Reference Architectures

for Peer-to-Peer Applications

James Walkerdine, Lee Melville, Ian Sommerville

{walkerdi, l.melville, is@comp.lancs.ac.uk}

Computing Department

Lancaster University

Lancaster

LA1 1BW

UK

Point of contact:
James Walkerdine
Computing Department
Lancaster University
Lancaster
LA1 1BW
UK
Email: walkerdi@comp.lancs.ac.uk
Telephone: +44 (0)1524 594117
Fax: +44 (0) 1524 593608

Summary

This paper presents a set of Peer-to-Peer application reference architectures and their associated capabilities that have been developed as part of the P2P ARCHITECT project. These reference architectures capture fundamental architecture features and functionality that designers can use to assist in the understanding of existing P2P applications or to assist in the design of new ones. The reference architectures presented here have been developed by studying a number of widely used existing P2P applications and abstracting their common structure and functionality. Reference architectures for a number of the more common P2P application domains have been produced including, Instant Messenger, shared workspace and computational. The paper also provides a number of comparisons that have been performed between the reference architectures and existing P2P applications.

Keywords: Peer-to-Peer, Reference architectures, application design

1. Introduction

This paper presents a set of Peer-to-Peer (P2P) application reference architectures that have been developed as part of the P2P ARCHITECT [1] project. Reference architectures capture the fundamental architectural features and functionality of domain specific software applications by providing an abstract and technology independent view of the domain [2]. The reference architectures described here can be used to assist in the understanding of a P2P applications structure, be used as a basis for architectural comparison, or simply to provide designers with an abstract starting point from which to build their own P2P applications of that domain type. To demonstrate their use a number of architectural comparisons are provided later in this paper.

In simple terms, a P2P application can be regarded as a class of application that takes advantage of the resources and services that are available at the edge of the Internet [3]. Over recent years there has been an increased interest in P2P applications, supported by the development of applications such as Napster[4], ICQ[5], SETI@home[6], and applications based on the Gnutella protocol[7]. The research within the P2P domain, however, has predominantly focused on low-level aspects, such as protocols, routing algorithms and discovery mechanisms. Less attention has been given to the designing of the higher-level P2P applications.

The EC funded P2P ARCHITECT project seeks to develop methods and tools to support software-developing organisations in building dependable P2P software applications. Such developments have focused on aspects of higher-level P2P application design, with one output of the project being the development of a set of application reference architectures. Designers can use these reference architectures to help structure and identify required functionality for their designs. As part of the project the end user partners made use of the reference architectures to help develop P2P applications for use within their organisations.

This paper presents six different types of P2P application reference architecture that were developed within the project. Due to the requirements of the end user project partners (who wished to develop collaborative styled P2P applications - a shared document editor and theatre booking system), a focus has been placed on co-operative application domains. The architectures include:

- General architectures for co-operative P2P applications
- Instant Messenger application architectures
- Shared Workspace application architectures
- Distributed search (and file sharing) application architectures
- Document Management application architectures
- Computational application architectures

The application reference architectures were developed by studying existing widely used P2P applications (including Napster, ICQ, MSN Messenger [8], SETI@home, Morpheus[9], and FreeNet [10], Gnutella and Jabber [11] based applications including Frost[12], Gnucleus [13], Limewire[14] and MirandaIM[15]) and abstracting the structure and general functionality that was common to them. These abstractions were then analysed and suitable application reference architectures produced. Widely used commercial and open source P2P applications were drawn upon as the basis for the reference architecture development as they represent applications that have been successfully deployed, tested and used on a large scale. Although small-scale P2P applications such as those developed as part of research projects are equally as valid, their limited usage makes it difficult to assess their success as an application design. Consequently they have not been considered here.

The presented architectures cater for the two key types of P2P structure: semi-centralised applications, those that possess one or more server nodes, and decentralised applications, those that possess nodes of 'equal' standing. The underlying logical network architectures that provide the foundation for such applications have been discussed in detail in our previous work[16][17], although to clarify the differences a summary is provided in this paper.

The paper begins by discussing the uses of reference architectures and in particular what benefits they can provide to application designers. An overview of how the developed architectures are represented within the paper is provided, with the layered based notation and the notion of capabilities described.

A brief summary of common P2P logical network architectures and substrates is then provided which highlights the significant relationship between the underlying P2P topology/technology, and the applications that are built on top. Decisions made with regards to these will have some affect on the suitability of the different application reference architectures.

The paper then moves on to present the developed application reference architectures that form the bulk of the paper. Generic reference architectures for co-operative P2P applications are first presented, before extending these into specific application domains. Descriptions of the layers and capabilities that are possessed by the architectures are provided.

To illustrate one way in which the application reference architectures can be used, towards the end of the paper a number of comparisons between the reference architectures and existing P2P applications are presented. These comparisons seek to identify similarities between the reference architectures and the individual application architectures.

Finally, the paper concludes with a brief discussion of related work, our experiences in using the developed reference architectures and our intentions for future work.

2. Reference Architectures

Reference architectures themselves can be interpreted in many ways. Within this paper, we do not regard them as merely being templates for the creation of P2P applications. As their name suggests, they should be considered as *points of reference*. Reference architectures can be a valuable resource to designers, with their benefits including where:

- They can be used to provide designers with a high level understanding of how such applications can be structured (in a functional sense)
- They can be used to help designers understand types of application architectures
- They can be used as a basis for performing architectural comparisons with existing applications
- They can be used as possible starting points (from an abstract perspective) for the development of the respective types of application.

The reference architectures presented here should not be regarded as *application designs*. Their main objectives are to represent functionality at abstract level and to highlight possible structure for this functionality. The reference architectures do not provide an illustration of how an application can be designed, as their very nature makes them far too abstract to achieve this. Such abstraction is required so that a wide range of different possible application designs can be encapsulated. As a result they cannot contain details of specific functionality, as these have to be elaborated on during the individual application designs.

For example, the Instant Messaging applications MirandaIM (based on Jabber) and ICQ can both be abstractly represented by the semi-centralised Instant Messenger reference architecture that will be presented later. However, their underlying functionality is significantly different. This functionality is captured by the individual application designs, rather than the reference architectures.

2.1. Architectural representation

Within the paper a layered based notation has been used to represent the different application reference architectures (as has been commonly used for reference architectures such as the OSI network model[18]). This type of representation aims to provide a simple overview of the architectures and identifies, at an abstract level, the main functionality and structure that would be commonly needed for the specific applications.

Layered based notations are commonly used for depicting reference architectures as they allow for the representation of functionality and structure without having to describe the specifics. However, the overview nature of the layered notation can make it difficult to clearly identify the differences between the various application types. Consequently the application reference architectures that are presented here possess a degree of similarity, as layers that capture common functionality are re-used.

Within the architectures, the borders of each layer should be regarded as an interface to the adjoining one. In practical reality, however, this may not necessarily be the case, as whether or not certain layers are actually relevant or how they interconnect will be dependent on the specific application that is being designed. Consequently, although a layer is included within an architecture, it does not necessarily mean that it will always be used. For instance the application layer may interface with the P2P Network Layer directly, under certain circumstances.

The representations of the reference architectures are also supported with descriptions of the common capabilities that they can possess. In this paper we refer to a *capability* as being an abstract system functionality. Example capabilities could be encryption, Quality of Service (QoS) monitoring, peer advert caching, etc. Capabilities have been identified for the complete application reference architectures, and for the individual layers that comprise them.

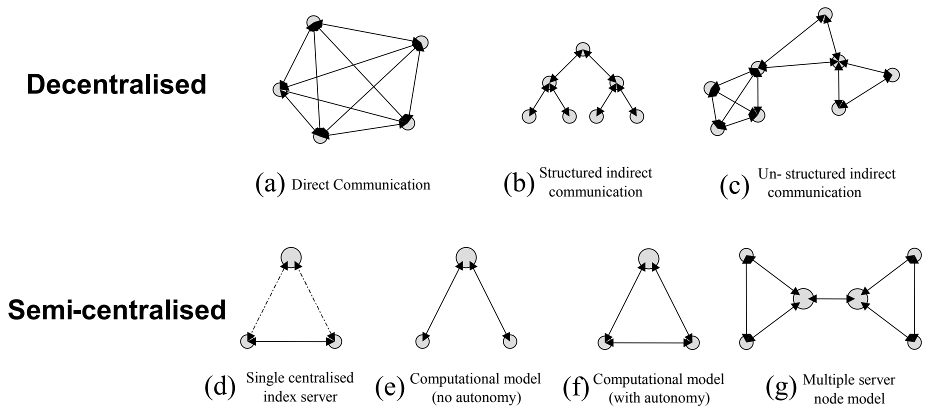
A degree of overlap can exist between these capabilities, in that a specific capability can exist at a number of layers or within a number of application domains. For example, encryption could take place at both low and high levels within a P2P application (for example, at the network level or at the higher application level). When a P2P application is being developed, it will be up to the designers to decide what individual capabilities are required and where they will be provided within the system.

By considering what capabilities are desired, the designers are forced into thinking about application functionality from an early stage. The identification of capabilities can also help in determining what application reference architectures (if any) may assist the designers in their task, as well assisting the designers in structuring their application design into sub-systems.

3. P2P Logical Network Architectures and Substrates

P2P applications can be built on top of a number of different types of logical network architecture. A logical network architecture can be thought of as an abstraction of the physical network architecture that considers just the peer nodes and the connections between them. For example, although a P2P logical network architecture might only contain five nodes, the actual routing of packets between these nodes could involve computers that do not fall into this architecture.

