INFORMATION SOCIETIES TECHNOLOGY (IST) PROGRAMME

P2P ARCHITECT

"Ensuring dependability of P2P applications at architectural level"

DELIVERABLE INTERNAL

WORKPACKAGE: WP3 – Models & notations for P2P application architectures

D9 – P2P Reference Architectures (Final Version)

Authors: **L.Melville, J.Walkerdine, I.Sommerville (Lancaster University)**

Submission Date: 12/6/2003

Partners: Athens Technology Center (GR), ENGINEERING Ingegneria Informatica (IT), University of Lancaster (UK), University of Athens (GR), Siceas Services (IT), TOP PROMOTION (GR).

SUMMARY

This document presents a set of Peer-to-Peer (P2P) reference architectures as well as providing some comparisons between them and existing P2P applications. These architectures have largely focused on common co-operative applications, and have been depicted using layered based and component based descriptions.

1. Introduction

1.1. General

This document provides a set of reference architectures for use with Peer-to-Peer (P2P) systems. Due to the requirements of the end user partners involved in the P2P Architect project, the described architectures have predominantly focused on co-operative P2P systems. More specifically, these include general architectures for co-operative P2P applications as well as architectures for Instant Messenger, Shared Workspace, Distributed Search and Document Management styled applications. In addition, due to the popularity of such systems, reference architectures have also been provided for computational P2P systems.

The presented reference architectures also cater for semi-centralised systems, those that possess one or more server nodes, and decentralised systems, those that possess nodes of 'equal' standing. These types of logical network architecture have been discussed in more detail in the 'Report on the dependability properties of P2P architectures' deliverable [2].

As well as these reference architectures, a set of more refined architectures are also presented. These should essentially be regarded as more specific instances of the reference architectures that illustrate one possible way in which they could be expanded.

Towards the end of this document a number of comparisons have been performed between the reference architectures and existing P2P applications. A comparison has also been made with Sun's JXTA [1].

At this point no standard notation had been decided upon for the reference architectures. This will be shortly addressed and the architectures will be updated accordingly.

1.2. Reference Architectures

Within this deliverable, reference architectures are not regarded as merely being templates for the creation of P2P systems. The architectures presented here do not depict how existing system's are designed or how future system's should be designed. Instead, as their name suggests, the architectures should be considered as *points of reference*. This could involve them being used to provide a high level understanding of how such systems can be structured, being used as a basis for architectural comparisons, or to act as possible starting points for the development of the respective types of system.

The reference architectures presented here should not be regarded as *system designs*. Their main objectives are to represent functionality at a very abstract level and to highlight possible structure for this functionality. The reference architectures do not provide an illustration of how a system can be designed, as they are far too abstract for this. For example, the Instant Messaging applications Jabber[9] and ICQ[3] can both be abstractly represented by the semi-centralised Instant Messenger reference architecture presented here. However, their underlying functionality is significantly different. This functionality is captured by the system design, rather than the reference architectures.

Finally, it is important to remember that the reference architectures presented here are abstract generalisations. It is not necessarily the case that all the layers that comprise the architectures will be needed, and likewise, it is expected that others will need to be added. They are abstract architectures because they need to encompass the wide range of different possible system designs that can be developed. As a result they cannot contain details of specific functionality, as these have to be elaborated during the individual system design.

1.3. Overview of the different architectural representations

As has been mentioned this document utilises two different architectural presentations:

- *Layered based reference architectures*, which aim to provide simple overviews of the architectures, identifying at an abstract level the main functionality and structure that would be needed for the specific applications. A layered base notation is commonly used for reference architectures as it allows for the representation of functionality and structure without becoming too specific. The common capabilities (i.e., encryption, Quality of Service (QoS) monitoring, etc) that can be possessed by each layer and by the different application domains are also indicated.

- *Further instantiated architectures*, that expand on the reference architectures by further grouping functionality and identifying potential relationships that may exist between then. These architectures represent examples of how the reference architectures could be further instantiated towards a system design. They themselves, however, are not reference architectures as they are already too specific and it is perfectly reasonable for similar systems to be designed in entirely different ways. The instantiated architectures are not represented using layered notation, but instead functionality is grouped into blocks and possible relationships between these blocks is identified. The notation used is described in more detail in section 3.

The overview nature of layered styled reference architectures means that it can be difficult to clearly identify the differences between the various application types. Consequently the layered architectures presented here do possess a degree of similarity, as layers that capture common capabilities are re-used.

For the more detailed instantiated architectures, however, the differences between the application types that exist become more apparent as specific functionality is further expanded. Obviously, it is not possible for these initial instantiations to be too specific as ultimately it depends heavily on the requirements and functionality for each individual application. Such functionality would be further elaborated during the systems design.

1.4. Decentralised Reference Architectures

The nature of decentralised P2P architectures can pose a number of problems for their usage as the basis of a P2P application. In particular their unpredictable network structure (e.g. nodes being removed/connected, network portioning, etc) makes it almost impossible for them to be fully managed, an issue that could be particularly important for critical business systems.

In addition, this lack of control can also restrict the types of 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 systems, difficult to implement (though in theory, not impossible). Furthermore, large scale decentralised networks will also reduce the usefulness of instant messenger applications (for example, two nodes may be online but not be able to communicate with each other), as well as limiting the effectiveness of search systems.

The use of direct communication decentralised architectures can overcome some of these issues, for example, all nodes on the network will know each other, removing the problem that instant messenger applications 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 D5: Report on the Dependability Properties of P2P Architectures [2], such architectures will not scale and so only making them suitable for small networks.

The decentralised architectures that are presented here are based on an indirect communication model, and consequently they possess mechanisms to publish and discover peers on the network. These architectures can also be adapted for a direct communication approach, by removing these mechanisms.

1.5. List of related documents

1. P2P Architect Technical Annex, submitted 17-SEP-2001
2. Peer-to-Peer: An Overview, Lancaster University, 2001
3. P2P Architect Project: ‘D5 Report on the Dependability Properties of P2P Architectures’
4. P2P Architect Project: ‘D8 Models and specification primitives for building dependable P2P Systems/Applications’
5. P2P Architect Project: ‘D6 Method definition for dependability specification for P2P systems’

1.6. Acronyms and Abbreviations

DMS	Document Management System
IM	Instant Messenger
IP	Internet Protocol
MDA	Model Driven Architectures
P2P	Peer-to-Peer
PIM	Platform Independent Model
QoS	Quality of Service

TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UML	Unified Modelling Language
XML	Extensible Markup Language

2. Layered based Reference Architectures

This section presents a set of layered based reference architectures. For each type of P2P application domain both a semi-centralised and decentralised architecture is described (with the former being broken down into separate server and client architectures). Common layers within each type of architecture are described first and then layers unique to the different node types (for example, client, server and decentralised) are described.

Capabilities that can be possessed by each application domain (for example, Instant Messengers) and within each layer are identified. These represent abstract system functionality and could include, for example, peer advert caching or version control. There does exist a degree of overlap with 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 system (at the network level or at the higher application level). It will be up to the designers to decide where the individual capabilities will be provided within the system.

The abstract functionality for each architecture is given with regards to the application domain it represents. Of course there is no reason why the architectures cannot be combined with functionality from several of the reference architectures, or even the possibility of new functionality being included. However, such actions would result in initial design decisions being made and so would result in the creation of instantiated architectures rather than the more abstract reference architectures. Due to the overview nature of these architecture descriptions there is a degree of repetition across the architectures.

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 used or how they interconnect will be dependent on the specific application. Consequently, although a layer is included in 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.

2.1. Layer Descriptions

This section discusses the various layers that are used in the presented architectures and identifies the main capabilities that can exist at each layer. Common layers are discussed first, followed by layers specific to the client, server or decentralised nodes.

2.1.1 Common Layers

These layers are common to all types of node.

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

The main capabilities of this layer include:

- Providing a software interface to the hardware network controls (Transport and Network layers of the ISO/OSI Network Model [14])
- Utilising the Transmission Control Protocol (TCP) – built on top of Internet Protocol (IP) and adds reliable communication, flow-control, multiplexing and connection oriented communication.
- Utilising the User Datagram Protocol (UDP) – 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 system, and could 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 could include 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 applications would typically fine-tune this layers functionality for their own individual usage.

There is the possibility of other layers, described in this section, to be included within this layer, and they shall be described when appropriate. They have been intentionally removed from this layer for explanatory purposes, as it allows the reader to understand more fully some of the operations of this layer.

The main capabilities of this layer include:

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

Message Resolver Layer – The Message Resolver allows higher-level applications to send and retrieve data. It would typically be capable of packaging any data to be sent into an appropriate format (i.e. using the Extensible Markup Language (XML)) and pass the message packets down to the P2P Network Layer for sending. Incoming messages from the P2P Network Layer would similarly be stripped down and the enclosed data would then be passed up to the higher layers. Some processing of the incoming data may occur here to allow for some intelligent decisions about which layer may require the data. This layer could be encapsulated within the P2P Network Layer

The main capabilities of this layer include:

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

Real Time Connection Monitor - This layer deals with the monitoring of connections in real time, it can help adjust bandwidth requirements and aid in maintaining a high Quality of Service (QoS). This is especially helpful for systems that incorporate video conferencing or streaming content. It could certainly be included within the P2P Network Layer.

The main capabilities of this layer include:

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

Workpackage Manager Layer - A computational based system would typically 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 system, as well as managing the results that are returned.

The main capabilities of this layer include:

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

2.1.2 Server Specific Layers

These are layers that could typically be used within server architectures.

Repository Manager Layer – This layer implements interfaces to any external data sources, it would typically co-operate 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 with the P2P Network Layer it is highly unlikely as it is application specific.

The main capabilities of this layer include:

- To connect to and administer a local or external repositories

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

Check in/out data Layer – Most systems will require concurrent and verified access to any data sources. This layer usually sits alongside the data source (*Repository Manager*), authentication controls and the communication mechanisms involved. Again this is more than likely to be 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:

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

Authentication Layer – Authentication is an extremely desirable attribute for business systems. This layer deals with authenticating peers connected to a P2P network and would likely be able to interconnect 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:

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

Version Control 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 next paragraph); although they could be merged together, they are being kept separate to keep with tradition. It is unlikely for this layer to be included within the P2P Network Layer.

The main capabilities of this layer include:

- To manage multiple versions of data
- To ensure concurrency of data
- To keep track of data versions that may exist on other peers
- To keep a history log of data versions and track changes

Document Management System (DMS) Layer – This layer handles any additional functionality that a document management system might need, that is not provided by the Check in/out data layer. This could be, 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. It is essentially a specialised form of repository manager layer. As mentioned in the previous paragraph the DMS layer could be merged with the version control layer. This layer is a high level one that is geared towards the specific operation of document managements systems and would therefore not be included within the P2P Network Layer.

The main capabilities of this layer include:

- To support transactions with connected repositories, most likely in a secure and concurrent fashion
- To manage multiple versions of data
- To ensure concurrency of data
- To keep track of data versions that may exist on other peers
- To keep a history log of data versions and track changes
- To provide high level document management facilities such as change awareness, co-authoring of documents, etc

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

The main capabilities of this layer include:

- To support and monitor peer awareness within the network
- To support and monitor user awareness within the network
- To support and monitor resource awareness within the network

Data Search/Filtering - This layer represents the search/filtering mechanisms that are used to search the index data that is held within the Data Repository. It can also filter the returned search data corresponding to the search parameters. It is unlikely that this layer would be included within the P2P Network Layer as it is more geared towards the specific application.

The main capabilities of this layer include:

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

2.1.3 Client Specific Layers

These are layers that would typically be used within a Client architecture.

Awareness Controller Layer – This layer would be tightly coupled with the awareness monitor described in the server architectures. The client version, apart from negotiating with the awareness monitor on a server, may also keep records and awareness of 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:

- To support the peers awareness on the network by liasing with server and client peers
- To support the peer's user awareness on the network by liasing with server and client peers
- To support the peer's resource awareness on the network by liasing with server and client peers

2.1.4 Decentralised Specific Layers

Decentralised architectures would generally use the same layers as used in semi-centralised architectures, except in some cases they would need to be altered.

P2P Network Layer - This layer would essentially be identical to the P2P Network Layer used for semi-centralised systems, however it would also need 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:

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

2.2. Layered based Architecture Descriptions

This section provides layered based reference architectures for generic co-operative P2P systems, instant messenger P2P applications, shared workspace P2P applications, search based P2P applications, document management P2P applications and computational P2P systems. It also identifies the main capabilities for each system type.

2.2.1 Generic Co-operative Architectures

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

- Allowing peers to discover and be aware of each other
- Allowing peers to communicate with each other
- Allowing peers to be uniquely identified within the network
- Providing security within the system
- Allowing data to be stored and managed on the peers
- Allowing for the creation of a user interface for the peers
- Allowing for the creation of de-centralised and semi-centralised systems

One of the main roles for the server, within such an architecture, would be to support/administer the communications between peers. In addition security constraints would likely be enforced here, and if required, the server could maintain a data repository. It is the P2P Network Layers role to communicate (possibly in a secure fashion) with other peers. Incoming data is passed up to the Message Resolver so it may un-pack[1] the data and determine what layer it is destined for, once this processing (which is potentially dependent upon the nature of the application) is complete the data is sent to the appropriate location, possibly passing through the Authentication Layer first.

[1] Data may be packaged in XML or some other format

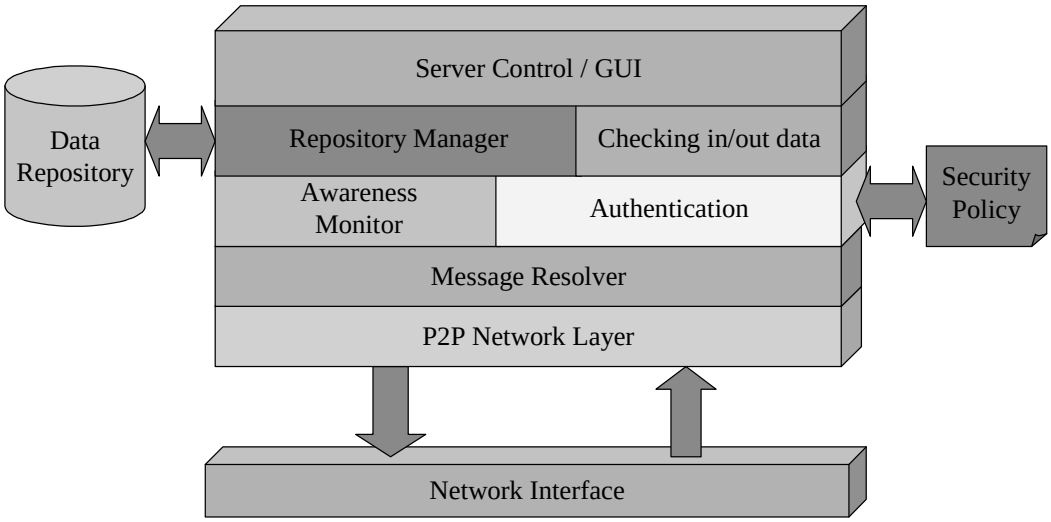


Figure 1 - Generic Co-operative Environment Server Architecture

Client peers would use an Awareness Controller to support updating the server of their status and likewise allowing the server to update them on the status of other peers that they are aware of. Again it is the Message Resolver's role to take incoming data messages, and after some form of processing pass them on to their destination.

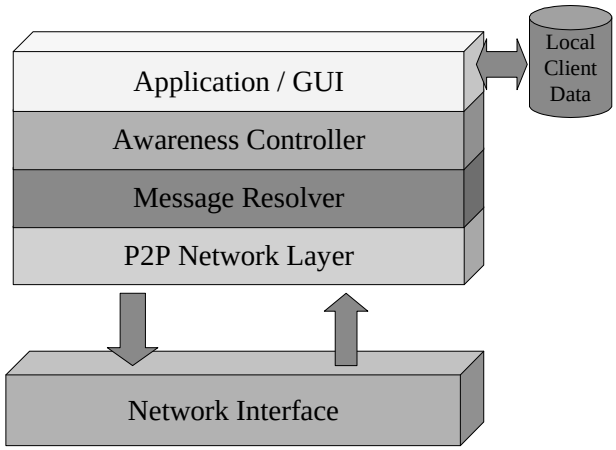


Figure 2 - Generic Co-operative Environment Client Architecture

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 would typically be incorporated into the P2P Network Layer, and would normally involve the creation and publication of adverts onto the network. In addition, some form of discovery mechanism would be required that could 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 would also need to be provided 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 would typically be used to store such messages.

