side processing can be found in the User's Manual.

The database layout follows a strict grammar, which has been written down as an XML schema document. Due to the poor validation engine currently part of the XML parser this document is not yet used by the application. Developers are, however, expected to comply to this guideline, so please read it before changing the database layout. The schema document can be found at http://atoms.sourceforge.net/atomsnet.xsd .

To summarize the grammar, the document consists of a standard namespace header and an ¡ rdf ¿ .. ¡ /rdf ¿ pair encapsulating the actual data. No actual

rdf validation is used, but adherence to the Resource Description Format is encouraged. Within these tags the ¡ atomsnet ¿ .. ¡ /atomsnet ¿ tags set apart the atomsnet data from possibly other rdf data. This is the root of our actual database. It consist of an application- and a datatree. The application part contains accounting information, such as the number of nodes in the database and the server's ip address(es). The database node can have a possibly unlimited number of subtrees, but contains only 3 in the base distribution, namely the mime, cat and sys trees. These contain MIME-type, Open Directory Project categorization and filesystem dependent information, respectedly.

Each subtree of the database can contain a possibly unlimited number of ¡ resource/ ¿ nodes, describing files. They can also contain a possibly unlimited number of plug-ins as described above.

The resource nodes can contain both subnodes and attributes describing metadata. In the reference version of the application all data is written in attributes. One of these attributes, the ID, works as a key for quick lookups. It is especially useful when finding different resource tags belonging to the same file. As an example, the ID s1 responds to the file with ID 1 in the sys subtree. A resource with ID m1 will have to refer to the same file as the s1 resource, but will reside in the MIME-type subtree.

Apart from the features described in the schema the database can be extended and altered to your personal need. The restrictions laid upon the grammar are needed to allow interoperability between atomsnet applications network wide. An example database containing one file is shown below.

```
<?xml version="1.0"?>
<?xml-stylesheet type='text/xsl' href='ane_html.xsl'?>
<rdf>
<atomsnet ip="http://0.0.0.0:80">
<programdata version="1.1" idcount="1"/>
<index>
<sub name="mime">
<plugin name="anedll_cat_dir_google.dll"/>
<sub name="audio">
<sub name="mpeg">
<plugin name="anedll_mimempegaudio.dll"/>
</sub>
</sub>
<sub name="application">
<sub name="octet-stream">
<resource id="m1" sysref="s1" catref="c1" name="atomsnet v1.lnk" size="378" date="2002-7-7"
</sub>
</sub>
</sub>
<sub name="sys">
<plugin name="anedll_mime.dll"/>
<sub name="c:">
```

```
<sub name="windows">
<sub name="start menu">
<sub name="programs">
<sub name="atomsnet">
<resource id="s1" mimeref="m1" name="atomsnet v1.lnk"/>
</sub>
</sub>
</sub>
</sub>
</sub>
</sub>
<sub name="cat">
<sub name="computers">
<sub name="software">
<sub name="operating_systems">
<sub name="network">
<sub name="distributed">
<resource id="c1" mimeref="m1" sysref="s1" artist="atomsnet v1"/>
</sub>
</sub>
</sub>
</sub>
</sub>
</sub>
<plugin name="anedll_sys.dll"/>
</index>
</atomsnet>
</rdf>
```

C.4 Further information

Apart from this document there exist a user manual and a research document based on the application. Furthermore, the application is also being made available as sourcecode, meaning that you can view precisely how it works. For this an understanding of C++, Windows API's and webstandards is very useful.

If you have specific questions you can also ask them directly by placing a message in the discussion board at the atomsnet website. The address is http://atoms.sourceforge.net/ . Remarks and bugreports can also be placed there.

D Database Schema

Below the database grammar is displayed. The grammar has been written down in a special XML format called XML Schema. Please visit the World Wide Web

Consortium's homepage (23) for more information about XML Schema.

```
<!-- http://atoms.sourceforge.net/atomsnet.xsd redirection page -->

<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:include href="http://atoms.sourceforge.net/atomsnet-1.0.xsd"/>
</xsd:schema>

<!-- http://atoms.sourceforge.net/atomsnet-1.0.xsd latest version -->
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- element definitions -->
  <xsd:element name="plugin" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
   <xsd:attribute name="name" type="xsd:string" use="required"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="resource" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
  <xsd:attribute name="id" type="xsd:ID" use="required"/>
  <xsd:attribute name="sysref" type="xsd:IDREF" use="required"/>
  <xsd:attribute name="catref" type="xsd:IDREF" use="required"/>
          <xsd:attribute name="name" type="xsd:string" use="required"/>
          <xsd:attribute name="title" type="xsd:string"/>
          <xsd:attribute name="description" type="xsd:string"/>
          <xsd:attribute name="publisher" type="xsd:string"/>
          <xsd:attribute name="subject" type="xsd:string"/>
          <xsd:attribute name="creator" type="xsd:string"/>
          <xsd:attribute ref="xml:lang" use="required"/>
          <xsd:attribute name="size" type="xsd:long"/>
          <xsd:attribute name="date" type="xsd:date"/>
  <xsd:attribute name="mimeref" type="xsd:IDREF"/>
         <xsd:attribute name="path" type="xsd:string"/>
  <xsd:attribute name="id" type="xsd:ID"/>
  <xsd:attribute name="mimeref" type="xsd:IDREF"/>
</xsd:complexType>
  </xsd:element>

  <xsd:element name="sub">
    <xsd:complexType>
  <xsd:attribute name="id" type="xsd:ID"/>
      <xsd:attribute name="name" type="xsd:string"/>
```

```
<xsd:sequence>
      <xsd:element ref="sub" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="resource" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="plugin" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
  </xsd:element>

  <xsd:element name="index">
    <xsd:complexType>
  <xsd:element ref="dir" minOccurs="0" maxOccurs="unbounded"/>
</xsd:complexType>
  </xsd:element>

  <xsd:element name="programdata">
    <xsd:complexType>
    <xsd:attribute name="version" type="integer" use="required"/>
<xsd:attribute name="idcount" type="integer" use="required"/>
</xsd:complexType>
  </xsd:element>

  <xsd:element name="atomsnet">
    <xsd:complexType>
       <xsd:attribute name="ip" type="xsd:string"/>
  <xsd:sequence>
    <xsd:element ref="programdata"/>
<xsd:element ref="index"/>
  </xsd:sequence>
  <xsd:key name="resourceid">
        <xsd:selector xpath="resource"/>
        <xsd:field xpath="id"/>
  </xsd:key>
  <xsd:keyref name="resourceidref" refer="resourceid">
    <xsd:selector xpath="resource"/>
<xsd:field xpath="catref"/>
  </xsd:keyref>
  <xsd:keyref name="resourceidref" refer="resourceid">
    <xsd:selector xpath="resource"/>
<xsd:field xpath="mimeref"/>
  </xsd:keyref>
  <xsd:keyref name="resourceidref" refer="resourceid">
    <xsd:selector xpath="resource"/>
<xsd:field xpath="sysref"/>
  </xsd:keyref>
</xsd:complexType>
  </xsd:element>
```

```
  <!-- attribute definitions -->

</xsd:schema>
```