It is possible to categorise logical network architectures into two main types. Decentralised architectures, in which all peers are regarded as being 'equal' and no controlling nodes exist, and semi-centralised (or hybrid) architectures, in which there exists one or more nodes that have an influence on the rest of the system (i.e. possessing server-styled functionality). Figure 1 provides a graphical overview of a number of commonly used logical network architectures. These have been described in more detail in our previous work[16,17], however we provide a brief summary here.



Decentralised
(a) Direct Communication
(b) Structured indirect communication
(c) Un- structured indirect communication
Semi-centralised
(d) Single centralised index server
(e) Computational model (no autonomy)
(f) Computational model (with autonomy)
(g) Multiple server node model

Figure 1 - Common types of logical network architectures

Common semi-centralised architectures include those that possess a single index server (*d* in figure 1) that plays a supporting role within the system (for example, for peer lookup). Computational based architectures (*e* and *f* in figure 1) that provide a structure for distributed computation co-ordinated by a server peer and where individual peers may or may not possess their own autonomy. Finally, architectures that possess more than one server peer (g in figure 1). Such architectures can provide advantages such as increasing P2P application reliability by removing a potential single point of failure.

Decentralised architectures can be divided into direct communication and indirect communication topologies. Within a direct communication architecture (*a* in figure 1) all nodes within the network can communicate directly with each other, within an indirect communication architecture such direct communication (*b* and *c* in figure 1) is not guaranteed and communication my have to be routed via other nodes (as with applications such as Gnucleus and Frost).

The choice of logical network architecture can have a direct impact on the properties possessed by the application that is built on top. For example, decentralised based P2P applications are likely to be better suited at handling denial of service attacks and aspects of fault tolerance, while the central authority within a semi-centralised based P2P application would be better suited for handling peer certification and safety critical applications. Similarly, the different types of decentralised and semi-centralised architectures can also influence an application's properties. For example, a structured decentralised network may be more efficient for message routing in contrast to an unstructured architecture, but in turn it requires a greater degree of management and potentially can be more prone to faults.

The various effects the choice of logical network architecture can have on a system's properties is a significant topic in its own right, and although it is important for designers to consider these factors during the design process, a detailed discussion is beyond the scope of this paper. Such analysis, however, has been carried out and documented in our previous work [16].

3.1. P2P Substrates

Over recent years a number of protocols and API's have been developed that provide a way for developers to more readily utilise P2P techniques. Notable examples include Gnutella, Pastry[19], Chord [20], Tapestry[21], Jabber, FreeNet and JXTA[22]. In many cases, implementations of these protocols have been produced and have formed the foundations for many P2P applications (for example, Gnucleus, Limewire, Frost). Although these substrates can benefit the development process it should be noted, however, that they are not applications in their own right and consequently the reference architectures presented here have not been derived from them. The P2P application reference architectures should be considered as being independent of such substrates, although it may be the case that layers within an architecture may be satisfied by their functionality (for example, Pastry and JXTA providing some of the functionality that falls into the P2P Network Layer within the reference architectures).

3.2. Decentralised P2P Applications

A significant number of widely deployed and successfully used P2P applications have been based around semi-centralised logical network architectures due to the advantages a central node can provide [16]. Although popular indirect communication based decentralised applications also exist (Gnucleus, Frost, etc) the uptake of such a topology as a basis for application development has been hindered due to a number of issues. In particular their unpredictable network structure (e.g. nodes being removed/connected, network partioning, etc) can make it very difficult for them to be fully managed or to fully support resource discovery, issues that could be particularly important for critical business applications [32].

This lack of control can also influence the types of P2P application that can be run over it. Of the application types presented in this document the lack of central control will make computational, document management and totally decentralised shared workspace applications, difficult to implement (though, in theory, not impossible). Furthermore, large scale decentralised networks can also reduce the usefulness of instant messenger applications (for example, two users may be online but not be able to discover or communicate with each other), as well as limit the effectiveness of search applications [23]. It should be noted, however, that ongoing work is being carried out to improve the reliability of indirect communication decentralised applications, with Chord and Pastry being examples of relevant substrates that have been developed (although resource discovery is still an issue in such systems [32]). As such technology is further refined it is likely that there will be an increasing number of indirect communication decentralised based applications being developed and deployed on a large scale.

To an extent, some of the issues possessed by indirect communication architectures can be overcome by the use of direct communication decentralised architectures. For example, all nodes on the network will be aware of each other; thus removing the problem that applications such as Instant Messengers could otherwise experience. Likewise this knowledge of other peers might make it easier for applications to be managed in a decentralised fashion. However, as indicated in [16], such architectures will not scale well and making them only really suitable for small-scale networks.

The decentralised architectures that are presented here are based on an indirect communication model, and consequently they require and possess mechanisms to publish and discover peers on the network. By removing these mechanisms it should also be possible to adapt the architectures for a direct communication approach, should it be desired.

4. Generic Architecture Descriptions

This section presents a set of layered based reference architectures that represent a generic co-operative P2P application. This generic architecture can act as a reference for the development of more specific co-operative P2P applications that could include, Instant Messenger applications, shared workspace applications, document management applications and distributed search applications. A number of more domain specific application reference architectures, which extend on the generic architectures presented here, are presented in section 6.

The development of these generic architectures has resulted from studying a number of existing P2P applications and abstracting their common structure and functionality. In particular, Napster, ICQ, SETI@home, and other P2P applications (including those based on Gnutella, FreeNet and Jabber) that are in use, have all been examined.

Because these existing applications use both decentralised and semi-centralised structures, reference architecture representations have been provided for both types of logical network architecture class. The semi-centralised reference architectures have been further broken down into client and server architectures, to reflect the different structure and functionality that these typically possess.

The main capabilities for the architecture have also been identified and presented. The individual layers that make up the reference architectures are described in detail, in section 5.

The following diagrams give an example of the layers that are typically required for a co-operative environment based P2P application. The main capabilities of such an application include:

- **Peer Discovery** - Allowing peers to discover and be aware of each other
- **Peer Communication** - Allowing peers to communicate with each other
- **Peer ID** - Allowing peers to be uniquely identified within the network
- **Security** - Providing security within the application
- **Data Management** - Allowing data to be stored and managed on the peers
- **User Interface** - Allowing for the creation of a user interface for the peers
- **Network Structure** - Allowing for the creation of de-centralised and semi-centralised applications

Semi-centralised: Server Node

One of the main roles for the server within a semi-centralised version of this architecture is to support/administer the communications between peers. In addition the server would most likely enforce the security protocols used within the system, and if required, maintain a system data repository. The P2P Network Layer deals with the communication (possibly in a secure fashion) with other peers. Incoming data is passed up to the Message Resolver so it may be decomposed and channelled to the layer it is intended for. In some cases communication may first be authenticated before it is acted upon.

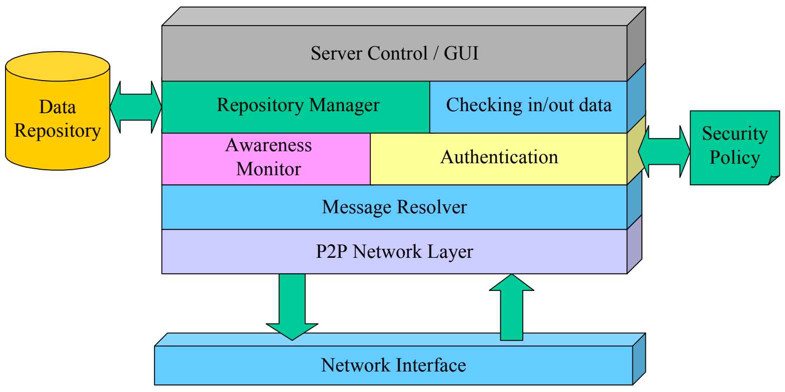


Figure 2 - Generic Co-operative Environment Server Architecture

Semi-centralised: Client Node

One of the key roles of the Client peers is to keep the server informed of their own status and likewise to be kept informed by the server on the status of other peers that they are aware of. The Awareness Controller typically carries out such tasks. Again the Message Resolver takes incoming data messages, and after some form of processing passes them on to their destination.

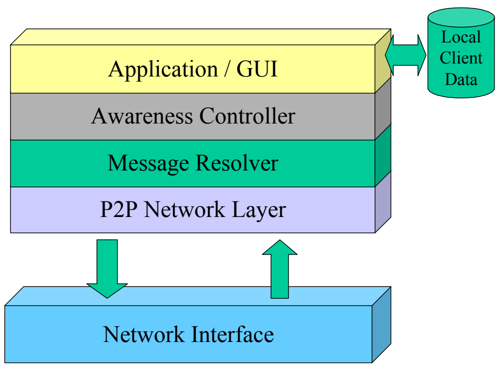


Figure 3 - Generic Co-operative Environment Client Architecture

Decentralised Node

The main difference between decentralised and semi-centralised architectures is that decentralised nodes need to handle the publishing and discovery of peers/resources themselves (rather than relying on a server), and also need to be able to route messages to other peers. Such additional functionality is typically incorporated into the P2P Network Layer, and normally involves the creation and publication of adverts onto the network. In addition, some form of discovery mechanism is required that can propagate discovery requests around the network and collate any responses. In order to reduce the over use of such a mechanism, discovered adverts would typically be stored in a cache allowing them to be re-used at a later date. Cached adverts can be assigned a lifetime, so that the peer/resources need to be rediscovered after a certain point.

A routing mechanism is also needed so that messages not for that peer can be forwarded onto other known peers (based on peer adverts in the cache). Normally messages that have been received by the peer are stored for a short time so that if the same message arrives again, it is not repeatedly forwarded around the network. A cache is typically used to store such messages.