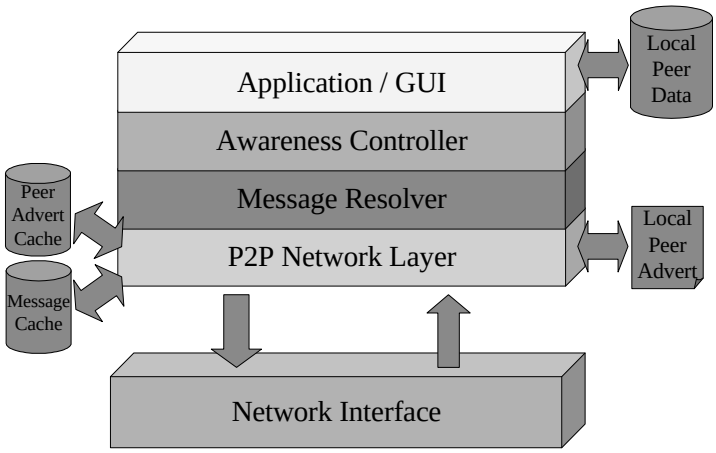


Figure 3 – Generic Co-operative Environment Decentralised Architecture

2.2.2 Instant Messenger Architectures

Instant messenger applications such as ICQ [3] and MSN Messenger [4], allow users to communicate directly with each other and in an instantaneous fashion. The main capabilities of such systems include:

- Allowing peers and users to discover and be aware of each other
- Allowing peers and users to communicate with each other
- Allowing peers and users to be uniquely identified on the network
- Providing security within the system
- Allowing data to be stored and managed on the peers
- Allowing for the creation of an Instant Messenger user interface for the peers
- Allowing for the creation of de-centralised and semi-centralised systems

Instant messenger applications have a high dependency upon awareness monitoring. It is possible to have awareness within the P2P Network Layer but for this form of application it would be more suitable for it to remain at a higher level so it is more visible to developers. Incoming requests, once they have been processed at the Message Resolver layer, are passed up through the Authentication layer, to the layer that will eventually deal with the request. If necessary messages could also pass through the Awareness Monitor. This would enable the server to update the status of the peer that is doing the request (i.e. the requesting peer may have been offline, by pumping the message through the Awareness Monitor enables the server to rectify the status of the peer).

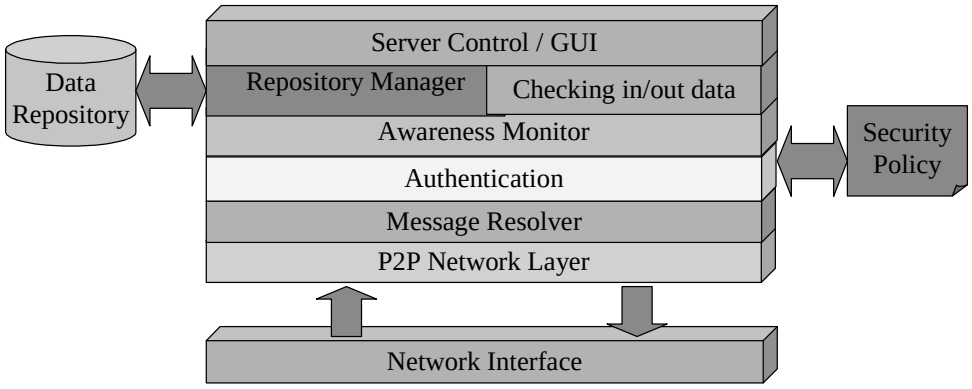


Figure 4 - Instant Messenger Server Architecture

The client version's Awareness Controller is likely to be tightly coupled with the Awareness monitor running on a server. For instance, if a peer is already known to a particular peer (i.e. it does not need to contact the server for location information as it had already done so in the recent past[2]), and tries to send a message to that peer but is unable to make contact, it could assume that the peer had gone offline and send a notification to the server. The server could then act upon this information accordingly.

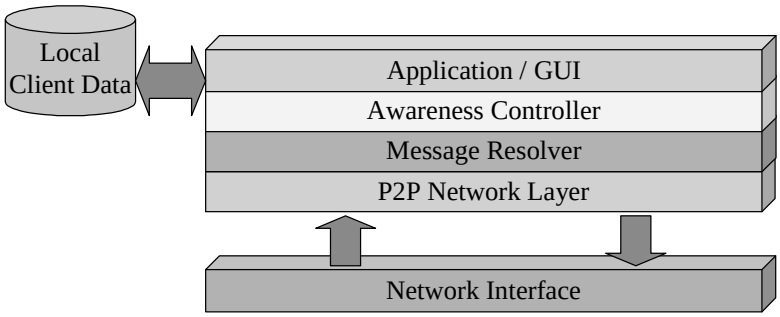


Figure 5 - Instant Messenger Client Architecture

Large scale decentralised instant messenger applications are unlikely to be feasible due to the restrictions posed by the network (i.e. not all peers being able to connect to each other, portioning, QoS issues, etc). However, for smaller scale (and especially direct communication architectures) it may be possible to support such a system. A decentralised node in such an architecture would 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.

[2] Meaning minutes, not hours or days.

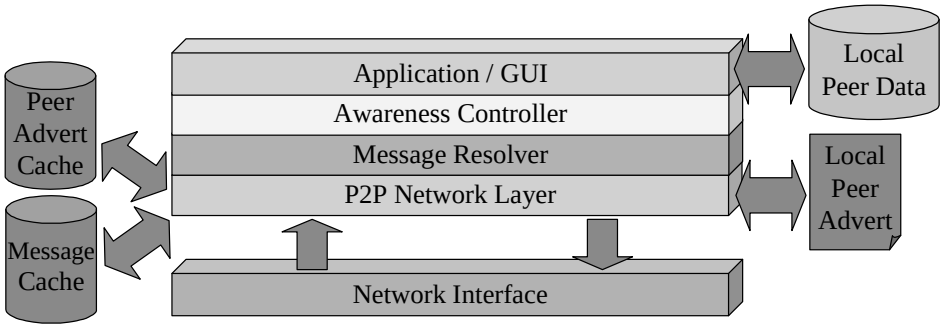


Figure 6 – Instant Messenger Decentralised Architecture

2.2.3 Shared Workspace Architectures

Shared Workspace systems can include those that support chat rooms, shared whiteboards, video conferencing, etc. The main capabilities of such systems include:

- Allowing peers and users to discover and be aware of each other
- Allowing peers and users to communicate with each other
- Allowing peers and users to be uniquely identified on the network
- Ensuring a high QoS for communications
- Providing security within the system
- Allowing data to be stored and managed on the peers
- Allowing for the creation of a shared workspace user interface for the peers
- Allowing for the creation of de-centralised and semi-centralised systems

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 stripped down by the Message Resolver, it may make a decision to pass the data/request through the Real Time Connection Monitor layer in case their are time constraints attached to this kind of data/request (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 could then make a decision to attach a high priority to the data/request.

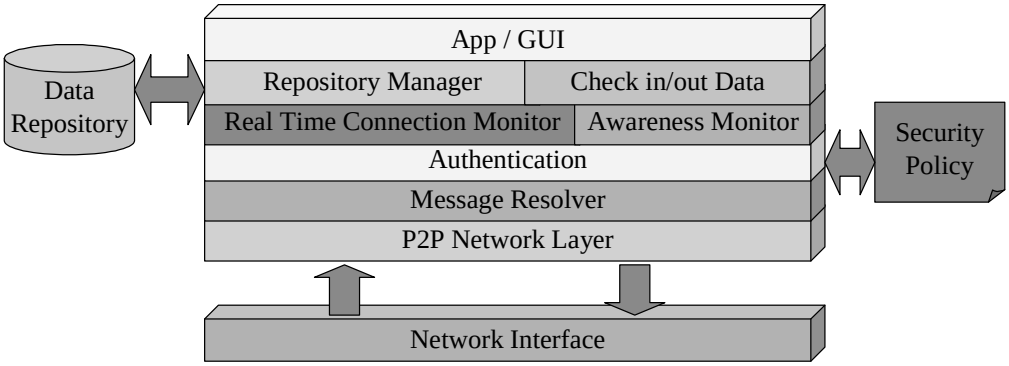


Figure 7 - Shared Workspace Server Architecture

As with the server version it is the Real Time Connection Monitor that is liable to be the crucial component for the client. Additionally, more control is available at the Awareness Controller by having the layer higher than the P2P Network Layer.

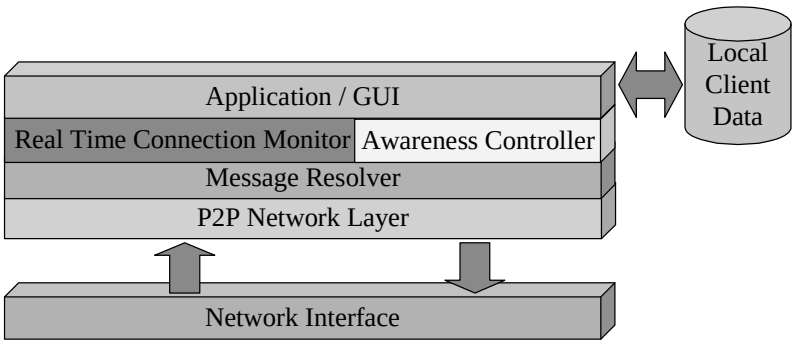


Figure 8 - Shared Workspace Client Architecture

Achieving a truly decentralised shared workspace system is particularly difficult due to the fact that the workspace itself is going to need 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 would be similar to that for a server node (described above), but also including the requisite mechanisms required for decentralised systems (e.g. publication, discovery and message routing).

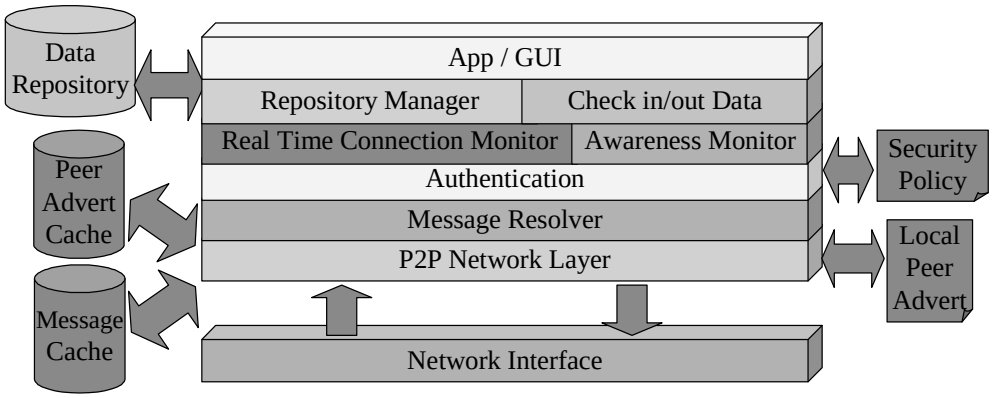


Figure 9 – Shared Workspace Decentralised Architecture

2.2.4 Search System Architecture

Search systems allow users/peers to search for data that is distributed around the P2P network. The main capabilities of such systems include:

- Allowing peers to discover and be aware of each other
- Allowing peers to discover and be aware of resources (for example, data) that may exist on the network
- Allowing peers to communicate with each other, including allowing the transference of data
- Allowing peers to search the network for resources, potentially using a variety of filtering techniques
- Allowing peers and resources to be uniquely identified on the network
- Providing security within the system
- Allowing data to be stored and managed on the peers
- Allowing for the creation of a search system user interface for the peers
- Allowing for the creation of de-centralised and semi-centralised systems

The server version of this architecture would typically utilise mechanisms within a layer for searching and filtering data indexed within a data repository. When a search request comes in and is processed in the usual way by the Message Resolver, it would typically pass through the Authentication layer on its way to the Data Search/Filtering layer. A detailed search could be carried out by this layer by interrogating the Repository Manager, returned results could be checked against current online peers and any returned results where the peer is offline or is overloaded could be filtered out. The search results, passing through the authentication layer once again, can carry on out to the requesting peer.

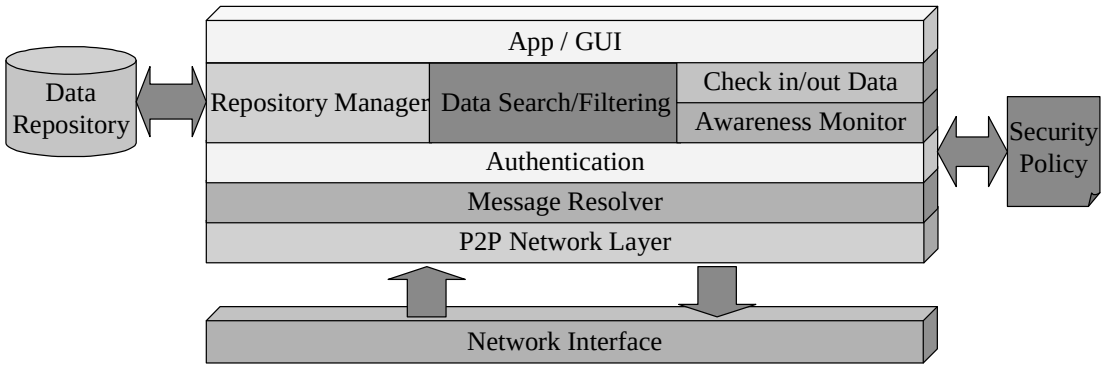


Figure 10 - Search System Server Architecture

The Awareness Controller shown in the client architecture is liable to be tightly coupled to the Awareness Monitor on the server version. These two layers would allow a client to keep track of other peers on the network so when a search response returns from the server the client would be able to contact the peer hosting the required data. Any authentication details regarding the peer hosting the required data/service could be attached to the search response as it passes through the server's authentication layer, on its way back to the requesting peer.

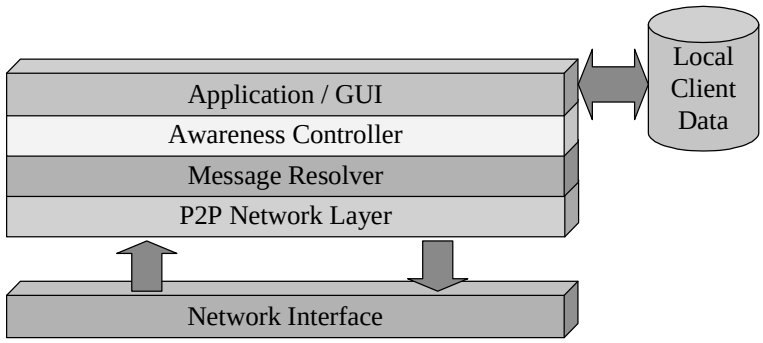


Figure 11 - Search System Client Architecture

Decentralised search systems are already quite common (Gnutella [7], Freenet [6], etc). A node within a decentralised system would possess the same functionality as a client node described above, but would also need to handle the data searching aspect as well. When that node receives a search request then it would need to perform a search on the data that is held locally. In addition the node would also need to handle the decentralisation specific issues (e.g., publishing, discovery and message routing).

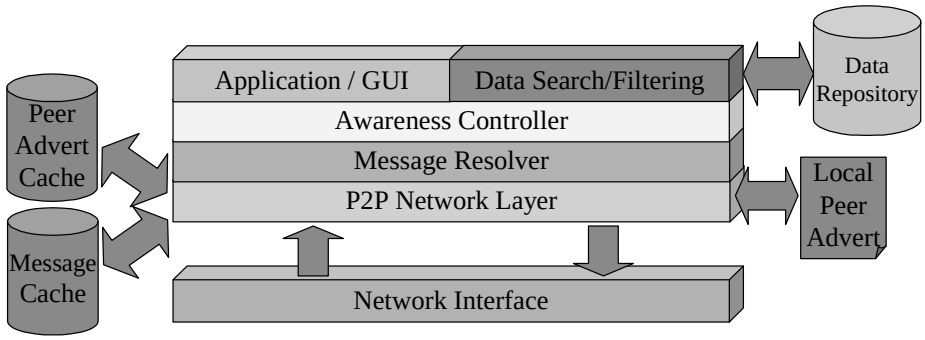


Figure 12 – Search System Decentralised Architecture

2.2.5 Document Management Architectures

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

- Allowing peers and users to discover and be aware of each other
- Allowing peers and users to discover and be aware of documents that may exist on the network
- Allowing peers and users to communicate with each other, including allowing the transference of documents
- Allowing peers, users and documents to be uniquely identified on the network
- Supporting the versioning of documents, change tracking, concurrency and various other document management facilities
- Providing security within the system
- Allowing data to be stored and managed on the peers
- Allowing for the creation of a document management user interface for the peers
- Allowing for the creation of de-centralised and semi-centralised systems

The server architecture for a document management system would involve the inclusion of a facility for version control and document management; these are represented as the DMS layer and Version Control layer. The Version Control layer would orchestrate between the DMS layer and both the Check in/out Data and Awareness Monitor layers. The Authentication layer, aside from authenticating peers on the network, could also authenticate access to the documents.