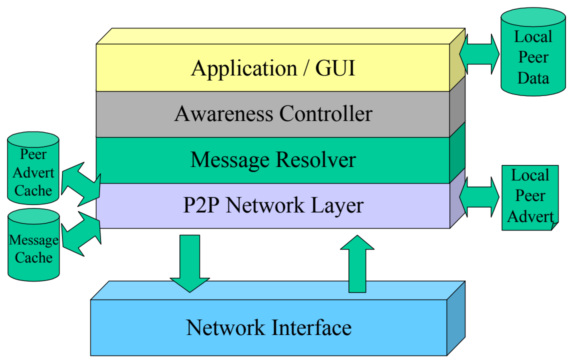


Figure 4 – Generic Co-operative Environment Decentralised Architecture

5. Layer Descriptions

As illustrated in the previous section, the presented application reference architectures are composed of various *layers*. This section moves on to discuss these individual layers in more detail. The main functional capabilities that can exist at each layer are also identified. The common layers are discussed first, followed by layers specific to the client, server or decentralised nodes. Descriptions are provided for the layers that comprise the generic architectures and the domain specific ones that are presented in section 6.

5.1. Common Layers

These layers are common to all types of node (i.e. client, server and decentralised).

Network Interface Layer - This layer represents a nodes physical connection to a network. Above this is typically the operating system, however the operating system is not described here.

The main capabilities of this layer include:

- **Software Interface** - Providing a software interface to the hardware network controls (Transport and Network layers of the ISO/OSI Network Model [18])
- **Utilising the Transmission Control Protocol (TCP)[24]** – built on top of Internet Protocol (IP) and adds reliable communication, flow-control, multiplexing and connection oriented communication.

- **Utilising the User Datagram Protocol (UDP)[25]** – built on top of IP and provides simple, efficient but unreliable datagram services.

P2P Network Layer - This layer encapsulates all the connection/communication components of a P2P application, and can possibly be considered as P2P middleware. Essentially it deals with all the basic P2P communication and organisation that can occur across the P2P network. This includes, for example, the sending of messages between peers, the local caching of peer addresses (if required), and publishing/discovery mechanisms. It is likely that higher-level parts of the application would fine-tune this layers functionality for their own individual usage. To an extent some existing P2P API's and implemented protocols, such as JXTA, Pastry, etc, already provide the user with similar functionality as described here.

There is also functionality in the other layers that could in theory be incorporated within this layer. Where this may be the case it is indicated in the text.

The main capabilities of this layer include:

- **Communication** - Establishing connections and communicating data between peers.
- **Connection Monitoring** - Monitoring connections that have been established and ensuring Quality of Service (QoS).
- **Message Creation** - Creation of message packages that can be sent between peers
- **Message Decomposition** - Decomposition of message packages
- **Communication Security** - Ensuring security of any communication via the use of appropriate protocols, encryption, etc
- **Advert Management** - Administrating peer/resource adverts and if necessary caching them.
- **Publishing and Discovery** - Peer/resource publishing and discovery mechanisms.
- **Peer Awareness**- To support awareness of peers, users and resources within the network

Message Resolver Layer – The Message Resolver layer allows the higher-level part of the application to send and retrieve data. Typically it packages any data to be sent into an appropriate format (for example, using the Extensible Markup Language (XML[26])) and passes these message packets down to the P2P Network Layer for sending. Incoming messages from the P2P Network Layer are similarly stripped down and the enclosed data passed up to the higher layers. Some processing of the incoming data can also occur here to allow for intelligent decisions about which part of the application requires the data. It is also feasible for much of the functionality to be incorporated within the P2P Network Layer, and thus removing the need for a separate layer.

The main capabilities of this layer include:

- **Message Creation** - Creation of message packages that can be sent between peers
- **Message Decomposition** - Decomposition of message packages
- **Message Routing** - Route received messages to the relevant part of the application so that it can be dealt with

Real Time Connection Monitor (used in section 6.2) - This layer deals with the monitoring of connections in real time. It tackles bandwidth requirements and aids in maintaining a high Quality of Service (QoS). This is especially important for applications that incorporate video conferencing or streaming content. This layer could also be incorporated within the P2P Network Layer.

The main capabilities of this layer include:

- **Connection Monitoring** - Monitoring connections that have been established and ensuring Quality of Service (QoS).
- **Bandwidth Resolving** - Resolve bandwidth requirements that may arise during the applications usage

Workpackage Manager Layer (used in section 6.5)- A distributed computational based application needs to breakdown the computational tasks into work packages. This layer deals with the assigning and managing of work packages that are to be processed within the application, as well as managing the results that are returned.

The main capabilities of this layer include:

- **Workpackage Creation** - Breaking down a computational task into work packages
- **Workpackage Management** - Managing the storage of these work packages on the peer
- **Workpackage Assignation** - The assigning of work packages to peers to be processed
- **Results Collection** - The collating together the processed results that are returned by the peers
- **Error Checking** - Error checking to ensure processed results have not been tampered with, or to ensure no errors occurred during processing

5.2. Server Specific Layers

These are layers that are used within server architectures.

Repository Manager Layer – This layer implements interfaces to any external data sources. It typically co-operates with the check in/out data and authentication layers for correct accessing of the held data. Although it is possible for this layer to be embedded within the P2P Network Layer it is highly unlikely, as it would tend to be accessed and manipulated from a higher level within the application.

The main capabilities of this layer include:

- **Repository Administration** - To connect to and administer local or external repositories
- **Transactions** - To support transactions with the repositories, most likely in a secure and concurrent fashion
- **Authentication** - To authenticate data requests and transfers with the repositories

- **History** - To maintain a history log of requests/transfers

Check in/out data Layer – Most applications will require concurrent and verified access to any data sources. This layer usually sits alongside the data source (*Repository Manager*), and provides the authentication controls and the communication mechanisms for accessing/manipulating the data. Again this is usually a high-level layer, dependent upon the specifics of the application. As such it probably would not be included within the P2P Network Layer.

The main capabilities of this layer include:

- **Transactions** - To support transactions with connected repositories, most likely in a secure and concurrent fashion
- **Authentication** - To authenticate data requests and transfers with connected repositories
- **History** - To maintain a history log of requests/transfers

Authentication Layer – Authentication is an essential attribute for business applications. This layer deals with authenticating peers connected to a P2P network and typically involves interacting with security control mechanisms such as access control lists or a security related database. This layer may also be incorporated into the P2P Network Layer.

The main capabilities of this layer include:

- **Authentication** - To authenticate data requests and transfers with connected repositories and with other peers
- **Encrypt/Decode** - To encrypt/decode data that is being transferred
- **Security Control** - To provide security control mechanisms such as access control lists

Version Control Layer (used in section 6.4) - This layer organises individual files into different versions. As files are accessed from across the network any updated files are cached and a versioning tag associated with each one. Versioning controllers traditionally work closely with document management systems (see following layer). Although they can be merged together, they are kept separate here to keep with tradition. It is unlikely that this layer would be included within the P2P Network Layer due to it being utilised at a high level within the applications.

The main capabilities of this layer include:

- **Version Management** - To manage multiple versions of data
- **Concurrency** - To ensure concurrency of data
- **Version Monitoring** - To keep track of data versions that may exist on other peers
- **Version Log** - To keep a history log of data versions and track changes

Document Management System (DMS) Layer (used in section 6.4) – This layer handles any additional functionality that a document management application might need that is not provided by the Check in/out Data and Version Control layers. This can include, for example, ensuring concurrent access of files within the network, ensuring that all files are kept up to date, and that conflicting access and updates of documents (or document sections) do not occur. Essentially it is a specialised form of the repository manager layer. As mentioned above the DMS layer can also be merged with the version control layer, however due to the fact that it operates at a high level within an application it would not be suitable to be included within the P2P Network Layer.

The main capabilities of this layer include:

- **Transactions** - To support transactions with connected repositories, most likely in a secure and concurrent fashion
- **Version Management** - To manage multiple versions of data
- **Concurrency** - To ensure concurrency of data
- **Version Monitoring** - To keep track of data versions that may exist on other peers
- **Version Log** - To keep a history log of data versions and track changes

- **Document Management Facilities** - To provide high level document management facilities such as change awareness, co-authoring of documents, etc

Awareness Monitor Layer – Awareness (or presence) [27] is one of the key operations of almost all P2P networks. This layer is responsible for maintaining knowledge of other peers within the network and through this can facilitate searching, sharing and aid in the authentication processes. This layer may be included within the P2P Network Layer.

The main capabilities of this layer include:

- **Peer Awareness** - To support and monitor peer awareness within the network
- **User Awareness** - To support and monitor user awareness within the network
- **Resource Awareness** - To support and monitor resource awareness within the network

Data Search/Filtering (used in section 6.3) - This layer represents the search/filtering mechanisms that are used to search the data that is held within the local repository (connected to via the Repository Manager). It can also be used to filter the returned search data. It is unlikely that this layer would be included within the P2P Network Layer due to its high level of operation within the application.

The main capabilities of this layer include:

- **Data Searching** - To provide mechanisms to allow for the searching of connected repositories based on specified criteria
- **Data Filtering** - To provide mechanisms to allow for the filtering of data within connected repositories. This could include, for example, collaborative filtering.

5.3. Client Specific Layers

These are the layers that are used within a Client node architecture.

Awareness Controller Layer – This layer is tightly coupled with the awareness monitor described in the server architectures. The client version, apart from negotiating with the awareness monitor on a server, will usually also keep records and awareness information with respects to other peers that have already been located through searches sent to the server. As with the Awareness Monitor, this component may also be part of the P2P Network Layer.