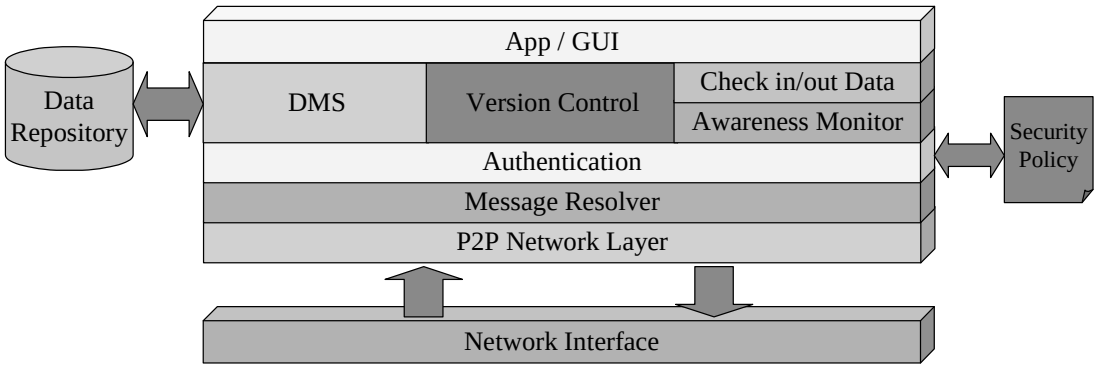


Figure 13 - Document Management Server Architecture

As with most other client architectures, the Awareness Controller is liable to be one of the crucial components for this kind of system. It would maintain contact with both the server and other peers in an attempt to keep consistency of sharable documents.

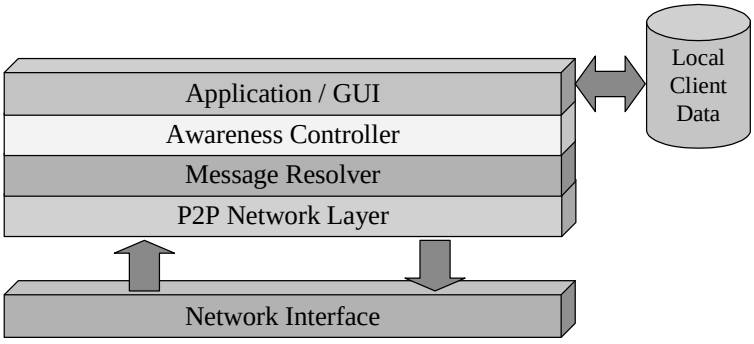


Figure 14 - Document Management Client Architecture

As with shared workspace systems, it is likely to be difficult to make a dependable decentralised document management system. With such a system keeping track of documents within the network would 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 system to succeed it will be necessary to resolve such issues.

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).

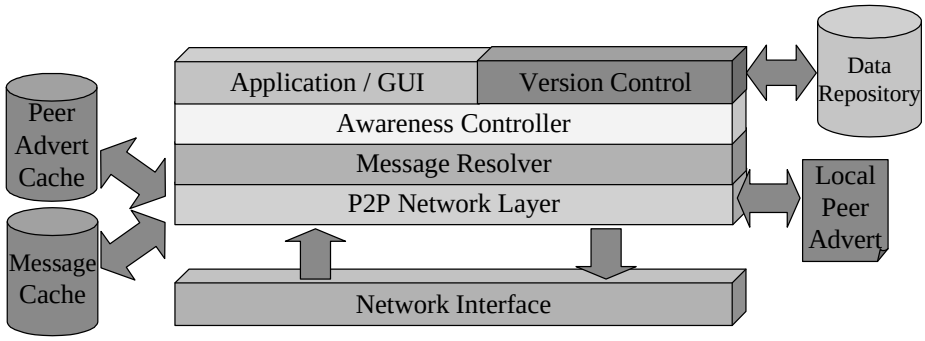


Figure 15 – Document Management Decentralised Architecture

2.2.6 Computation System Architectures

Computational systems rely heavily on one or more server peers distributing work (work packages) to a network of peers and collating the returned results. The client-server relationship of such systems is often so strong that computational systems (such as SETI@home) are often considered to be not true P2P systems.

The main capabilities of such systems include:

- Allowing peers to discover and be aware of each other
- Allowing peers to communicate with each other
- Allowing peers to be uniquely identified on the network
- Allowing for the breakdown of work into work packages that can then be distributed throughout the network for processing
- Keeping track of work packages within the system (who is processing them, etc)
- Allowing for the collation of results, error checking and redundancy
- Providing security within the system
- Allowing data to be stored, managed and processed on the peers
- Allowing for the creation of a computational system user interface for the peers (if needed)
- Allowing for the creation of de-centralised and semi-centralised systems

As indicated in D5: Report on the Dependability Properties of P2P Architectures [2], it can also be possible for the client peers to posses autonomy and to be able to communicate directly with each other. This, for example, could allow client peers to communicate results to each other, or possibly to work together as a cluster. The architectures presented here represent those computation systems where the client peers do possess autonomy.

Obviously one of the most important aspects of computational systems is that the actual computation that is carried out is correct and not compromised. As a result the best policy to adopt within P2P systems would be 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).

The architecture of a server peer would operate in a very similar manner to the document management system. Rather than managing documents the server would instead manage work packages. This would include 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 would be beneficial and so results returned by client peers need to be authenticated and compared.

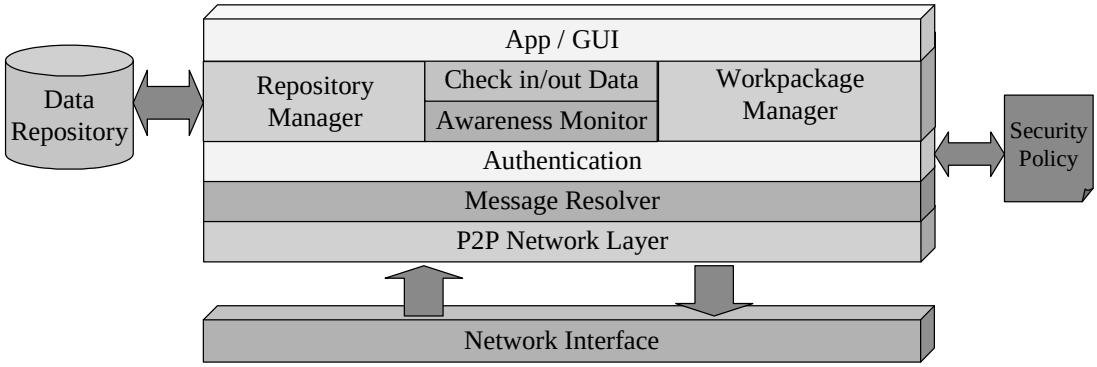


Figure 16 – Computational System Server Architecture

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

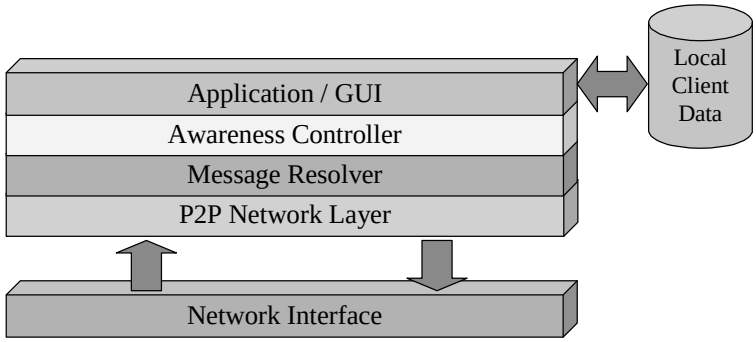


Figure 17 – Computational System Client Architecture

Decentralised computational systems can 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, unpredictable nature of the network and the need to assign peers unique ID's may hinder this process, and so such a system would need to be carefully designed. Decentralised systems are likely to make much

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 would also need to handle the decentralisation specific issues (e.g., publishing, discovery and message routing).

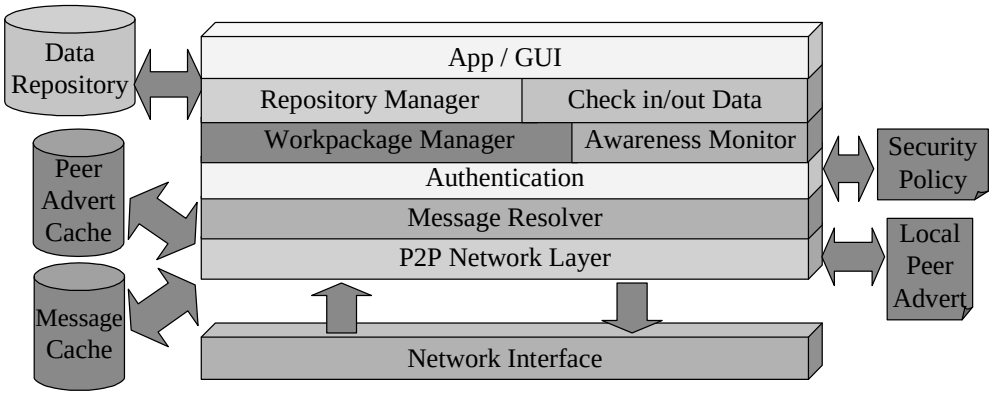


Figure 18 – Computation System Decentralised Architecture

3. Further Instantiated Architectures

This section provides a set of more detailed *instantiated* architectures. These architectures represent a possible way in which the reference architectures could be further expanded, and portray the systems in terms of potential functionality groupings and the relationships that might exist between them. It should be noted, however, that despite this these groupings and relationships are still very abstract. Specific functionality and how it is provided will ultimately be down to the requirements and the further system design.

Initially a generic reference architecture is provided, and this forms the basis for the more application specific reference architectures that follow. As with the layered reference architectures, no standard method of notation was decided upon. Consequently, the notations used should be interpreted to mean the following:

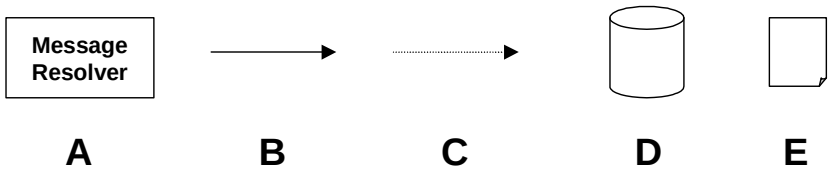


Figure 19 - Notation used in the instantiated architectures

A – Groupings of functionality. These are not necessarily specific, and in some cases may not be required at all. However they do indicate aspects of the application that the designers may want to consider.

B – A relationship exists between the two functionality groupings. In some cases this relationship may be just one way, in others, it can be two way.

C – A flow of data between functionality groupings. As with B this can be one way or two way

D – A repository of some form.

E – Data of some form.

The following sections present each of the system reference architectures in turn. A description is provided of the main functionality groupings involved.

3.1. Generic Co-operative Environment Architectures

3.1.1 Server Architecture and Grouping Descriptions

The architecture for the server peers has been split into two modules, an *Application/Data Module* and a *Communication Module*.

The Application/Data Module represents the higher-level parts of the server peer that could include the server application, authentication, and database management.

The Communication Module deals with the network aspects of the peers. This can include the sending and receiving of messages onto the peer network, and also the interpretation of these messages.

Clearly in reality it is likely to be very difficult to create such definite splits, and it is probable that certain functionality groupings of this architecture will fall into both modules (for example, the Operation Resolver).

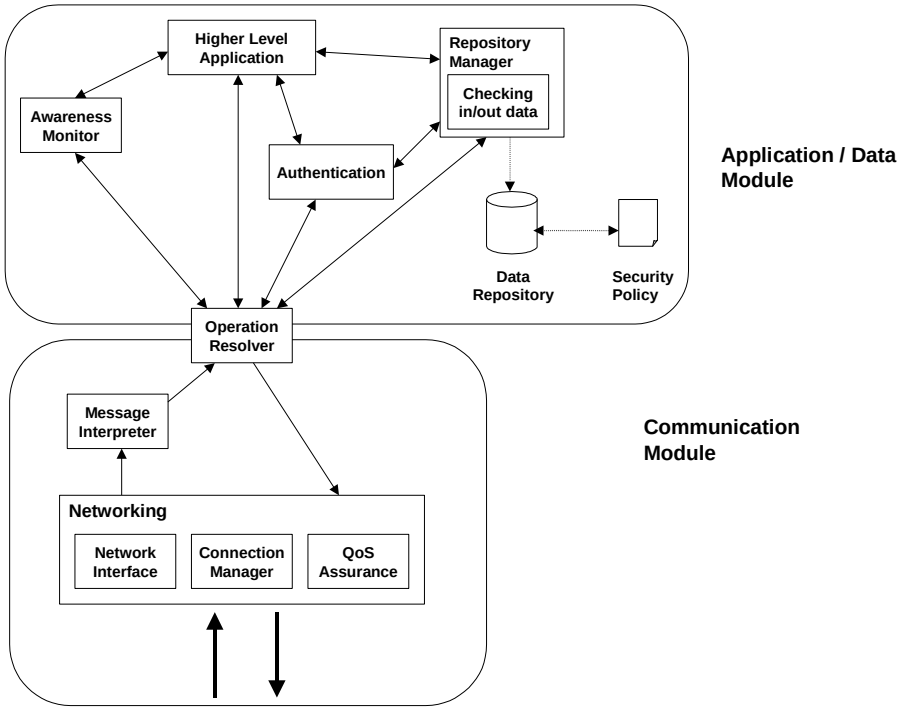


Figure 20 - Generic Co-operative Environment Architecture for Server Peers

Network Interface – this represents the peer's low-level connection with the P2P network. Essentially it would deal with the sending and receiving of data. This could also be part of a protocol such as JXTA.

Connection manager – this functionality grouping deals with the establishing of connections between peers on the logical network (not the physical). This could be part of a protocol such as JXTA. This grouping may also be strongly linked to the QoS Assurance functionality grouping.

QoS Assurance – this functionality grouping seeks to assure quality of service within a connection between two peers. In some applications, this grouping may also be monitored from within the application module (for example, in situations where QoS is of paramount importance). This grouping could be part of a protocol such as JXTA.

Message Interpreter – this functionality grouping essentially takes the received data and tries to construct a coherent message from it. This message can then be passed up to the operation resolver where it can be processed. This grouping could be part of a protocol such as JXTA.

Operation Resolver – this functionality grouping deals with the processing of messages and is likely to operate within both the Application and Communication module. Essentially it illustrates the operations that the system should cater for. Some operations may be specifically for higher-level parts of the application; some may be lower level and deal with networking issues (e.g., pinging). The operation resolver also deals with the construction of messages that may be sent from the peer.

Authentication – it is likely that in a co-operative environment authentication will be important. This functionality grouping deals with authenticating those peers/users who request data, or make changes. It works in tangent with specified security information that could be stored in a database, or even be apart of the data (e.g. access rights for a document).

Repository Manager – this functionality grouping essentially manages the storage of the data that is utilised by the application. In reality it is likely that the Checking in/out data and the Repository Manager functionality groupings will be very closely linked, and possibly be parts of a single grouping. The Repository Manager will also have strong ties with the Authentication functionality grouping. An example Repository Manager could be a database management system.

Checking in/out data – most co-operative systems will involve the storing and exchanging of shared data. This functionality grouping manages these data transactions, and can be used to ensure concurrency and to ensure the security of the data.

Awareness Monitor – this functionality grouping monitors the state of the other peers/users on the network. Although how and what this grouping achieves is likely to be very application specific, awareness (or presence) is likely to be a fundamental functionality grouping of most co-operative P2P applications. Depending on the level of awareness (user, peer, etc), this component may also be part of the Communication module.

Higher Level Application – this functionality grouping represents the main application that is running on the server. This will vary according to the actual application, but this grouping could deal with issues such as user interface, file handling, etc.

3.1.2 Client Architecture and Grouping Descriptions

As with the server architecture, the client architecture is split into two modules, a *Communication Module* and an *Application Module*. The communication module is identical to the one presented in the server architecture, and so descriptions of this modules functionality groupings can be found in the previous section. The application module represents the higher-level aspects of the client peer and can include the actual application and mechanisms for promoting awareness.

As with the server architecture, it is unlikely that such a clear split will be apparent in reality.

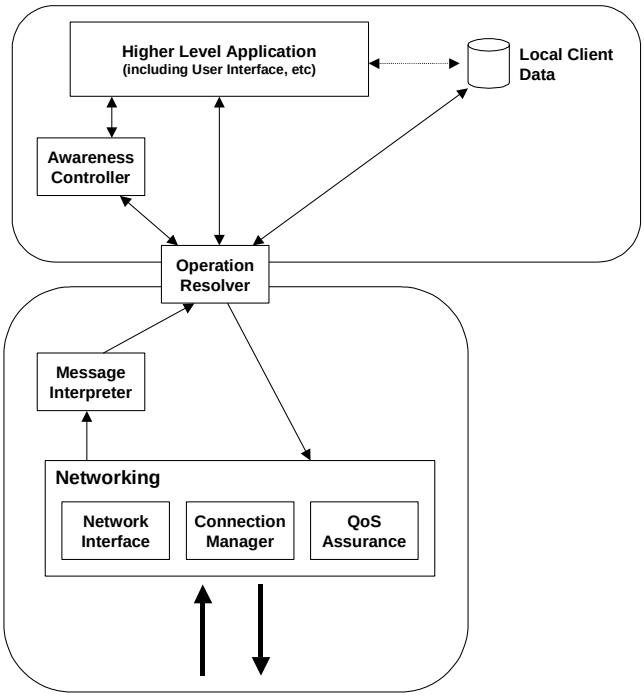


Figure 21 - Generic Co-operative Environment Architecture for Client Peers

Awareness Controller – this functionality grouping deals with the gathering, control, and distribution of awareness information. As awareness is likely to be significant in most co-operative P2P applications, it has been identified as a separate grouping here. As with the Awareness Monitor, this grouping may also be part of the Communication module.

Higher Level Application – this functionality grouping represents the main application that is running on the client. This will vary according to the actual application, but this grouping could deal with issues such as user interface, file handling, etc.

3.1.3 Decentralised Architecture and Group Descriptions

As with the client-server architectures presented above, the instantiated architecture for a decentralised node is also split into *communication* and *application* modules. The main difference, however, is that a decentralised node also needs to handle the publication of itself, the discovery of other peers, and the routing of messages on the network. This functionality would typically be incorporated within the communication module aspect. In addition, depending on the application type, it may be necessary to incorporate additional functionality (that which would typically be carried out by a server node) into the application module.

Existing functionality groupings may also need to change slightly in the way in which they operate. Given the varied and unpredictable nature of decentralised systems, issues such as QoS assurance and the monitoring of awareness information, may need to be tackled in different ways or their objectives re-examined.

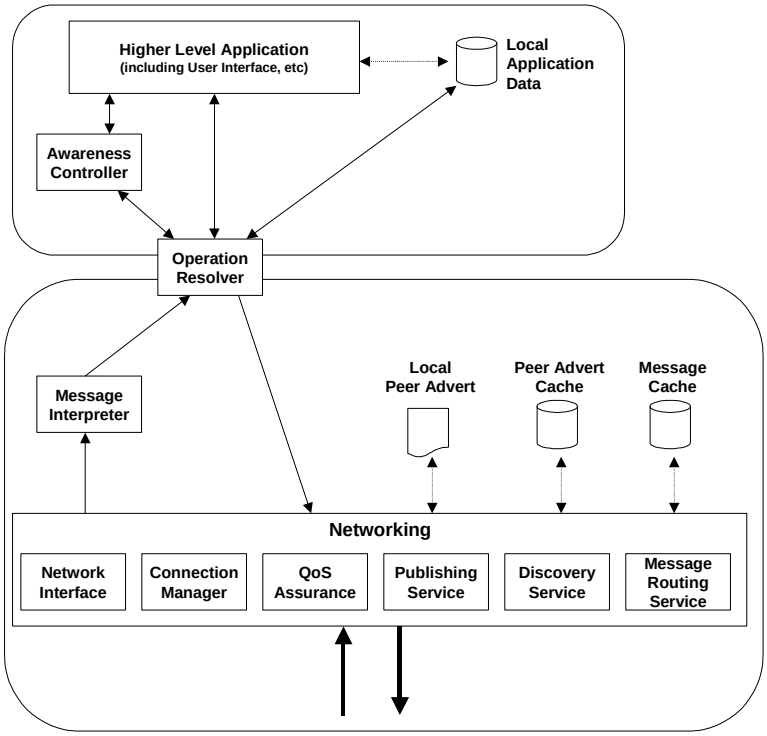


Figure 22 - Generic Co-operative Environment Architecture for Decentralised Peers

Publishing Service – this functionality grouping deals with the publishing of the peer or peer services on to the network. Typically this involves the use of adverts that encapsulate relevant information. These adverts are normally broadcast to any other peers the local peer is aware of (i.e. ones that have been discovered).

Discovery Service – this functionality grouping deals with the discovery of other peers and peer services that exist on the network. A typical approach is to broadcast a discovery request that is propagated around the network. Any peers that receive the request and meet the criteria would then respond with the relevant adverts. Discovered adverts would typically be stored within a cache for later re-use.

Message Routing Service – this functionality grouping deals with the routing of messages the peer has received that are not addressed to itself. Typically this involves the peer forwarding the message to any peers it is currently aware of (based on the peer adverts that are stored in the peer advert cache). In order to stop loops where the same message is forwarded again and again, messages are normally temporarily stored. If the peer should receive a message it has already forwarded on once before, then it will drop it.

Local Peer Advert – this represents the advert(s) for this peer. These are typically created when the peer is first instantiated. As well as advertising the actual peer, adverts can be used to publicise any services or resources that the peer provides.

Peer Advert Cache – a cache would typically be used to store any adverts that have been discovered on the network. This allows them to be re-used for communication/publishing purposes. Adverts within a cache usually possess a lifetime, after which they are no longer valid.

Message Cache – messages that the peer has received would normally be temporarily stored in some form of cache. These messages would be deleted after a short period (normally the lifetime of that message).

3.2. Instant Messenger Reference Architectures

The instantiated architectures for the instant messenger system extend the generic architectures presented in section 3.1. Consequently replicated functionality groupings have not been re-described.

3.2.1 Server Architecture and Group descriptions

The instant messenger server architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- User Details
- Messages to be routed

The architecture is presented in Figure 23.

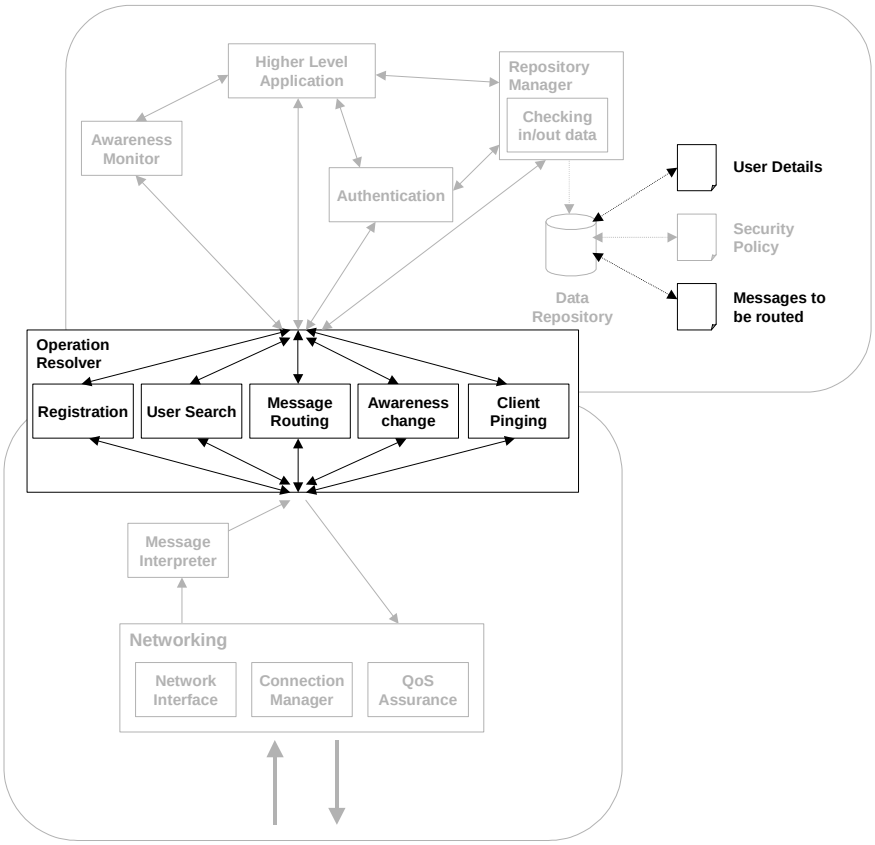


Figure 23 - Instant Messenger Architecture for Server Peers

Operation resolver – for an instant messenger application a number of operations need to be catered for by the server. These include:

- *User Registration* – allowing a user to register themselves with the P2P system. This should not only occur when a new user connects, but also whenever a user connects, so that the server always hold up to date information about the users of the system.

- *User Search* – allowing a user to search for other users of the P2P system. This is particularly important, as it is the main way for users to 'discover' other users of the system.

- *Message Routing* – allowing messages to be routed via the server, rather than directly between client peers. It is necessary to support this so users can still send messages even if the target user is not currently connected to the network.

- *Awareness Change* – allowing a user or client peer to be able to inform the server (and thus other relevant client peers), that there has been a change in state (e.g. from 'Free' to 'Busy').

- *Client Pinging* – allowing the server to check on the status of the peers/users on the network. This is necessary should peers/users disconnect from the network without being able to inform the server.

User Details – details about all users of the application.

Messages to be routed – messages that a user has requested to be routed via the server.

3.2.2 Client Architecture and Group descriptions

The instant messenger client architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- User Details
- Contact List Data
- User Interface Aspects

The architecture is presented in Figure 24.

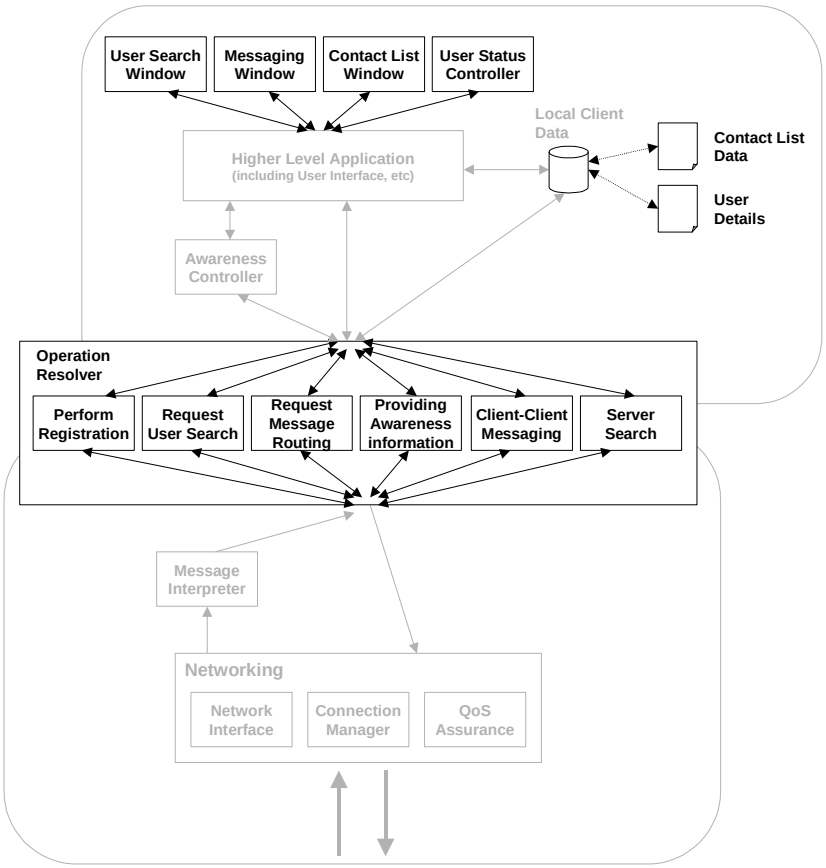


Figure 24 - Instant Messenger Architecture for Client Peers

Operation resolver – the client within an Instant Messenger application has to deal with a number of operations. These include:

- *User Registration* - the client peer needs to support user registration with the server. This should be done not only when a new user connects to the system, but also every time a user connects.
- *Request User Search* – the client peer needs to allow users to search the server for other users.
- *Request Message Routing* – the client peer should provide the user with the possibility of routing messages via the server, should the target peer not be currently online.
- *Providing Awareness Information* – the client peer should allow the user to specify and publish their awareness information to the server. Some aspects of this can also be automated (for example, if the user shuts down the client, then the server could be automatically informed of this).
- *Client-Client Messaging* – the client needs to support the exchanging of messages/data between other clients. This is the heart of the Instant Messenger application.
- *Server Searching* – the client needs to be able to locate a server upon start-up.

User Interface components – the client would need to provide various user interfaces to the various operations it can perform. For example, a contact list window, a messaging window, etc

Contact List Data – details of who the user is interested in (on their contact list)

User Details – details of the user. This can include information that is also made publicly available to the rest of the system.

3.2.3 Decentralised Architecture and Group descriptions

The instant messenger decentralised architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- User Details
- Contact List Data
- User Interface Aspects

Many of these functionality groupings are similar to those presented in the client architecture (section 3.2.2). The architecture is presented in Figure 25.

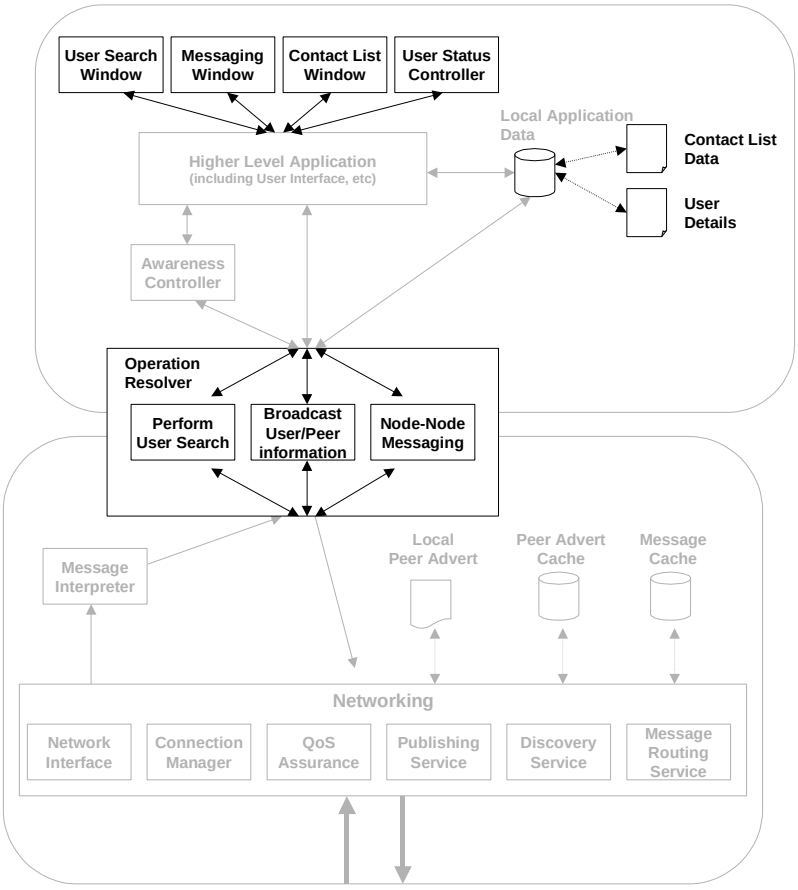


Figure 25 – Instant Messenger Architecture for Decentralised Peers

Operation resolver – the peers within a decentralised Instant Messenger application have to deal with a number of operations. These include:

- *Perform User Search* – the peer needs to allow users to search the network for other users. Similarly the peer needs to respond when it receives a user search request.

- *Broadcast User/Peer Information* – the peer needs to be able to publish its user/peer information to the rest of the network. Such broadcasting should occur whenever the status of the user/peer changes. Some aspects of this can also be automated (for example, if the user shuts down the application, then this fact can be broadcast to the rest of the network).

- *Node-Node Messaging* – the peer needs to support the exchanging of messages/data between other peers. This is the heart of the Instant Messenger application.

User Interface components – the peer would need to provide various user interfaces to the various operations it can perform. For example, a contact list window, a messaging window, etc

Contact List Data – details of who the user is interested in (on their contact list)

User Details – details of the user. This can include information that is also made publicly available to the rest of the system.