The main capabilities of this layer include:

- **Peer Awareness** - To support the peers awareness on the network by liasing with server and client peers
- **User Awareness** - To support the peer's user awareness on the network by liasing with server and client peers
- **Resource Awareness** - To support the peer's resource awareness on the network by liasing with server and client peers

5.4. Decentralised Specific Layers

Decentralised architectures generally use the same layers as used in semi-centralised architectures, but they need to be altered to take into account the different network architecture properties.

P2P Network Layer - This layer is essentially identical to the P2P Network Layer used for semi-centralised applications, however it also needs to deal with the publishing of data (or adverts) onto the network, the discovery of such data, and the routing of data to other peers.

The main capabilities of this layer include:

- **Communication** - Establishing connections and communicating data between peers.
- **Connection Monitoring** - Monitoring connections that have been established and ensuring Quality of Service (QoS).
- **Message Creation** - Creation of message packages that can be sent between peers
- **Message Decomposition** - Decomposition of message packages
- **Communication Security** - Ensuring security of any communication via the use of appropriate protocols, encryption, etc

- **Advert Management** - Administrating peer/resource adverts and if necessary caching them.
- **Publishing and Discovery** - Peer/resource publishing and discovery mechanisms.
- **Peer, User and Resource Awareness** - To support awareness of peers, users and resources within the network

6. Application Domain Reference Architectures

This section presents a set of reference architectures for a number of specific application domains that reflect the more common types of developed P2P applications. Existing P2P applications have tended to provide functionality that supports instant communication between users (for example, ICQ, MSN Messenger) or allows users to share resources with each other (for example, Napster, SETI@home, Gnutella and FreeNet based applications,). The set of developed reference architectures presented here reflect this trend and cater for the following domains:

- *Instant messenger P2P applications* - those that provide instant communication between users (for example, ICQ or MSN Messenger)
- *Shared workspace P2P applications* - those that provide users with a shared space in which they can interact. For example, chat rooms, shared whiteboards, etc (for example Groove [28])
- *Search based P2P applications* - those that provide users with a distributed database containing resources that peers/users make available to the rest of the network (for example Napster or Gnutella based applications). These resources can then be searched for and accessed.
- *Document management P2P applications* - those that provide users with distributed document management facilities. This has traditionally been a client-server domain.

- *Computational P2P applications* - those that provide uses with a means in which to have computational tasks carried out in a distributed fashion (for example SETI@home)

These architectures extend the generic architectures that are presented in section 4 and have been developed as a result of studying existing P2P applications within each application domain.

As before both semi-centralised and (indirect communication) decentralised architecture representations have been provided. Additionally, the main capabilities for each domain type have also been identified and presented. The individual layers that make up the reference architectures have been described in section 5.

6.1. Instant Messenger Application Architectures

Instant messenger applications such as ICQ, MSN Messenger and more recently Skype [33], allow users to communicate directly with each other and in an instantaneous fashion. The main capabilities of such applications include:

- **Peer and User Discovery** - Allowing peers and users to discover and be aware of each other
- **Peer and User Communication** - Allowing peers and users to communicate with each other
- **Peer and User ID** - Allowing peers and users to be uniquely identified on the network
- **Security** - Providing security within the application
- **Data Management** - Allowing data to be stored and managed on the peers
- **User Interface** - Allowing for the creation of an Instant Messenger user interface for the peers
- **Network Structure** - Allowing for the creation of de-centralised and semi-centralised applications

Semi-centralised: Server Node

Instant messenger applications have a high dependency upon awareness monitoring as users of the system need to be aware of the status of other users. It is possible to incorporate awareness within the P2P Network Layer but for this type of application it is more suitable for it to remain separate and at a higher level to increase its visibility to developers. Awareness can also be tied to incoming communications. For example, if the server receives a communication from a peer that it previously thought was off-line, then it can update its records accordingly. Aside from this, the server architecture operates in a similar way to the generic server discussed previously.

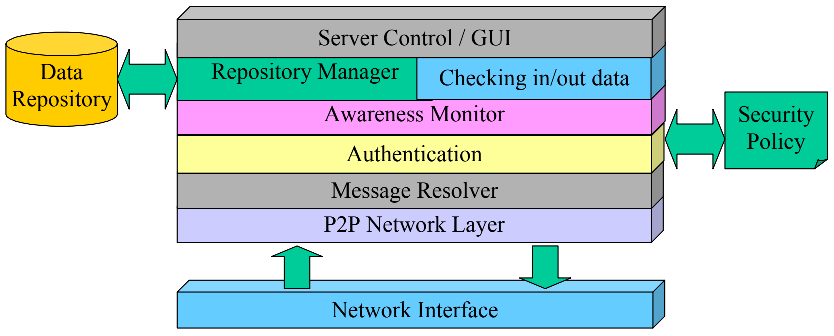


Figure 5 - Instant Messenger Server Architecture

Semi-centralised: Client Node

The client node possesses the mechanisms and interfaces for user-user communication. Given the importance of awareness in such an application the client version’s Awareness Controller will be tightly coupled with the Awareness monitor running on a server. Aside from this the client architecture operates in a similar way to the generic client discussed previously.

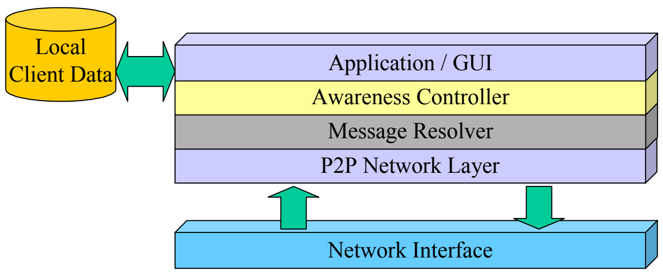


Figure 6 - Instant Messenger Client Architecture

Decentralised Node

Large scale decentralised instant messenger applications can be difficult to develop due to the restrictions posed by the network (i.e. not all users being able to discover each other, partitioning, QoS issues, etc) [23]. However, for smaller scale (and especially direct communication architectures) it may be possible to support such an application and they do remove the need for a server. A decentralised node in such an architecture will essentially be the same as a client node (detailed above), but with provision for the publishing and discovery of peer/resource adverts, and for the routing of messages. One issue that will certainly need to be considered is how users of the system can be provided unique ID's.

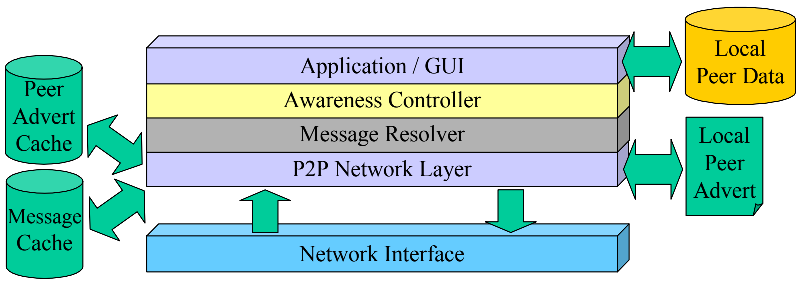


Figure 7 – Instant Messenger Decentralised Architecture

6.2. Shared Workspace Application Architectures

Shared Workspace applications include those that support chat rooms, shared whiteboards, video conferencing, etc, (as demonstrated by the Groove P2P software). The main capabilities of such applications include:

- **Peer and User Discovery** - Allowing peers and users to discover and be aware of each other
- **Peer and User Communication** - Allowing peers and users to communicate with each other
- **Peer and User ID** - Allowing peers and users to be uniquely identified on the network
- **QoS** - Ensuring a high QoS for communications
- **Security** - Providing security within the application
- **Data Management** - Allowing data to be stored and managed on the peers

- **User Interface** - Allowing for the creation of a shared workspace user interface for the peers
- **Network Structure** - Allowing for the creation of de-centralised and semi-centralised applications

Semi-centralised: Server Node

Communication in real time is a crucial aspect for most shared workspace applications. The Real Time Connection Monitor layer is responsible for acting upon information obtained from the Awareness Monitor, Repository Manager and Check in/out data layers to keep shared workspaces synchronised. Once any data or request has passed through the P2P Network Layer and is decomposed by the Message Resolver, it is passed through the Real Time Connection Monitor layer if there are time constraints attached to it (i.e. a global update of a workspace, video conferencing may be in use or streaming of some form of media). The Real Time Connection Monitor can then make a decision to attach a higher priority to the data/request if needs be.

The server node within a shared workspace application typically creates and manages workspaces, and ensures that all the relevant parties are kept up to date.

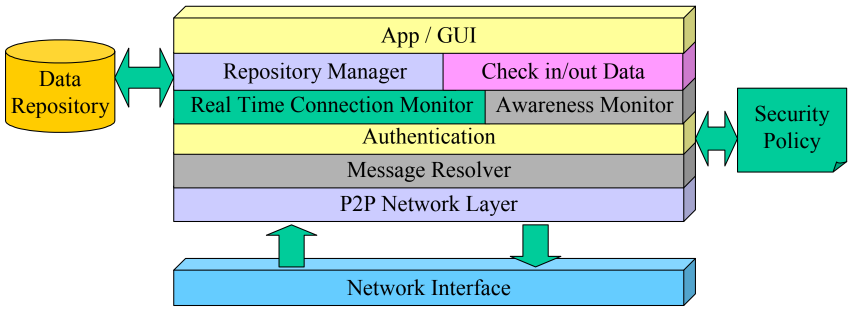


Figure 8 - Shared Workspace Server Architecture

Semi-centralised: Client Node

As with the server node it is the Real Time Connection Monitor that is the crucial component for the client. Aside from this, the client architecture operates in a similar way to the generic client discussed previously, liasing with the server for updates from relevant shared workspaces.

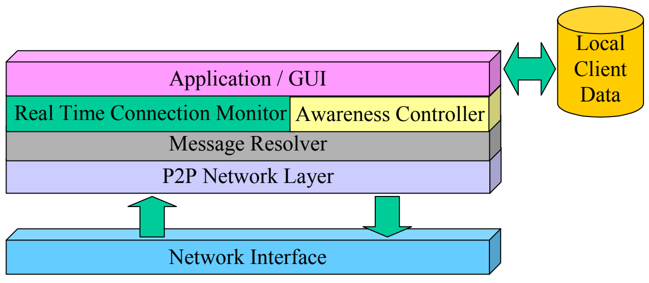


Figure 9 - Shared Workspace Client Architecture

Decentralised Node

Achieving a truly decentralised shared workspace application is particularly difficult due to the fact that the workspace itself needs to be managed and achieving this in a decentralised manner is unlikely to be a simple task. One possible solution is to allow any peer on the network to create and manage a shared workspace. This, in essence, means that each peer can effectively take on a role as a server within the network (handling authentication, QoS issues, etc). An architecture for such a node is similar to that used for a server node (described above), but also including the requisite mechanisms required for decentralised applications (e.g. publication, discovery and message routing).

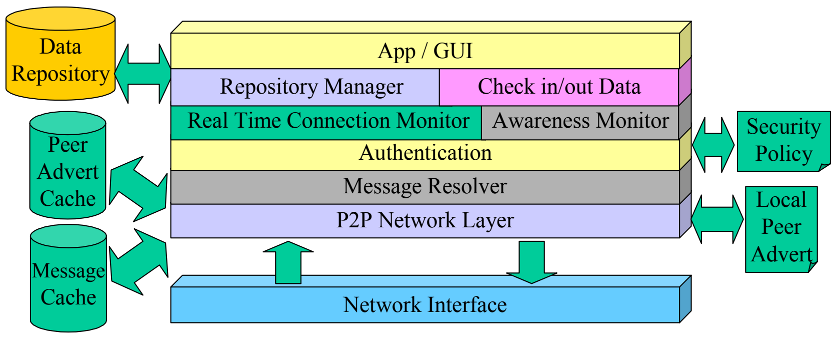


Figure 10 – Shared Workspace Decentralised Architecture

6.3. Distributed Search Application Architectures

Search applications allow users/peers to search for and obtain data that is distributed around the P2P network (example applications include Napster, Morpheus and FreeNet/Gnutella based applications). The main capabilities of such applications include:

- **Peer Discovery** - Allowing peers to discover and be aware of each other
- **Resource Discovery** - Allowing peers to discover and be aware of resources (for example, data) that may exist on the network
- **Peer Communication** - Allowing peers to communicate with each other, including allowing the transference of data
- **Resource Searching** - Allowing peers to search the network for resources, potentially using a variety of filtering techniques
- **Peer and Resource ID** - Allowing peers and resources to be uniquely identified on the network
- **Security** - Providing security within the application
- **Data Management** - Allowing data to be stored and managed on the peers
- **User Interface** - Allowing for the creation of a search application user interface for the peers
- **Network Structure** - Allowing for the creation of de-centralised and semi-centralised applications

Semi-centralised: Server Node

The server version of this architecture domain typically utilises mechanisms for searching and filtering data that is indexed within a local data repository. When a search request comes in it is passed to the Data Search/Filtering layer where it is acted upon. The Data Repository is then interrogated and the returned results can be crosschecked against current online peers if necessary. Returned results belonging to a peer that is offline or is overloaded can be filtered out. The final set of search results is then returned back to the requesting peer. Server nodes need to ensure that the indexed data is kept up to date.

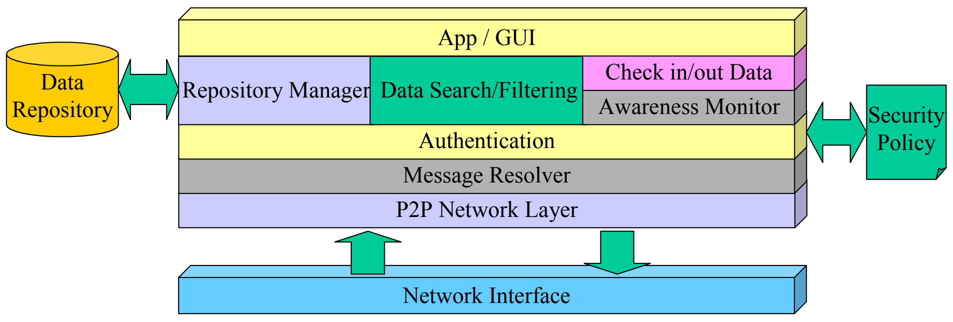


Figure 11 - Search Application Server Architecture

Semi-centralised: Client Node

It is important for the server to be aware of what data a client peer is making publicly available and what client peers are on-line. As a result the Awareness Controller within the client is liable to be tightly coupled to the Awareness Monitor within the server. Typically a client node supplies details of locally stored data at start-up, or whenever the data is changed in someway (for example, new data being added). When a client receives the search results from the server, usually the details regarding the location and authentication requirements of these peers (if needed) are also included. This allows the client peer to then establish a direct connection with the target peer.

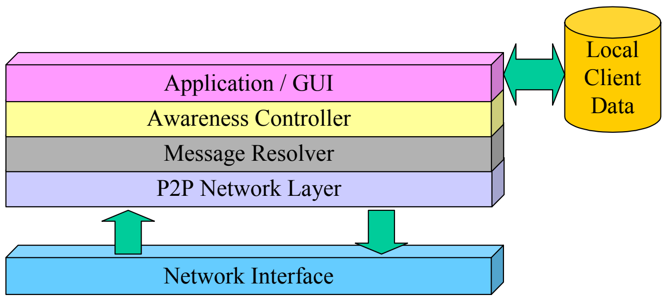


Figure 12 - Search Application Client Architecture

Decentralised Node

Decentralised search applications are already quite common (Gnutella and FreeNet based applications, etc). The main advantage of using a decentralised approach is that there is no need for an index server, and thus no single point of failure. On the negative side it might not be possible to search every node on the network, and the actual search process is less efficient.

A node within a decentralised application possesses the same functionality as a client node described above, but also needs to be able to handle the data searching aspect as well. When that node receives a search request then it needs to perform a search on the data that is held locally. In addition the node also needs to handle the decentralisation specific issues discussed previously (e.g., publishing, discovery and message routing).

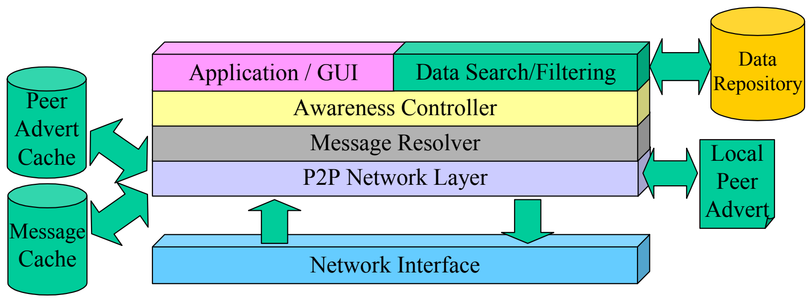


Figure 13 – Search Application Decentralised Architecture

6.4. Document Management Application Architectures

Document Management applications are similar to search applications, but place more emphasis on managing the data that is distributed throughout the network. The main capabilities of such applications include:

- **Peer and User Discovery** - Allowing peers and users to discover and be aware of each other
- **Document Discovery** - Allowing peers and users to discover and be aware of documents that may exist on the network
- **Peer and User Communication** - Allowing peers and users to communicate with each other, including allowing the transference of documents
- **Peer, User and Document ID** - Allowing peers, users and documents to be uniquely identified on the network
- **Version Management** - Supporting the versioning of documents, change tracking, concurrency and various other document management facilities
- **Security** - Providing security within the application

- **Data Management** - Allowing data to be stored and managed on the peers
- **User Interface** - Allowing for the creation of a document management user interface for the peers
- **Network Structure** - Allowing for the creation of de-centralised and semi-centralised applications

Semi-centralised: Server Node

Server nodes within a semi-centralised document management application typically store and manage the documents that exist within the system, and also possess functionality to support version control and document management facilities (represented by the DMS layer and Version Control layer). The Authentication layer, aside from authenticating peers on the network, can also authenticate access to the documents.

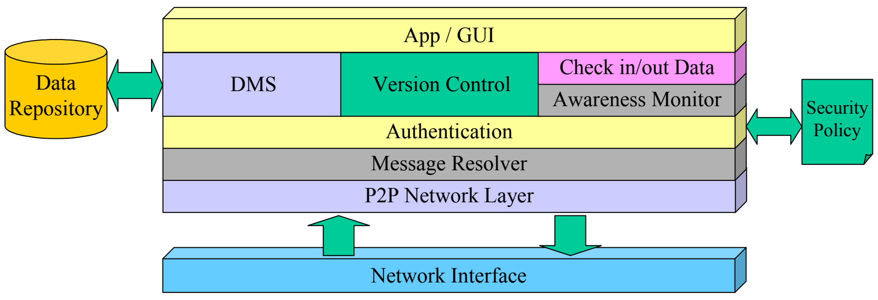


Figure 14 - Document Management Server Architecture

Semi-centralised: Client Node

As with most of the other client architectures, the Awareness Controller is one of the more important components for this kind of application. It needs to maintain contact with both the server and other client peers in an attempt to keep the consistency of sharable documents. Client nodes typically request documents from the server, and upload them if necessary. Client nodes could also distribute documents to other client nodes, but the server would need to be informed so that it maintains a consistent view of the documents within the system.