3.3. Shared workspace Reference Architectures

The instantiated architectures for a shared workspace system extend the generic architectures presented in section 3.1. Consequently replicated functionality groupings have not been re-described.

3.3.1 Server Architecture and Group descriptions

The shared workspace server architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- Workspace Details
- Workspace Creation
- Real-time Connection Monitor

The architecture is presented in Figure 26.

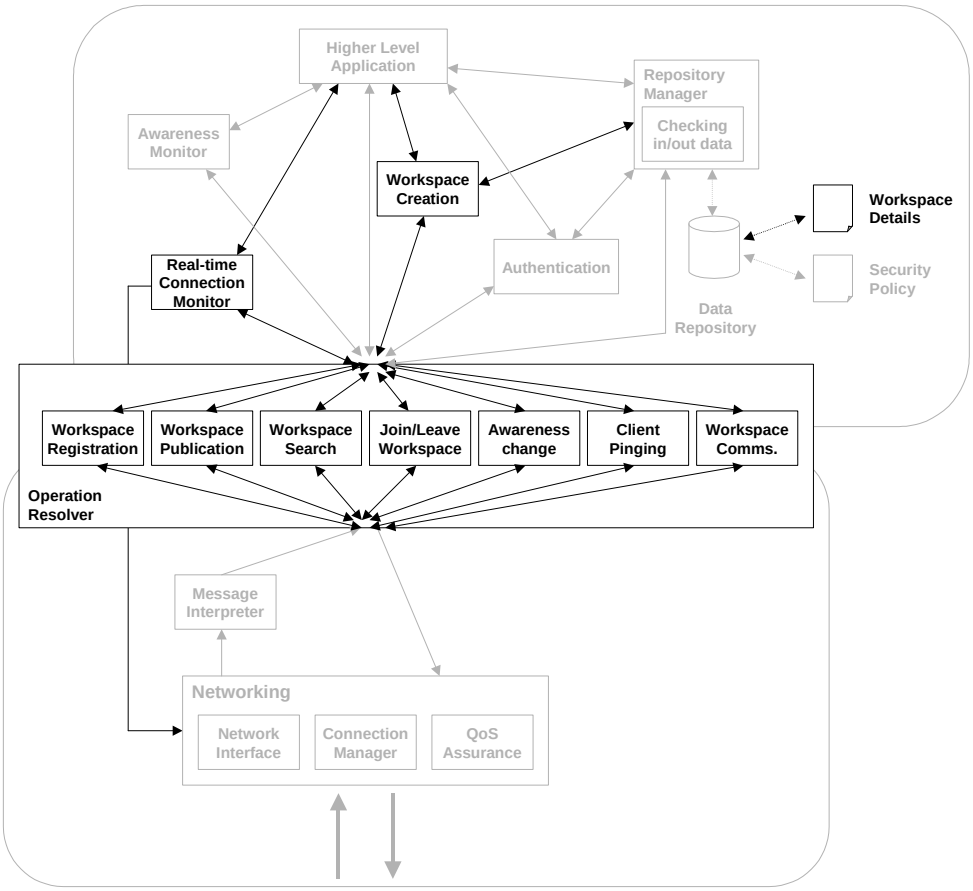


Figure 26 - Shared Workspace Architecture for Server Peers

Operation resolver – For a shared workspace architecture the server needs to cater for a number of operations. These include:

- *Workspace Registration* – allowing a user or client peer to create and register a workspace. These workspaces are created and stored on the server.

- *Workspace Publication* – informing all client peers/users within the system of what workspaces exist, and when a new one has been created. This could also involve the 'inviting' of users into a workspace.

- *Workspace Search* – allowing a user or client peer to search the server for other workspaces that may exist.

- *Join/Leave Workspace* – to allow a user or client peer to join or to leave a workspace.

- *Awareness Change* - allowing a user or client peer to be able to inform the server (and thus other relevant client peers), that there has been a change in state (e.g. from 'Free' to 'Busy').

- *Client Pinging* - allowing the server to check on the status of the peers/users on the network. This is necessary should peers/users disconnect from the network without being able to inform the server.

- *Workspace Communication* – ensuring that all communication directed at a specific workspace is passed on to all interested parties. For example, if an item were added to a shared workspace then interested parties would be informed of this. Communication between users could also be routed via the server, although it would probably be more efficient to support direct communication between client peers.

Real-time Connection Monitor – depending on the nature of the workspace ensuring a high QoS for the system may be paramount. This is particularly the case for workspaces that might utilise streaming audio or video. It is likely that the QoS provided by the system will also be monitored by the higher-level application.

Workspace Creation – this functionality grouping allows for the creation of shared workspaces. The repository manager stores workspaces that have been created.

Workspace Details – details about all the workspaces that exist within the application. This can include information about what exists within the workspace, and details of the users who have access to it.

3.3.2 Client Architecture and Group descriptions

The shared workspace client architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- Workspace List Details
- User Details
- User Interface Aspects
- Real-time Connection Monitor

The architecture is presented in Figure 27.

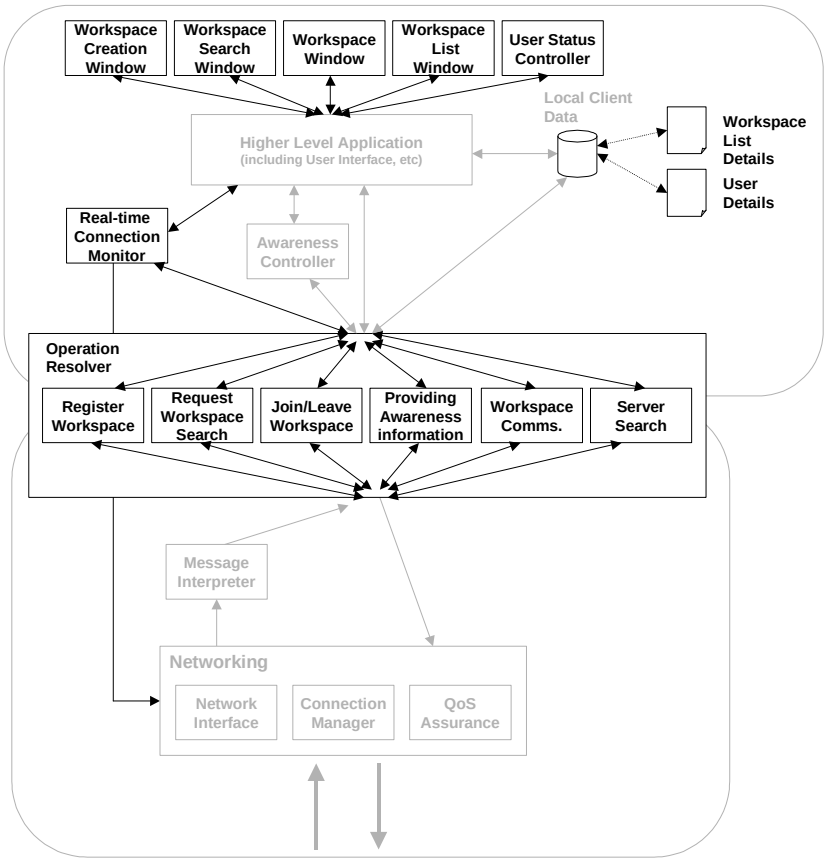


Figure 27 - Shared Workspace Architecture for Client Peers

Operation resolver - the client within a Shared Workspace application has to deal with a number of operations. These include:

- *Register Workspace* – the client peer needs to support workspace registration with the server. This should be done whenever a user wishes to create a new workspace.

- *Request Workspace Search* – the client peer needs to allow users to search the server for other workspaces.

- *Join/Leave Workspace* – the client peer needs to allow users to join and leave workspaces. This will of course be dependent on whether or not the user has the requisite permissions.

- *Providing Awareness Information* – the client peer should allow the user to specify and publish their awareness information to the server (and also to any relevant shared workspaces). Some aspects of this can also be automated (for example, if the user shuts down the client, then the server could be automatically informed of this).

- *Workspace Communication* - the client needs to support the exchanging of messages/data with the shared workspace, and with other clients if necessary.

- *Server Search* - the client needs to be able to locate a server upon start-up.

Real-time Connection Monitor – depending on the nature of the workspace ensuring a high QoS for the system may be paramount. This is particularly the case for workspaces that might utilise streaming audio or video. It is likely that the QoS provided by the system will also be monitored by the higher-level application.

User Interface components – the client would need to provide various user interfaces to the various operations it can perform. For example, a workspace list window, a workspace window, etc

Workspace List Data – details of which workspaces the user is interested in

User Details – details of the user. This can include information that is also made publicly available to the rest of the system.

3.3.3 Decentralised Architecture and Group descriptions

The instant messenger decentralised architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- Workspace Details
- Workspace Creation
- Real-time Connection Monitor
- Workspace List Details
- User Details
- User Interface Aspects
- Repository Manager, Checking in/out data and Authentication

Many of these functionality groupings are similar to those presented in the client and server architectures (sections 3.3.1 and 3.3.2 respectively). To avoid repetition, in some cases a reference is provided back to the section where they have been previously described. The architecture is presented in Figure 28.

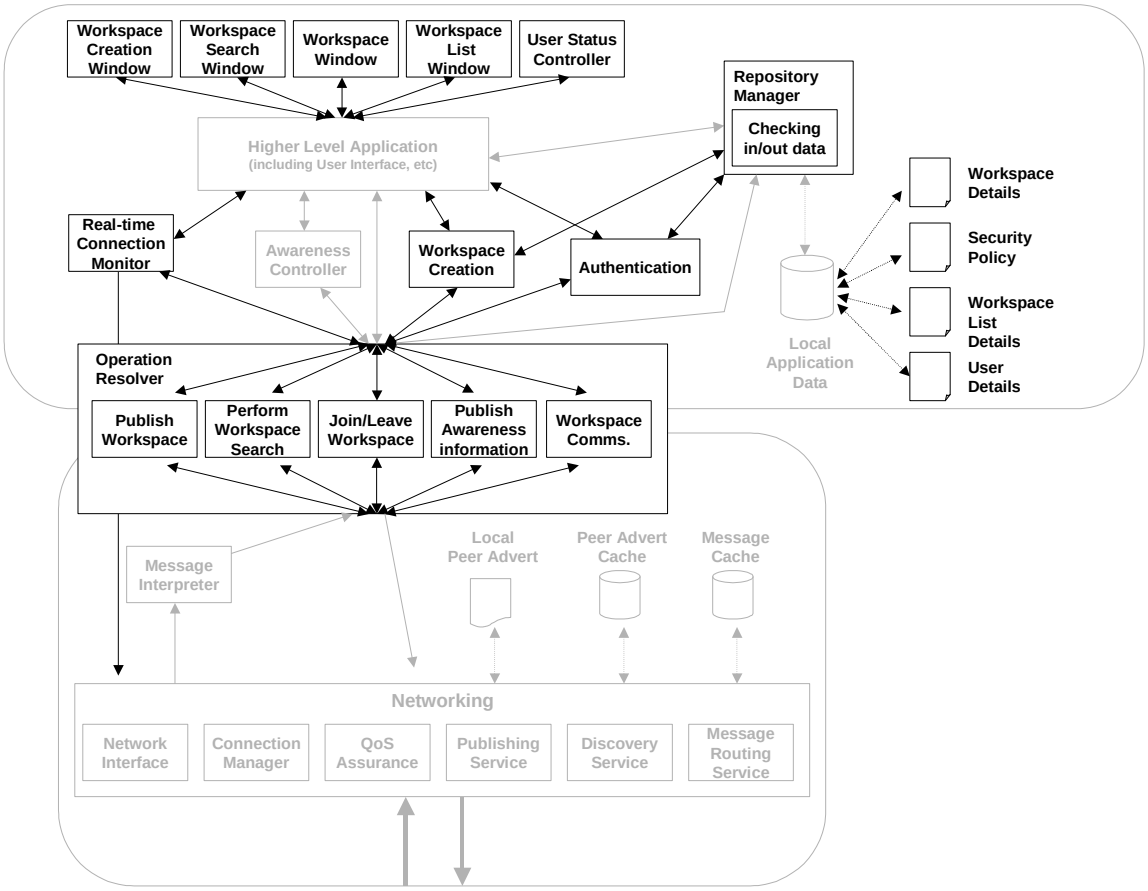


Figure 28 - Shared Workspace Architecture for Decentralised Peers

Operation resolver - peers within a decentralised Shared Workspace application have to deal with a number of operations. These include:

- *Publish Workspace* – the peer needs to be able to publish information about any workspaces it has created, to the rest of the network.

- *Perform Workspace Search* – the peer needs to allow users to search the network for other workspaces that have been created and published.

- *Join/Leave Workspace* – the peer needs to allow users to join and leave workspaces. This will of course be dependent on whether or not the user has the requisite permissions.

- *Publish Awareness Information* – the peer should allow the user to specify and publish their awareness information onto the network. Such broadcasting should occur whenever the status of the user/peer changes. Some aspects of this can also be automated (for example, if the user shuts down the application, then this fact can be broadcast to the rest of the network).

- *Workspace Communication* - the peer needs to support the exchanging of messages/data with the shared workspace, and with other peers if necessary.

Real-time Connection Monitor – depending on the nature of the workspace ensuring a high QoS for the system may be paramount. This is particularly the case for workspaces that might utilise streaming audio or video. It is likely that the QoS provided by the system will also be monitored by the higher-level application.

User Interface components – the peer would need to provide various user interfaces to the various operations it can perform. For example, a workspace list window, a workspace window, etc

User Details – details of the user. This can include information that is also made publicly available to the rest of the system

Workspace List Data – details of which workspaces the user is interested in.

Workspace Creation – this functionality grouping allows for the creation of shared workspaces. The repository manager stores workspaces that have been created.

Workspace Details – details about all the workspaces that have been created by this peer. This can include information about what exists within the workspace, and details of the users who have access to it. It is also a possibility that details about workspaces created by other peers could be cached here (in essence acting as workspace adverts).

Authentication – this functionality grouping deals with authenticating those peers/users who request data, or make changes. This has been described in section 3.1.1 (Generic Server Architecture).

Repository Manager – this functionality grouping essentially manages the storage of the data that is utilised by the application, as described in section 3.1.1 (Generic Server Architecture).

Checking in/out data – this functionality grouping manages data transactions, as described in section 3.1.1 (Generic Server Architecture).

3.4. Search System Reference Architectures

The instantiated architectures for a search system extend the generic architectures presented in section 3.1. Consequently replicated functionality groupings have not been re-described.

3.4.1 Server Architecture with Group descriptions

The search system server architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- Index of data held on nodes

- Details of peers on the network
- Data Search Mechanism
- Data Filtering Mechanism

The architecture is presented in Figure 29.

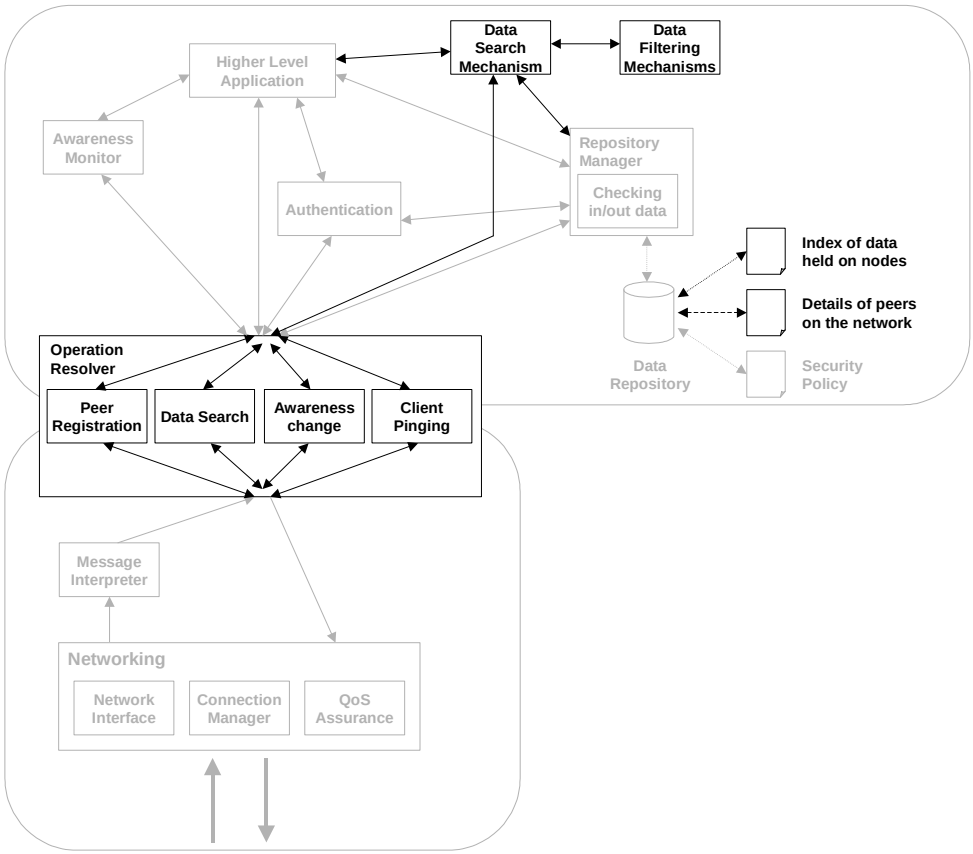


Figure 29 - Search System Architecture for Server Peers

Operation resolver – For a search system architecture the server needs to cater for a number of operations. These include:

- *Peer Registration* – allowing a client peer to register itself with the P2P system. This should not only occur when a new peer connects for the first time, but also whenever a peer connects, so that the server always hold up to date information about the users of the system
- *Data Search* – allowing a peer (or user) to search for data on the P2P system. This is particularly important as it is the main focus for this type of system
- *Awareness change* – allowing a client peer to be able to inform the server (and thus other relevant client peers), that there has been a change in state (e.g. from 'On-line' to 'Off-line').