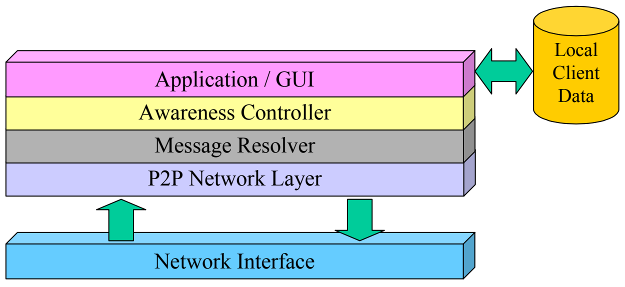


Figure 15 - Document Management Client Architecture

Decentralised Node

As with shared workspace applications, it is difficult to make a dependable decentralised document management application due to the fact that these types of applications rely on central points of control. Within a decentralised application keeping track of documents within the network will be much more difficult, and so concurrency issues may arise. Such issues could be caused by the latest version of a document being offline when it is required, or by multiple copies of the document being edited at the same time. For a decentralised document management application to succeed it will be necessary to resolve such issues. As with other decentralised solutions, the main advantage of using such an approach is the removal of the single point of failure.

The layered architecture presented here assumes that such issues can be resolved. It extends the client architecture presented above by including mechanisms to handle the different document versions that are likely to exist on the network. In addition the node would also need to handle the decentralisation specific issues (e.g., publishing, discovery and message routing).

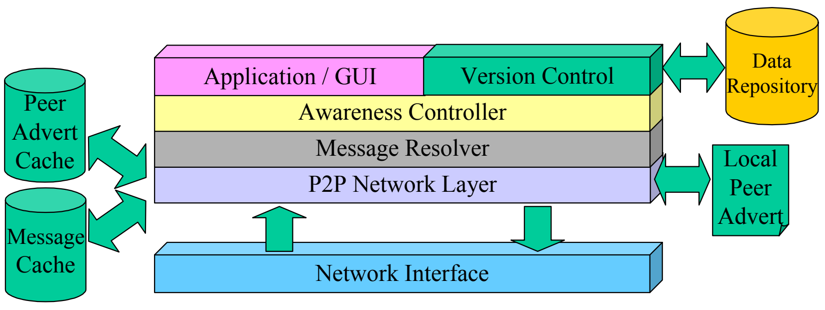


Figure 16 – Document Management Decentralised Architecture

6.5. Computational Application Architectures

Computational applications rely heavily on one or more server peers distributing work (work packages) to a network of peers and collating the returned results (an example application being SETI@home). Although it has been debated whether or not the master-slave styled relationship that typically exists within such applications is real P2P, applications such as SETI@home are generally accepted as being P2P due to the fact that they exploit resources on the edge of the network [29].

The main capabilities of such applications include:

- **Peer Discovery** - Allowing peers to discover and be aware of each other
- **Peer Communication** - Allowing peers to communicate with each other
- **Peer ID** - Allowing peers to be uniquely identified on the network
- **Work package Distribution** - Allowing for the breakdown of work into work packages that can then be distributed throughout the network for processing
- **Work package Tracking** - Keeping track of work packages within the system (who is processing them, etc)
- **Result Collection** - Allowing for the collation of results, error checking and redundancy
- **Security** - Providing security within the application
- **Data Management** - Allowing data to be stored, managed and processed on the peers
- **User Interface** - Allowing for the creation of a computational application user interface for the peers (if needed)
- **Network Structure** - Allowing for the creation of de-centralised and semi-centralised applications

As indicated in our previous work [16], it is not unreasonable for client peers within a computational application to possess autonomy and to communicate directly with each other. This, for example, can allow client peers to communicate results to each other, or possibly to work together as a cluster. The architectures presented here represent those computation applications where the client peers do possess some form of autonomy.

Obviously one of the most important aspects of computational applications is that the actual computation that is carried out is correct and not compromised. As a result the best policy to adopt within P2P applications is to incorporate redundancy into the system, i.e. having a work package processed by multiple peers and comparing the results. Ideally these peers' hardware profiles would also be different (i.e., different operating systems, etc).

Semi-centralised: Server Node

The architecture of a server peer operated in a very similar manner to the document management application. However rather than managing documents the server instead manages work packages. This includes sending out work packages to client peers, collating the results, comparing the returned results (if redundancy is used), and also keeping track of which client peers are processing which packages. As mentioned, incorporating redundancy is beneficial and so results returned by client peers need to be authenticated and compared.

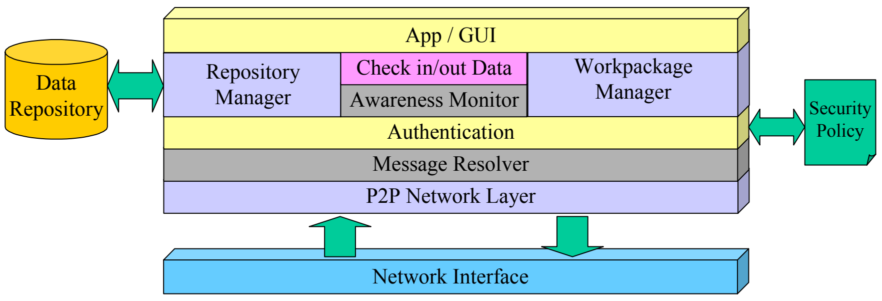


Figure 17 – Computational Application Server Architecture

Semi-centralised: Client Node

The client peers essentially do nothing more than process the work packages that they have been assigned. Such activity is usually transparent to the actual user of the peer. If the client peers possess autonomy then they might also communicate with other client peers. This can, for example, allow client peers to work together in clusters. Obviously, security and trust will be important issues in such scenarios.

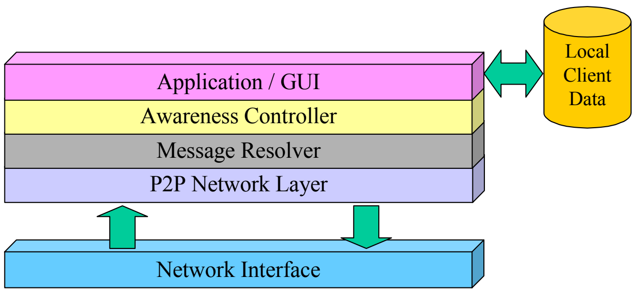


Figure 18 – Computational Application Client Architecture

Decentralised Node

Decentralised computational applications raise a number of design issues. As with the decentralised shared workspace architectures, they will be decentralised in the fact that any peer on the network can initiate a distributed computation, but also centralised in the fact that the peer will need to co-ordinate the work. The lack of control, the unpredictable nature of the network and the need to assign peers unique ID's may hinder this process, and so such an application will need to be carefully designed. Decentralised applications are likely to make more use of redundancy, and possibly allow peers to further delegate work in a hierarchical fashion.

The decentralised layered architecture extends the client architecture presented above by including mechanisms to manage the work packages. In addition the node also needs to handle the decentralisation specific issues (e.g., publishing, discovery and message routing).

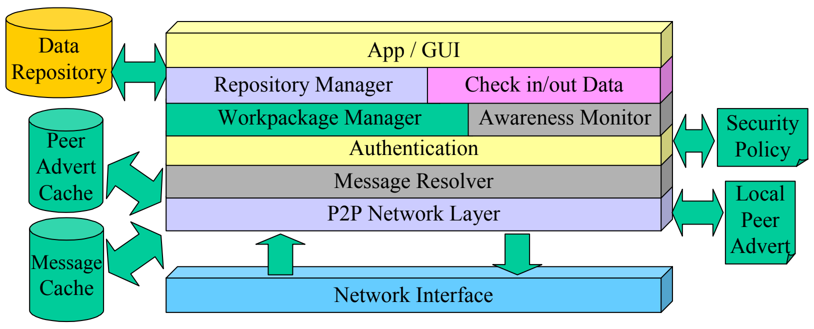


Figure 19 – Computation Application Decentralised Architecture

7. Architectural comparisons

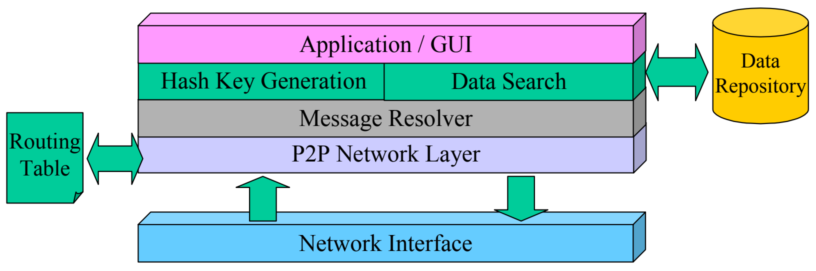
This section compares four existing P2P applications with the reference architectures that have been presented in this document. Not only does this provide an analysis of how these applications architectural structures compare with the relevant reference architectures, but it also illustrates one way in which the reference architectures can be used.

It is important to remember that the application reference architectures represent abstract functionality and structure rather than detailed descriptions of application functionality. Consequently, although the discussed applications might possess similar structure the majority have notable differences in the functionality of the layers.

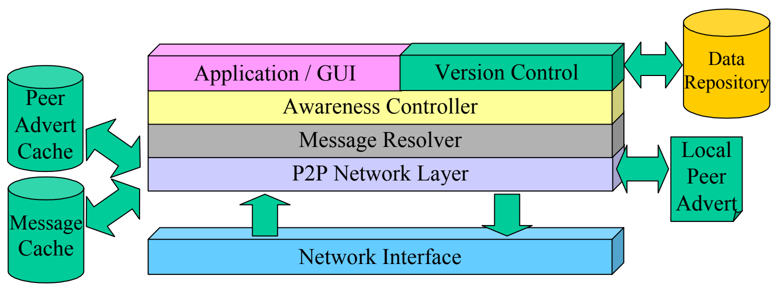
7.1. Frost - a FreeNet based client

FreeNet is a decentralised P2P protocol used to provide secure global information storage. The general idea behind FreeNet is that each peer donates a certain amount of disk space to the system, which can then be used to store other user's data. Data storage and retrieval is done in an entirely anonymous fashion. A number of applications have been developed based on the FreeNet protocol including Frost - an application used for file sharing.

In respect to the reference architectures, Frost can be considered to be a simple version of a decentralised document management application (discussed in section 6.4). However unlike the architectures that have been provided in this document, because FreeNet is based on providing anonymity, Frost cannot provide sophisticated document management facilities such as versioning or authentication. Instead there is a much greater importance on the use of hash keys as identifiers for stored data. When it is desired to retrieve a certain piece of data then the network is searched for the relevant hash key. Figure 20 provides a general layered based approximation of how Frost is structured.



(a)



(b)

Figure 20 – a) Layered architecture representation of Frost), b) Document Management Decentralised Reference Architecture (from section 6.4)

In comparison with the decentralised document management reference architecture, it can be seen that there is a large degree of similarity (with Frost perhaps being simpler in structure). Due to the desire for anonymity the workings of the P2P Network Layer differ slightly as there is no need for peer adverts. Instead each peer maintains a routing table that stores details of any other known peers and what data they store (as hash keys). As a result the Frost representation lacks the *Peer Advert Cache* and the *Local Peer Advert* aspects, but gains a *Routing Table.*

Frost does have to resolve messages, but there are only four types it actually needs to be able to handle. Although messages can be routed to other peers, each peer does not really possess any form of cache to store these messages. Furthermore, Frost does not really possess any form of awareness control; it simply updates the routing table depending on what messages/data is routed through it. As a result of these differences the Frost representation lacks the *Awareness Controller* and *Message Cache*.

The main differences between Frost and the reference architectures occur with the actual handling of the stored data. As has been mentioned, the FreeNet protocol deals with this in an anonymous fashion and so Frost cannot possess general document management facilities (such as versioning). Instead at this higher level, the application deals with the generation of hash keys to act as indexes for the stored data. These differences are represented by the replacement of the *Version Control* aspect with *Hash Key Generation* and *Data Search* aspects.

7.2. Napster

Napster is a semi-centralised file-sharing application, in which users can search for data being held by the other peers that make up the system network. Server nodes maintain a catalogue of what data is currently available on-line which client peers can then interrogate. Client peers can then form direct connections between other peers, in order to download the data. Napster also supports simple chat rooms that are hosted on the server nodes.

The structure of the peers in Napster can be compared to an extent with a hybrid of the search application and shared workspace reference architectures. The server nodes within the network have to maintain the catalogue of data in real-time, as well as allowing it to be searched. In order to keep the catalogue up to date the server nodes need to be aware of what is available on the network, and the application uses similar presence techniques as those discussed in our previous work [19]. Users need to register with the server node before they can make use of the application, and this allows Napster to provide simple authentication. Similar functionality is represented in the search application server node reference architecture.

Napster only provides very simple shared workspace support in the form of chat rooms. Consequently it does not concern itself with ensuring that a real time connection exists between peers. The server attempts to update interested client peers of changes in the chat room as soon as possible. Figure 21 provides a general layered based approximation of how a Napster server is structured. The only difference to the search application server reference architecture presented in section 6.3 is the replacement of the generic *Data Repository* with separate *User Data* and *Data Catalogue* repositories.

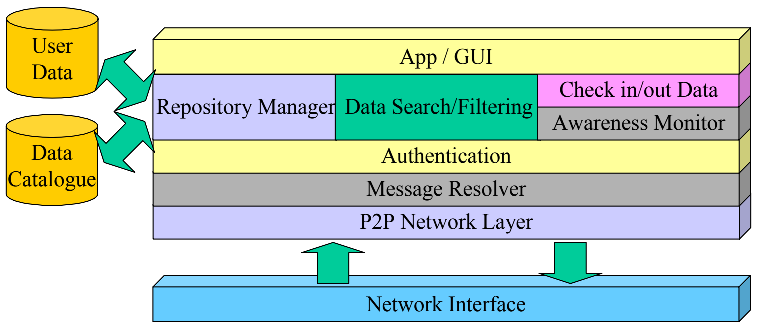


Figure 21 – Layered architecture representation of the server peers in Napster

The structure of the client nodes within Napster closely mirrors that which is presented in the search application reference architecture. Users create a public folder on their machine and the details of any data that is stored in there is uploaded to the server nodes to be catalogued. Once the clients receive details of the location of data, then a direct connection link can be established between the two client peers to allow for it's downloading. Communication between users in the chat rooms is not done directly, but via the server nodes. When the client peer connects and disconnects from the Napster network, the server nodes are informed to keep the applications presence information up to date. Similar functionality can be represented by the search application client node reference architecture (figure 12).

In summary, the structure of Napster is comparable to a hybrid of the search application and shared workspace semi-centralised reference architectures. However, the shared workspaces supported by Napster are only very simplistic (chat rooms) and a real time connection is not critical.

7.3. SETI@home

SETI@home is a semi-centralised computational application that makes use of spare CPU cycles on client peers to process data measured by radio telescopes utilised by SETI. The client peers within the system do not possess any control over what data they can process nor can they communicate with one another. They essentially are nothing more than slaves to a central node.

Consequently it has been debated whether SETI@home is P2P, however it is generally accepted that it is as it utilises resources that reside on the edge of the network [29].

The structure of the client nodes within SETI@home essentially mirrors that as presented in the client computational reference architecture (section 6.5). Work packages are sent from the server nodes and are then processed by any spare CPU cycles that the client peer possesses. In most cases this processing is tied in with a screen saver and so only takes place when a user is not using the peer. The clients cannot, however, communicate with each other.

The server nodes also possess a very similar structure to the reference architectures. Three databases are used that store information about the users who are registered with the application, scientific information about each work package, and the storage of work packages and their results. In order to remove errors, SETI@home sends the same work package out to multiple peers for processing. When processed data is received it is first authenticated (with the user's username and password), then examined to see if the results match permitted values and then finally cross checked with the processed results from other peers. The authentication and work package manager layers as depicted in the reference architecture would carry out such error checking. Figure 22 provides a general layered based approximation of how the SETI@home server node is structured. The only difference to the computational application server reference architecture presented in section 6.5 is the replacement of the generic *Data Repository* with separate *User, Science* and *Work package Database's*.

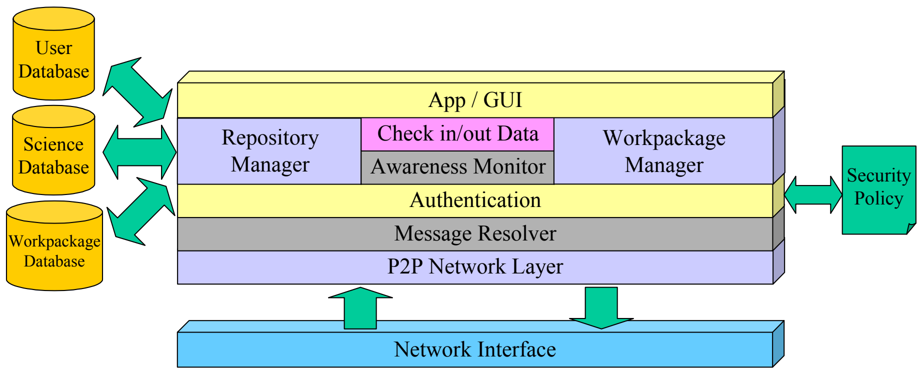


Figure 22 – Layered architecture representation of the server peers in SETI@home

In summary, the structure of SETI@home is comparable to the semi-centralised computational reference architecture presented in section 6. The server nodes make use of three databases to store data that is required by the application. The client nodes do not possess any autonomy.

7.4. Jabber based applications

Jabber is coined as a 'universal' Instant Messenger (IM) protocol. It is different to more widely known IM protocols such as those used within ICQ and MSN Messenger because its XML core allows, in theory, a Jabber based application (for example, MirandaIM) to connect to all other IM applications. In order to achieve this, whereas in other IM applications (like ICQ) all client nodes connect to a server, in a Jabber based application there exists many server nodes which are in turn connected together and all communication is routed through them (similar to how email functions). For example, for two clients peers to communicate with each other, the first client peer sends the message to the server it is connected to, this then routes it to the server that the target client is connected to (possibly routed via other servers in the process), and finally the message is passed onto the target client. There currently exists a large number of Jabber based applications, a list of which can be found here [30].

In terms of logical network architecture, Jabber can be considered to use a semi-centralised structure, but one involving multiple server nodes (*g* in figure 1). This means that on one hand Jabber based applications can operate in a similar fashion to alternative IM applications, in that local client peers (those that use the same server) can communicate directly with each other with some initial help from a server peer. But in addition, the server nodes also connect to each other in a more decentralised fashion.

The structure of the client nodes is comparable with the IM client reference architecture (section 6.1). Communication between other client peers that are supported by the same server can be established directly, after an initial lookup is made with the server. However communication between other clients that are not supported by the same server (i.e. on a different IM network, e.g., MSN and ICQ) cannot be established directly, and instead has to be routed via the servers. Communication here not only represents the messages that are sent between clients, but also awareness information.