- *Client Pinging* - allowing the server to check on the status of the peers/users on the network. This is necessary should peers/users disconnect from the network without being able to inform the server.

Data Search Mechanism – this functionality grouping represents the search mechanism that is used to search the index data that is held within the Data Repository. If desired it could also be tied in with data filtering mechanisms.

Data Filtering Mechanism – this functionality grouping represents possible data filtering techniques that could be used to help search the index data. Such mechanisms could include collaborative filtering techniques, for example, allowing users of the system to attach recommendations to data or the providers of data. These recommendations could capture how reliable the data source is, or the quality of the actual data.

Index of data held on nodes – details of what data each peer within the system is making publicly available. This index is what the server searches when it tries to locate data.

Details of peers on the network – details about all peers that are participating within the system.

3.4.2 Client Architecture with Group descriptions

The search system client architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- Publicly Available Data
- Data Search Mechanism
- Peer Details
- User Interface Aspects

The architecture is presented in Figure 30.

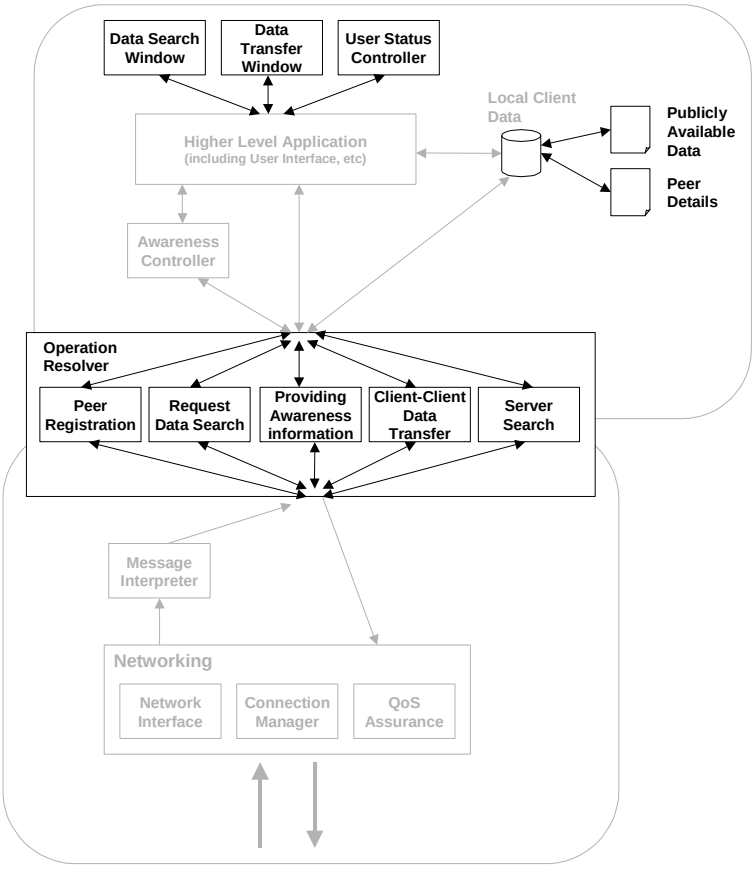


Figure 30 - Search System Architecture for Client Peers

Operation resolver - the client within a Search System application has to deal with a number of operations. These include:

- *Peer Registration* – the client peer needs to be able to register itself with the server. This should be done whenever a peer connects to the system.

- *Request Data Search* – the client peer needs to allow users to search the server for data that has been made publicly available by other peers.

- *Providing Awareness Information* – the client peer should allow the user to specify and publish their peer's awareness information to the server (and thus other peers). For this type of system most aspects can be automated (for example, if the user shuts down the client, then the server could be automatically informed of this).

- *Client-Client Data Transfer* - the client needs to support the exchanging of data with other clients.

- *Server Search* - the client needs to be able to locate a server upon start-up.

User Interface components – the client would need to provide various user interfaces to the various operations it can perform. For example, a data search window, a data transfer window, etc

Publicly Available Data – this represents the data that the peer is making publicly available to the rest of the system.

Peer Details – details of the peer. This information is likely to include index information for the data held on the peer, as well as information about the peer's resources (bandwidth, etc).

3.4.3 Decentralised Architecture with Group descriptions

The search system decentralised architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- Publicly Available Data
- Peer Details
- User Interface Aspects
- Data Search Mechanism

Many of these functionality groupings are similar to those presented in the client architecture (section 3.4.2). The architecture is presented in Figure 31.

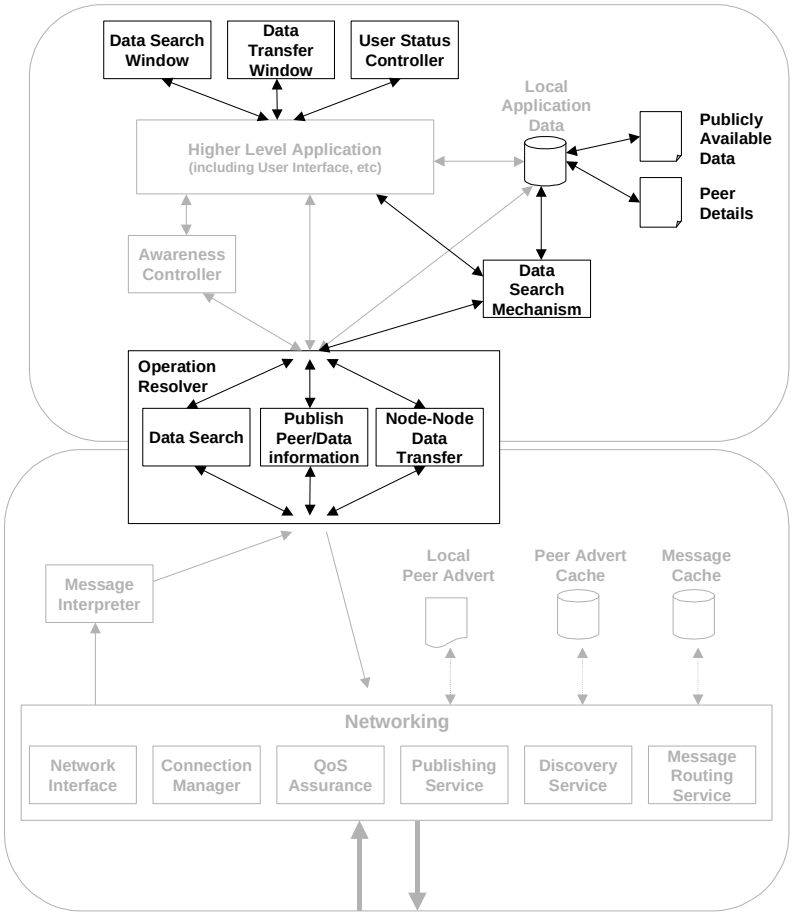


Figure 31 – Search System Architecture for Decentralised Peers

Operation resolver - a peer within a decentralised Search System application has to deal with a number of operations. These include:

- *Data Search* – the peer needs to allow users to search the network for data that has been made publicly available by other peers.

- *Publish Peer/Data Information* – the peer needs to be able to publish its peer/data information to the rest of the network. Such broadcasting should occur whenever the status of the peer/data changes. Some aspects of this can also be automated (for example, if the user shuts down the application, then this fact can be broadcast to the rest of the network).

- *Node-Node Data Transfer* - the peer needs to support the exchanging of data with other peer.

User Interface components – the peer would need to provide various user interfaces to the various operations it can perform. For example, a data search window, a data transfer window, etc

Data Search Mechanism – this functionality grouping represents the search mechanism that is used to search the data that is held by the peer. If desired it could also be tied in with data filtering mechanisms.

Publicly Available Data – this represents the data that the peer is making publicly available to the rest of the system.

Peer Details – details of the peer. This information is likely to include index information for the data held on the peer, as well as information about the peer's resources (bandwidth, etc).

3.5. Document Management Reference Architectures

The instantiated architectures for a document management system extend the generic architectures presented in section 3.1. Consequently replicated functionality groupings have not been re-described.

3.5.1 Server Architecture and Group descriptions

The document management server architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- Version Control
- User Details
- Documents

The architecture is presented in Figure 32.

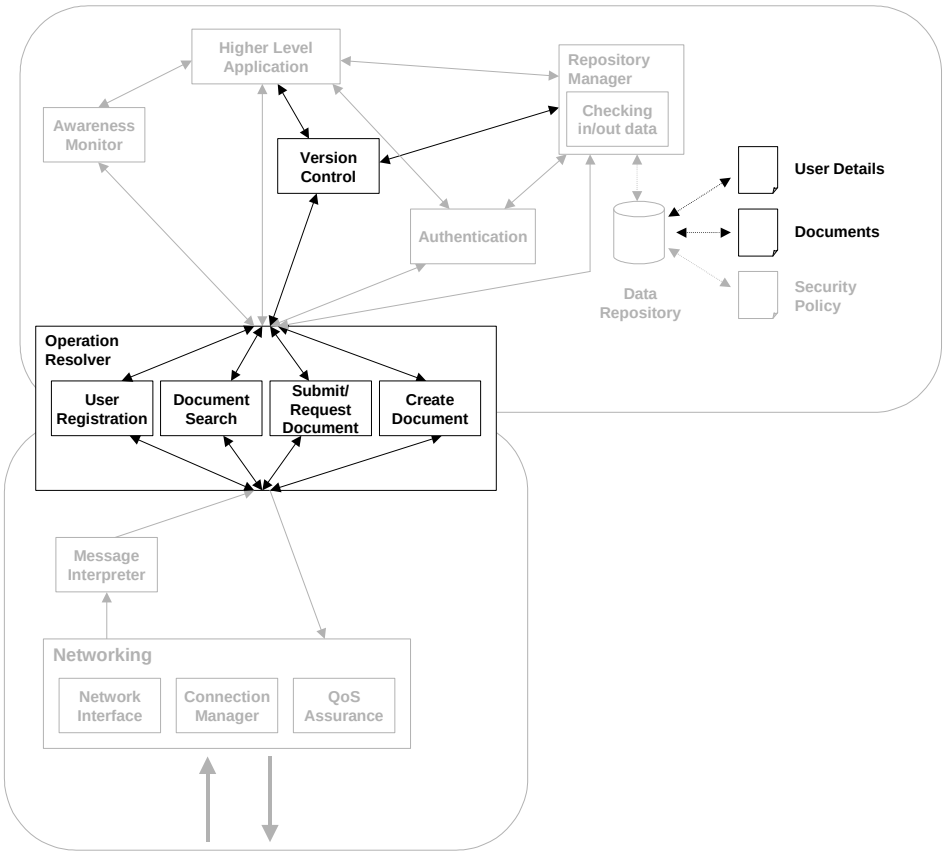


Figure 32 – Document Management Architecture for Server Peers

Operation resolver – For a document management architecture the server needs to cater for a number of operations. These include:

- *User Registration* – allowing a user to register itself with the P2P system.
- *Document Search* – allowing a user to search for documents held by the server.
- *Submit/Request Document* – allowing a user to either submit a document they have been working on, or to request on from the server. Both submitting and requesting a document are likely to be influenced by access rights. Submitting a document would likely involve some form of version control.
- *Create Document* – allowing a user to create a document on the server. It is likely that the user will also have to provide additional information such as assigning the document access rights, and specifying what should happen when changes are made (i.e. does it create a new version of the document.)

Version Control – it is likely that a document management system will require some form of document version management, this functionality grouping, along with the Repository Manager sets out to achieve this.

User Details – details about all users of the application.

Shared Documents – this represents the documents that have been created and made publicly available.

3.5.2 Client Architecture and Group descriptions

The document management server architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- User Interface Aspects
- User Details
- Local Documents

The architecture is presented in Figure 33.

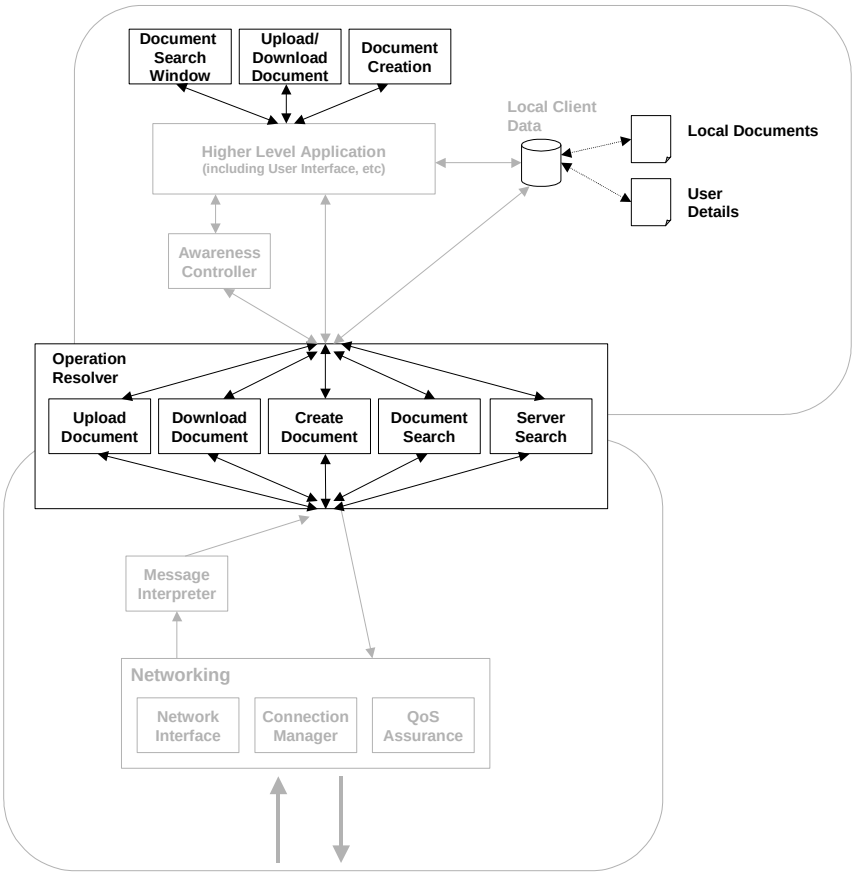


Figure 33 - Document Management Architecture for Client Peers

Operation resolver - the client within a Document Management application has to deal with a number of operations. These include:

- *User Registration* – the client peer should allow users to register with the server when they first use the system.

- *Upload Document* – the client peer should allow users to upload documents to the server.

- *Download Document* – the client peer should allow users to download documents from the server.

- *Create Document* – the client peer should allow users to create new documents and to add them to the server. It is also likely that when creating a new document, the user will also have to specify access rights and what should happen when the document is changed (e.g. whether a new version is created).

- *Document Search* – the client peer needs to allow users to search the server for documents that have been made publicly available by other peers.

- *Server Search* - the client needs to be able to locate a server upon startup.

User Interface components – the client would need to provide various user interfaces to the various operations it can perform. For example, a document search window, and a document creation window, etc

Local Documents – this represents any documents that the user may have downloaded and be editing locally.

User Details – details of the user of the client.

3.5.3 Decentralised Architecture and Group descriptions

The document management system decentralised architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)

- Local Documents

- User Details

- User Interface Aspects

- Version Management

Many of these functionality groupings are similar to those presented in the client architecture (section 3.5.2). The architecture is presented in Figure 34.

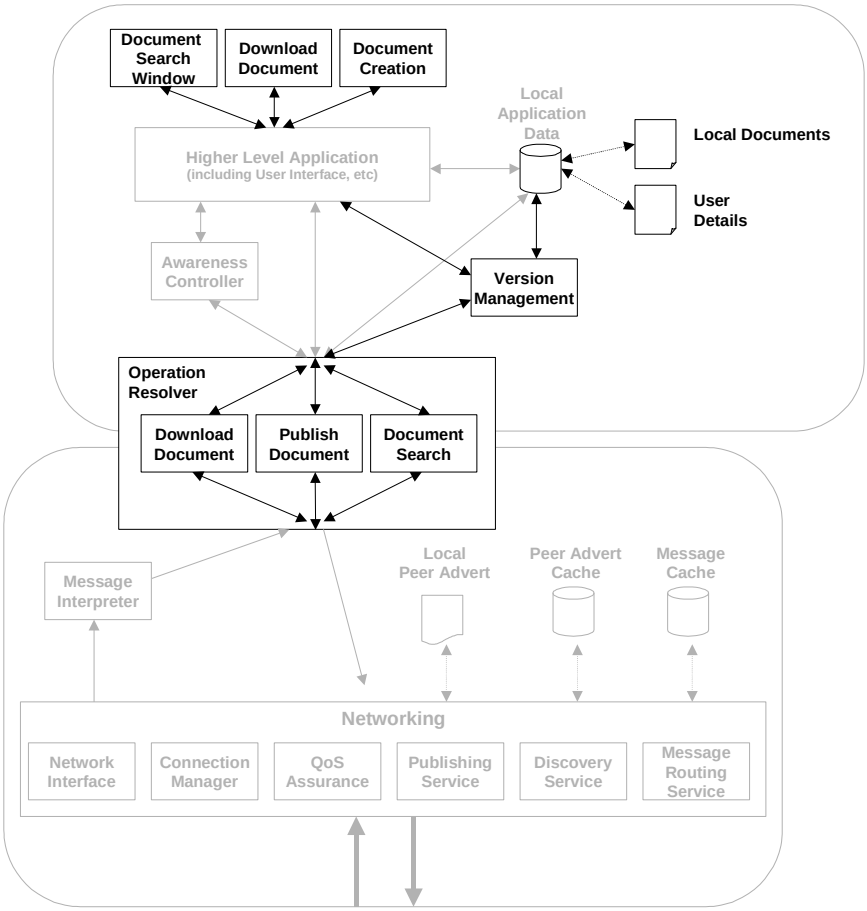


Figure 34 – Document Management Architecture for Decentralised Peers

Operation resolver - a peer within a decentralised Document Management application has to deal with a number of operations. These include:

- *Publish Document* – the peer should allow users to publish documents they have created or altered on to the network. It may also be necessary that when creating a new document, the user can specify access rights and what should happen when the document is changed (e.g. whether a new version is created).

- *Download Document* – the peer should allow users to download documents from other peers located on the network.

- *Document Search* – the peer needs to allow users to search the network for documents that have been made publicly available by other peers.

User Interface components – the peer would need to provide various user interfaces to the various operations it can perform. For example, a document search window, and a document creation window, etc

Version Management – it is likely that a document management system will require some form of document version management, this functionality grouping, sets out to achieve this.

Being decentralised it is likely that this component will need to be more sophisticated than its semi-centralised counterpart. It might be necessary to enforce additional restrictions on the documents.

Local Documents – this represents any documents that the user may have downloaded and be editing locally.

User Details – details of the user of the client.

3.6. Computational Systems Reference Architectures

The instantiated architectures for a computational system extend the generic architectures presented in section 3.1. Consequently replicated functionality groupings have not been re-described.

3.6.1 Server Architecture and Group descriptions

The computational system server architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- Work package Management
- Peer Details
- Work packages
- Peer-Work package Assignment
- Messages to be routed

The architecture is presented in Figure 35.

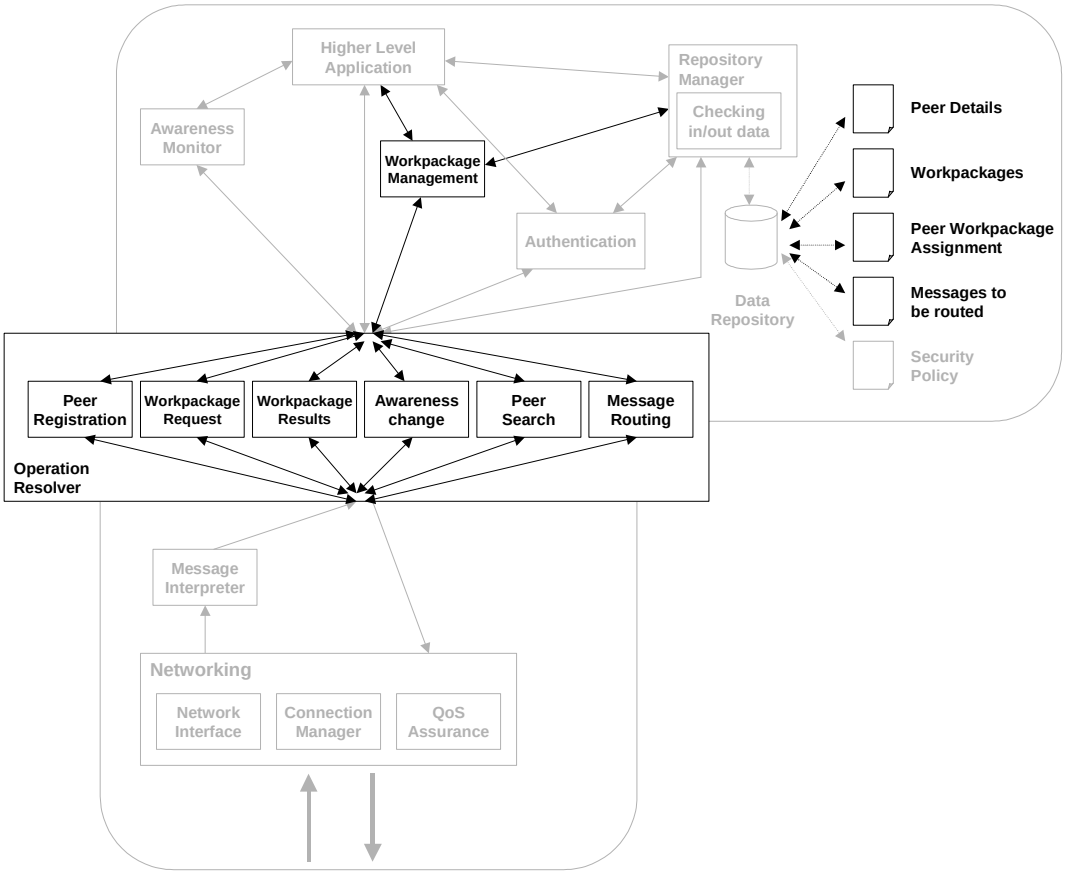


Figure 35 – Computational System Architecture for Server Peers

Operation resolver – For a computational system architecture the server needs to cater for a number of operations. These include:

- *Peer Registration* – allowing a client peer to register itself with the P2P system.
- *Work package Request* – allowing a client peer to request a work package for processing.
- *Work package Results* – allowing a client peer to communicate the results of a processed work package back to the server.
- *Awareness change* – allowing a client peer to be able to inform the server (and thus other relevant client peers), that there has been a change in state (e.g. from 'On-line' to 'Off-line').
- *Peer Search* – allowing a client peer to search for other peers that are using the system.
- *Message Routing* – allowing messages to be routed via the server, rather than directly between client peers. It is necessary to support this so users can still send messages even if the target user is not currently connected to the network.

Workspace Management – a computational system would most likely breakdown any computational tasks into work packages. A server within the system would need to manage these work packages, assign work packages to peers, collect processed results, error check, etc.

Peer Details – details about all peers that are registered with the system.

Work packages – this represents all the work packages that need to be processed within the system.

Peer-Work package assignment – this captures details of what work packages each peer has been assigned to process.

Messages to be routed – messages that a client peer has requested to be routed via the server.

3.6.2 Client Architecture and Group descriptions

The computational system client architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- Peer Details
- Work package
- Higher Level Aspects

The architecture is presented in Figure 36.

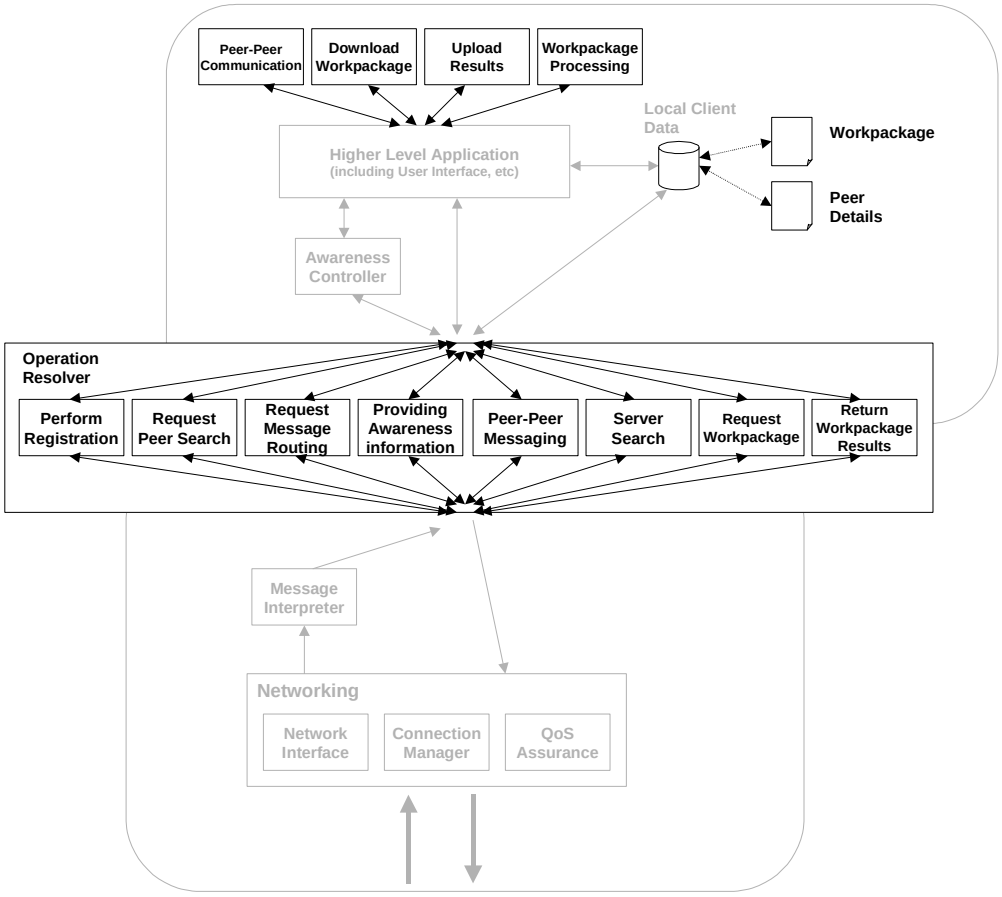


Figure 36 – Computational System Architecture for Client Peers

Operation resolver – For a computational system architecture the server needs to cater for a number of operations. These include:

- *Perform Registration* – allowing a client peer to register itself with the P2P system server.

- *Request Peer Search* – depending on the specific application the client peer needs to be able to search the server for other users.

- *Request Message Routing* – depending on the specific application the client peer should allow for the routing messages via the server, should the target peer not be currently online.

- *Providing Awareness Information* – the client peer should publish its awareness information to the server. Some aspects of this can also be automated (for example, if a user shuts down the client, then the server could be automatically informed of this).

- *Peer-Peer Messaging* – depending on the specific application the client may need to support the exchanging of messages/data between other clients.

- *Server Searching* – the client needs to be able to locate a server upon start-up.

- *Request Work package* – the client peer needs to be able to request a work package from the server for processing.

- *Return Work package Results* – the client peer needs to be able to communicate the results of a processed work package back to the server.

High Level components – although the client peer would possess some form of user interface it would mainly focus on the functional aspects of the system. For example, the client would need to be able to process the work package, the downloading/uploading of work packages and the respective results, etc.

Peer Details – details about the peers.

Work packages – this represents the work package that is currently being processed by the client.

3.6.3 Decentralised Architecture and Group descriptions

The computational system decentralised architecture expands the generic architecture with the following functionality groupings:

- Operation resolver (expanded)
- Work packages
- Peer Work package Assignment
- Peer Details
- Higher Level Aspects
- Workspace Management
- Repository Manager, Authentication and Checking in/out data

Many of these functionality groupings are similar to those presented in the client and server architectures (sections 3.6.2 and 3.6.1). The architecture is presented in Figure 37.

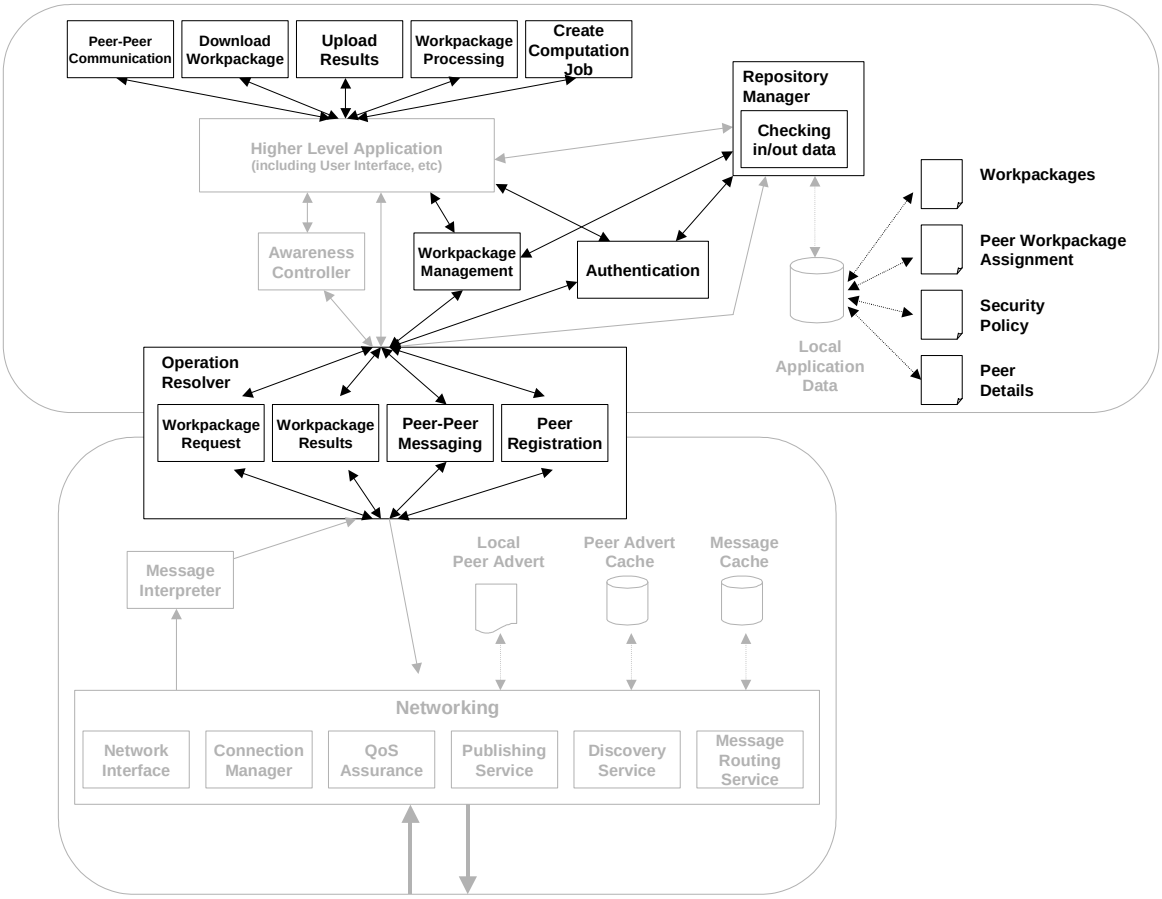


Figure 37 – Computational System Architecture for Decentralised Peers

Operation resolver – For a computational system architecture the decentralised peer needs to cater for a number of operations. These include:

- *Peer Registration* – allowing the peer to register itself with another peer that seeks to distribute computation workload, and likewise allowing other peers to register themselves with this peer should it be the source of the computation.