The structure of the server nodes is also comparable to the IM server architecture. The main difference is the additional functionality that is required to communicate with other servers. In particular, the server node needs to store additional information about the other servers it can connect to, and the Message Resolver and P2P Network layers need cater for server-server communication. Figure 23 provides a general layered based approximation of how typical Jabber application server nodes are structured. The only differences to the instant messenger server reference architecture presented in section 6.1 is the replacement of the generic *Data Repository* with separate *Server Details, User Details* and *Messages to be routed* repositories.

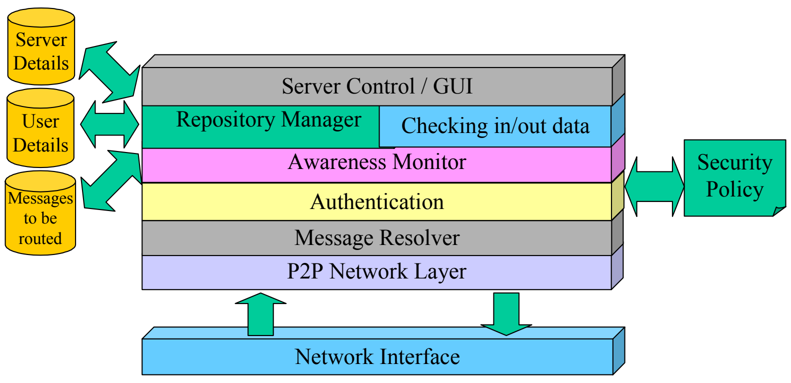


Figure 23 – Layered architecture representation of the server peers in typical Jabber clients

In summary, the structures of Jabber based applications do not differ significantly from the IM reference architectures that have been presented in section 6. However, the functionality possessed by the layers (in particular with the server), is significantly different from the equivalent layered functionality that is possessed by other IM applications such as ICQ, MSN Messenger, etc. This is mainly due to the fact that a Jabber server can connect to other servers and also to other IM protocols.

8. Related Work

As previously mentioned, the vast majority of P2P research and developments have focused on lower level issues such as protocols and API's. There has been less attention paid towards higher-level P2P application issues such as their design and specification. To an extent we have tried to address this within the P2P ARCHITECT project, which has specifically focused on the development of methods and tools to support the building of dependable P2P applications. Although general reference architectures have been developed for the various substrates that exist, to our knowledge this has not been the case for common P2P application domains (aside from what is presented here).

The applicability of P2P technology as a basis for different types of applications has been examined, however, with Roussopoulos et al. [31] identifying a number of characteristics that should be considered. These include the budget for the development, the relevance of resources to the participants of the application, the role of trust within the application, how frequently the application will change, and how critical its operation is. Initial analysis they have performed indicates that P2P technology would be less suitable for applications that are critical or possess a high rate of change. Such conclusions are supported by our previous work [16], which illustrated that properties such as safety and maintainability can be difficult in some types of P2P systems.

9. Experiences and Future Work

As part of the P2P ARCHITECT project the developed methods and tools were assessed by two end user industrial partners, who used them to develop real P2P applications for use within their own organisations. As part of the design process the user partners made use of the application reference architectures that have been presented, selecting the ones that were appropriate for their application requirements and using them to inform their designs.

Our Greek user partner, wished to develop a P2P application to support their magazine development. In particular they wanted to facilitate collaboration between writers and editors in terms of instant messaging and joint document editing. As part of the design process they made use of the Instant Messenger and Shared Workspace application reference architectures to help in developing their own design.

Our Italian user partner, wished to develop a P2P application to support on-line theatre bookings. Within such a system ticket allocation would be spread between many box-office peers scattered around the country, who could transfer spare tickets between themselves if needs be. As part of the design process they made use of the Document Management, Distributed Search and Computational application reference architectures to help structure their own design.

Feedback from the user partners with regards to the application reference architectures indicated that they were generally well received and found to benefit the design process by helping them not only to structure their designs but also to identify additional functionality areas they had not considered. Observations indicated that it would often take a number of iterations of the design process before designers felt they were able to select appropriate reference architectures, but when this was achieved the reference architectures were found to be a valuable input to the design.

It is our intention to obtain additional feedback on the use of the reference architectures by utilising them within further P2P application development projects. This will not only include ongoing projects within our department, but hopefully also within further industrial related projects. Further refinement of the reference architectures could occur as a result of such feedback.

It is also our intention to increase the number of reference architectures to cover other application domains, for example distributed storage mechanisms such as a P2P based federated database. Again these will be evaluated in conjunction with suitable application developments.

10. Summary

This paper has presented a set of application reference architectures that can be a valuable resource for developers of co-operative P2P applications. These reference architectures can be used for activities including allowing designers to gain a high level understanding of how certain types of P2P applications function and our structured, being used as a basis for architectural comparisons with existing P2P application designs, or to act as possible starting points for future P2P application design.

As a result of our work within the P2P ARCHITECT project, the paper has provided reference architectures for generic co-operative P2P applications, instant messenger P2P applications, shared workspace P2P applications, distributed search P2P applications, document management P2P applications and computational P2P applications. These architectures have been derived from studying existing and widely used P2P applications. A layered based notation has been used to depict the architectures, and both semi-centralised and decentralised models have been catered for. Identifying the key capabilities that reference architectures, and the layers that comprise them, possess has further enhanced their description. A capability is regarded as being an abstract system functionality.

The paper has also compared a selection of existing P2P applications with the reference architectures, to examine what similarities and differences exist.

11. References

1. The EC funded P2P ARCHITECT project (IST-2001-32708). More information can be found at the URL http://www.atc.gr/p2p_architect/index.htm
2. M. Wilson, Standards and Reference Architectures: Their Purpose and Establishment, Invited Talk at the Workshop in Reference Architectures and Data Standards for NLP, AISB'99.
3. C. Shirky, Listening to Napster, *in Peer-to-Peer: Harnessing the power of Disruptive Technologies*, Oram, A., (editor), O'Reilly publishing, 2001
4. Napster. MP3 file sharing application. More information can be found at http://www.napster.com
5. ICQ. Instant Messenger Application. More information at the URL http://www.icq.com
6. The SETI@home project. Distributed computation application. SETI. More information can be found at the URL http://setiathome.ssl.berkeley.edu/
7. The gnutella protocol specification v0.4. Clip2 Distributed Search Services. Available from http://www9.limewire.com/developer/gnutella protocol 0.4.pdf.

8. MSN Messenger, Microsoft Corporation. More information can be found at the URL http://messenger.msn.com

9. The Free Network P2P project (FreeNet). Anonymous file sharing application. More information can be found at http://freenet.sourceforge.net/

10. Morpheus. File sharing application. More information at the URL http://www.morpheus.com/

11. Jabber. Streaming XML protocols and technologies, developed by the Jabber SoftwareFoundation. More information can be found at the URL http://www.jabber.org

12. Frost. FreeNet based file-sharing application. More information at the URL http://jtcfrost.sourceforge.net/

13. Gnucleus. Gnutella based file-sharing application. More information at the URL http://www.gnucleus.com

14. Limewire. Gnutella based file-sharing application. More information at the URL http://www.limewire.com

15. MirandaIM. Jabber based Instant Messenger application. More information at the URL http://miranda-im.org/

16. L. Melville, J. Walkerdine, I. Sommerville, Report on the dependability properties of P2P architectures, *Information Societies Technology Institute, IST-2001-32708*, 31st July, 2002

17. J. Walkerdine, L. Melville, I. Sommerville, Dependability Properties of P2P Architectures, *In the proceedings of IEEE P2P 2002*, Sweden, 5th-7th September 2002

18. H. Zimmermann, OSI reference model - the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, COM-28 (4), 452-32.

19. A. Rowstron, P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November, 2001, pp. 329-350.

20. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, *ACM SIGCOMM 2001*, San Deigo, CA, August 2001, pp. 149-160.

21. Zhao, B., Kubiatowicz, J., Joseph, A., Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, Computer Science Division, U.C. Berkeley, April 2001.

22. JXTA v2.0 Protocols Specification, Sun Microsystems, The Internet Society, 2001-2003. More information can be found at the URL http://www.jxta.org

23. N. Minar, M. Hedlund, A Network of Peers - Peer-to-Peer Models Through the History of the Internet, *in Peer-to-Peer: Harnessing the power of Disruptive Technologies*, Oram, A., (editor), O'Reilly publishing, 2001

24. J. Postel, The Transmission Control Protocol (TCP), Standard RFC0793, 1981

25. J. Postel, User Datagram Protocol (UDP), Standard RFC0768, 1980

26. Extensible Markup Language 1.0 (XML), WWW Consortium, 2000

27. J. Walkerdine, L. Melville, I. Sommerville, Providing Presence within P2P systems, *Technical Report COMP-003-2004, Computing Department, Lancaster University*, 2003

28. Groove Peer Computing Platform, Groove Networks Inc., 2000. More information can be found at the URL http://www.groove.net

29. D. Anderson, SETI@home, *in Peer-to-Peer: Harnessing the power of Disruptive Technologies*, Oram, A., (editor), O'Reilly publishing, 2001

30. List of current Jabber based application can be found at the URL http://www.jabber.org/software/clients.php

31. Roussopoulos, M., Baker, M., Rosenthal, D. S. H., 2 P2P or Not 2 P2P?, *In the proceedings of IPTPS 2004*

32. M. Castro, M. Costa, A. Rowstron, Should we build Gnutella on a structured overlay?, *in ACM SIGCOMM Computer Communication Review, January 2004*

33. Skype. P2P based Internet Telephony system. More information at the URL http://www.skype.com/

Acknowledgements

The work that is presented in this paper has been carried out as part of the EC funded P2P ARCHITECT project (IST-2001-32708).