- *Peer-Peer Messaging* – depending on the specific application the peer may need to support the exchanging of messages/data between other peers.

- *Work package Request*– the peer needs to be able to request a work package from the serving peer for processing. In addition if the peer is the source of the computation, it needs to be able to process such a request.

- *Work package Results* – the peer needs to be able to communicate the results of a processed work package back to the serving peer. In addition if the peer is the source of the computation, it needs to be able to collate together the returned results.

High Level components – although the peer would possess some form of user interface it would mainly focus on the functional aspects of the system. For example, the peer would

need to be able to process the work package, the downloading/uploading of work packages and the respective results, manage the work packages, set-up a computation, etc.

Peer Details – details about the peer and other peers that are taking part in a distributed computation.

Work packages – this can represent the work package that is currently being processed by the peer. It can also represent all the work packages that need to be processed if this peer is the source of the computation.

Peer-Work package assignment – this captures details of what work packages each peer has been assigned to process, should this peer be the source of the computation.

4. Architectural comparison with existing P2P systems

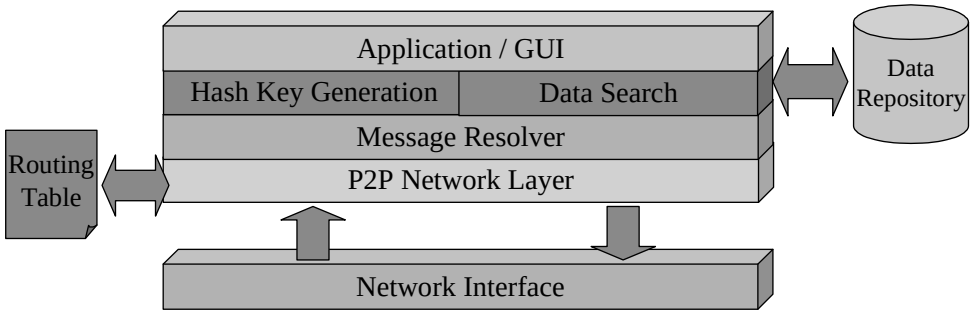
This section compares four existing P2P systems with the reference architectures that have been presented in this document. Not only will this provide an analysis of how these systems architectural structures compare with the relevant reference architectures, but it will also illustrate one way in which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

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

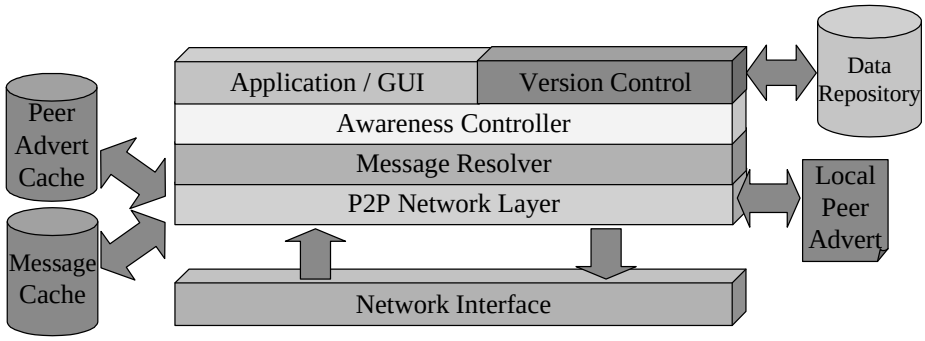
4.1. Freenet

Freenet[6] is a decentralised P2P system 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.

In respect to the reference architectures, Freenet can be considered to be a simple version of a decentralised document management system (discussed in section 2.2.5). However unlike the architectures that have been provided in this document, Freenet is based on providing anonymity and so cannot provide sophisticated document management facilities such as versioning or authentication. Instead Freenet puts 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 38 provides a general layered based approximation of how Freenet is structured.



(a)



(b)

Figure 38 – a) Layered architecture representation of Freenet, b) Document Management Decentralised Reference Architecture (from section 2.2.5)

In comparison with the decentralised document management reference architecture, it can be seen that there is a large degree of similarity (with Freenet 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 Freenet representation lacks the *Peer Advert Cache* and the *Local Peer Advert* aspects, but gains a *Routing Table.*

Freenet 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, Freenet 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 Freenet representation lacks the *Awareness Controller* and *Message Cache*.

The main differences between Freenet and the reference architectures occur with the actual handling of the stored data. As has been mentioned, Freenet deals with this in an anonymous fashion and so 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.

4.2. Napster

Napster[10] is a semi-centralised file-sharing system, 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 system 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 system uses similar presence techniques as those discussed in the

document 'Providing Presence within P2P systems'[12]. Users need to register with the server node before they can make use of the system, and this allows Napster to provide simple authentication. Similar functionality is represented in the search system 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 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 39 provides a general layered based approximation of how a Napster server is structured. The only difference to the search system server reference architecture presented in section 2.2.4 is the replacement of the generic *Data Repository* with separate *User Data* and *Data Catalogue* repositories.

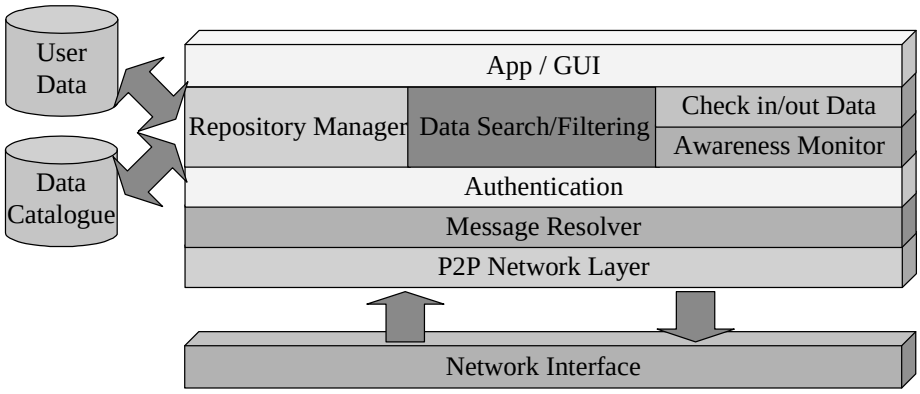


Figure 19 – Layered architecture representation of the server peers in Napster

The structure of the client nodes within Napster closely mirrors that which is presented in the search system 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 systems presence information up to date. Similar functionality can be represented by the search system client node reference architecture (figure 11).

In summary, the structure of Napster is comparable to a hybrid of the search system 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.

4.3. SETI@home

SETI@home[11] is a semi-centralised computational system 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.

The structure of the client nodes within SETI@home essentially mirrors that as presented in the client computational reference architecture (figure 17). 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 system, 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 40 provides a general layered based approximation of how the SETI@home server node is structured. The only difference to the computational system server reference architecture presented in section 2.2.6 is the replacement of the generic *Data Repository* with separate *User, Science* and *Work package Database's.*

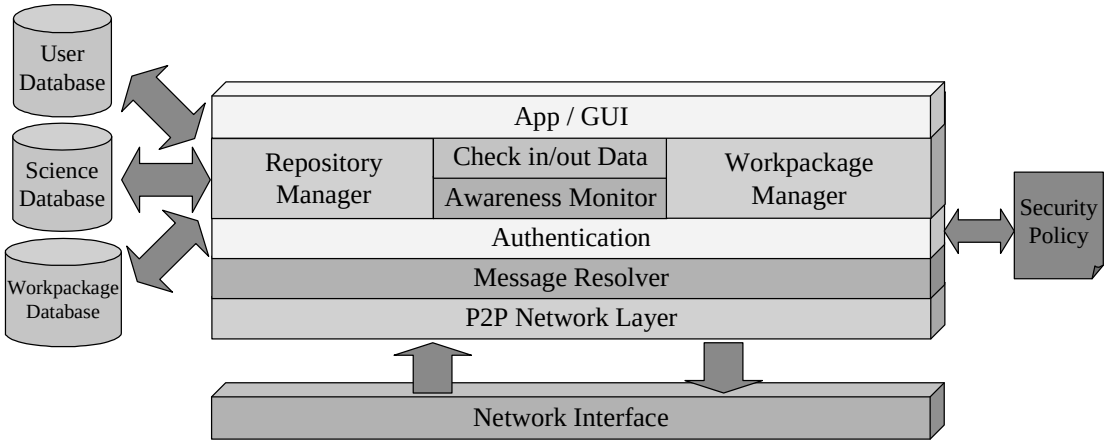


Figure 40 – Layered architecture representation of the server peers in SETI@home

In summary, the structure of SETI@home is comparable to the semi-centralised computational reference architecture presented in section 2. The server nodes make use of three databases to store data that is required by the system. The client nodes do not possess any autonomy.

4.4. Jabber

Jabber[9] is a 'universal' Instant Messenger (IM) protocol. It is different to more widely known IM applications such as ICQ[3] and AIM[13] because its XML based protocol allows,

in theory, a Jabber client 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 Jabber 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.

In terms of logical network architecture, Jabber can be considered to use a 'multiple server nodes architecture' (discussed in D5). This means that on one hand Jabber operates in a similar fashion to alternative IM applications, in that a client peers 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 (figure 5). 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., AIM and ICQ) cannot be established directly, and instead has to be routed via the servers. Communication 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 41 provides a general layered based approximation of how the Jabber server nodes are structured. The only differences to the instant messenger server reference architecture presented in section 2.2.2 is the replacement of the generic *Data Repository* with separate *Server Details*, *User Details* and *Messages to be routed* repositories.

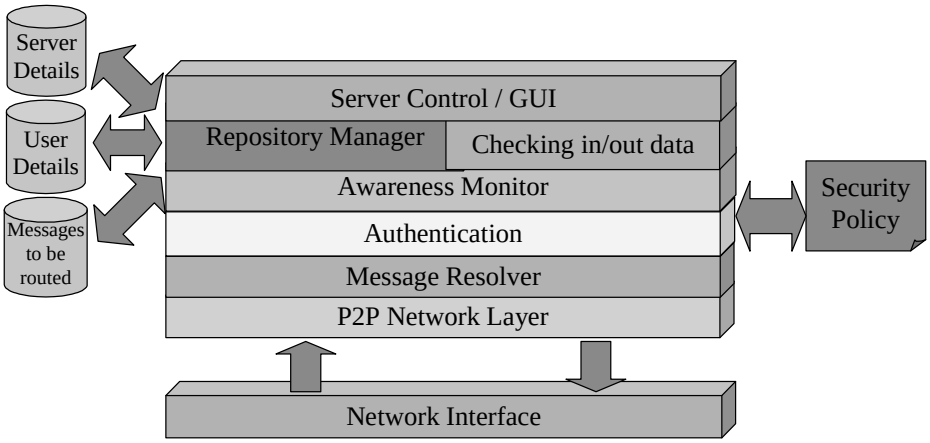


Figure 41 – Layered architecture representation of the server peers in Jabber

In summary, the structure of Jabber does not differ significantly from the IM reference architectures that have been presented in section 2. 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, AIM, etc. This is mainly due to the fact that a Jabber server can connect to other servers and also to other IM protocols.

5. Architectural comparison with existing P2P development tools and methodologies

This section looks at JXTA [1] and examines how it compares to the reference architectures that have been presented in this document. When making such a comparison it is important to realise that the intended use of JXTA is to allow for the development of any type of P2P system, whereas the reference architectures in this deliverable represent specific applications. Consequently JXTA has a very general structure and approach, and focuses on providing a supporting layer between the network and the P2P application.

It is possible to perform a degree of comparison against this supporting layer that JXTA provides.

5.1. JXTA

Sun's JXTA [1] project aims at providing developers with a general protocol and API for the development of P2P applications. Essentially it can be viewed as providing a layer between the physical network and the application that handles all issues relating to P2P. This technology has been reviewed in more detail in D1: Comprehensive Survey of contemporary P2P technology [8]. An overview of the JXTA architecture is presented in figure 42.

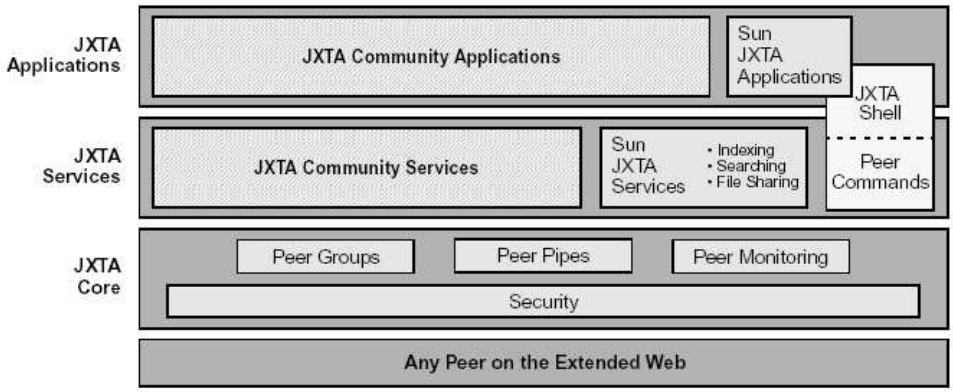


Figure 42 – The JXTA architecture

The JXTA architecture is broken down into four layers. The lowest level essentially represents the peer on the network. This is comparable to the *Network Interface* layer presented in the layered based reference architectures (Section 2).

Above this is the JXTA Core layer that deals with the general management of the peer and network connections. At this level issues such as security, routing and general communication management are considered, and consequently it is comparable to the *P2P Network Layer* from the layered based architectures.

The JXTA Services layer handles higher-level concepts such as indexing, searching and file sharing. The idea behind these is that a generic service (for example, a file sharing service) can exist and then be used by any application that is built on top. Although this can make for

easy development of P2P applications, it does restrict the options of the developer. The reference architectures presented in this document assume that any services will be provided by the individual applications and so do not cater for a service layer. That aside, there does exist functionality similarities between the JXTA Services layer and the layers that can exist between the *P2P Network Layer* and *Application/GUI layer* of the layered based reference architectures (for example, data searching and awareness monitoring).

The JXTA Applications layer represents the actual applications that make use of the P2P technology (and the services). In the generic reference architectures presented in this deliverable, the split between the application and P2P layer is less clear as it is believed that such a strict separation cannot be fully achieved. However, the amount of application specific functionality is likely to increase the further you move up the reference architecture.

In summary, JXTA being a P2P API is not geared towards one particular type of application. However it operates by essentially acting as a layer between the network and the P2P application, and consequently does mirror some of the layers from the generic architectures presented here. One of the main differences with JXTA is that it possesses a services layer that provides a range of generic services that can be used by the applications built on top. The reference architectures in this deliverable do not possess a services layer as this was believed to be too specific and restrictive.

6. Summary

This deliverable has presented a set of reference architectures that can act as a valuable resource for developers of co-operative P2P systems. Reference architectures have been provided for generic co-operative P2P systems, instant messenger P2P systems, shared workspace P2P systems, distributed search P2P systems, document management P2P systems and computational P2P systems.

The deliverable has also presented a set of instantiated architectures that illustrate how the reference architectures can be further expanded upon to meet more specific objectives. These architectures are too specific, however, to be regarded as reference architectures in their own right.

The deliverable has also compared some key existing P2P applications with the reference architectures, to examine what similarities and differences exist. A comparison has also been made with Sun's JXTA as this aims to provide some of the functionality that is encapsulated in the reference architectures.

References

[1] JXTA. Java P2P API. Web site http://www.jxta.org
[2] P2P Architect Project "D5: Report on the Dependability Properties of P2P Architectures", 0305D01_DependabilityReport.doc
[3] ICQ. Instant Messenger Application. Web site http://www.icq.com
[4] MSN Messenger. Instant Messenger Application. Web site http://specials.msn.com/ms/default.asp
[5] P2P Architect Project "D8: Models and specification primitives for building dependable P2P Systems/Applications", D8_2611.doc
[6] FreeNet. More information can be found at the URL http://freenet.sourceforge.net
[7] Gnutella. More information can be found at the URL http://www.gnutella.com
[8] P2P Architect Project "D1: Comprehensive Survey of contemporary P2P technology", 0101F05_P2P Survey.doc
[9] Jabber. Instant Messaging Protocol. More information can be found at the URL http://www.jabber.com
[10] Napster. More information can be found at the URL http://www.napster.com
[11] SETI@home. More information can be found at the URL http://setiathome.ssl.berkeley.edu
[12] "Providing Presence within P2P systems", Internal Document. Lancaster University
[13] AOL Instant Messenger (AIM). Web site http://www.aol.co.uk/aim/
[14] ISO OSI Reference Model