Project no.:
AAL-2009-2-137

PeerAssist

A P2P platform supporting virtual communities to assist independent living of senior citizens

Deliverable 3.5
"P2P overlay networks for PeerAssist"

Lead Participant/Editor	UoA/ Christos Xenakis
Authors	Christos Xenakis, Foivos Demertzis, Giannis karras, Nikos Giannopoulos

Table of Contents

1 Introduction 1
2 P2P technology 1
2.1 Evolution 2
2.2 Latest and most prominent P2P 4
2.3 P2P architectures....................4
2.3.1 Purely Decentralized....................4
2.3.2 Hybrid Architecture....................5
2.4 Discovery mechanisms for P2P systems 6
2.4.1 Centralized indexes and repositories....................6
2.4.2 Flooding broadcast of queries....................7
2.4.3 Routing Model....................7
2.5 P2P networks structure 8
2.6 Benefits....................9
2.7 Drawbacks....................9
3 Analysis of available P2P technologies....................10
3.1 JXTA....................10
3.2 Microsoft Windows P2P 12
3.3 The Peer-to-Peer Trusted Library 13
3.4 JINI....................14
3.5 Enterprise service bus (ESB) - Mule....................15
3.6 Unmanaged Internet Architecture (UIA)....................16
3.7 MACEDON and Mace 17
3.8 Ezel 21
3.9 Microsoft Groove 21
3.10 Summary....................22
4 Platform selection process and criteria 23
4.1 Connectivity, communication, grouping 23
4.2 Service, Interoperability, Openness and Extensibility 24
4.3 System architecture 25
4.4 Efficiency and scalability 25
4.5 Security and trust 25
4.6 OSGI 26
4.7 Support a wide range of end-user devices 27
4.8 Why JXTA ? 27
5 JXTA 28
5.1 Overview....................28
5.2 Peer 30
5.3 Peer group 31
5.4 Service....................31
5.5 Modules....................33
5.6 Message....................34
5.7 Pipes....................34
5.8 Advertisement 37
5.9 Security....................38
5.10 IDs....................39

5.11 Network architecture 39
5.11.1 Shared Resource Distributed Index (SRDI).. 40
5.11.2 Queries.. 40
5.12 Firewalls and NAT.. 42
5.13 JXTA protocols.. 43
5.13.1 Peer Discovery Protocol.. 44
5.13.2 Peer Information Protocol.. 44
5.13.3 Peer Resolver Protocol.. 45
5.13.4 Pipe Binding Protocol.. 45
5.13.5 Endpoint Routing Protocol.. 46
5.13.6 Rendezvous Protocol.. 47
5.14 OSGI (IAN) 47
6 Conclusions 48
7 References.. 49

1 Introduction

This document describes the work has been done in Task 3.4 of the project Peerassit entitled “Peer-to-peer overlay network selection”. In this task, we have analyzed and evaluated the existing platforms for building P2P networks. Platforms evaluation has been conducted considering the entire set of PeerAssist requirements including functional and non functional. The most appropriate technology for PeerAssist is JXTA, which has been implemented in a small scale testbed for peerassist. The basic functionality of JXTA has been tested and possible improvements and enhancements has been drawn. The rest of this deliverable is organized as follows:

Chapter 2 briefly presents and analyzes the evolution of P2P technology focusing on its most important representatives. It describes the different architectures employed in P2Ps that provide decentralization, as well as the different discovery mechanisms utilized to locate specific data and resources within the system. Finally, it outlines the possible advantages and drawbacks associated with P2P applications and systems.

Chapter 3 examines a number of existing tools and platforms that have been developed to facilitate the implementation of P2P systems and applications. A developer may use one of them to build its own P2P system or alternatively provide a custom tailored one.

Chapter 4 presents and analyzes the criteria and the selection process followed in choosing the most appropriate available platform to build Peerassist. The specific criteria used derive from the users and services requirements of the PeerAssit platform, determined in WP2, including both functional and non functional. We have concluded that the most appropriate available tool is JXTA.

Chapter 5 presents a detailed description of JXTA focusing on its functional components as well as the provided services. Moreover, the granted API, which enables developers to interact with JXTA and implement their own systems and application on top of JXTA has been analyzed and studied. Finally, the entire JXTA functionality and the provided services have been evaluated and tested in order to assess their conformance with peerassist. This gave a significant input in the forthcoming design phase of the peerassist platform as well as the followed design choices.

Finally, chapter 6 contains our conclusions.

2 P2P technology

Peer-to-Peer (P2P) is defined as a concept allowing direct communication between individual computers by some, or as a set of networking design principles by others. The actual definition of what P2P is varies according to researchers, but most agree that a peer-to-peer system traditionally rejects the client/server model and the underlying hierarchy it imposes between computers operating on a network.

A P2P network is a distributed network composed of a large number of distributed, heterogeneous, autonomous, and highly dynamic peers in which participants share a part of their own resources such as processing power, storage capacity, softwares, and files contents. The participants in the P2P network can act as a server and a client at the same time. They are accessible by other nodes directly, without passing intermediary entities. The P2P models can be

pure or hybrid. In pure P2P any single, arbitrary chosen terminal entity can be removed from the network without having the network suffering any loss of network service. Hybrid P2P allows the existence of central entities in its network to provide parts of the offered network services.

P2P became widely popular for the first time through file exchange applications such as Napster, Kazaa or Gnutella, instant messaging application such as ICQ or Yahoo! Messenger, or when we started “calling” over the Internet with Skype. Others discovered P2P with Groove(now Microsoft Groove), a software application developed by Groove Networks in the 1990's.

Since the year 2000, P2P technologies have boomed which resulted in a high number of file sharing and chatting applications. Unfortunately, some P2P applications have often been associated with illegal file transfers, copyright infringements on music and films, and all sorts of other illegal or unsafe activities. Those using P2P technologies have been accused of participating in a movement damaging the economy.

2.1 Evolution

The general idea behind P2P is that computer devices belonging to users should act both as client and server on the network and that they should connect directly to each other, that is, without the need of a central server, to exchange information or services. A P2P software application is enabling such operations between several computer devices connected to each other via a network.

ICQ

ICQ was one of the first P2P applications made available to a wide audience. It allowed users to exchange instant messages and to be notified when they were on-line. Purists do not consider ICQ as a pure P2P application, since it is using a central server to identify users appearing on the network and to notify other connected users of their presence. Once the notification is sent, users communicate directly. Therefore it is a combination of client/serverand P2P design principles.

Napster

Three years after ICQ, Napster appeared on the Internet in 1999 and provided users with the possibility to exchange MP3 audio files. Users uploaded their file list on a central server. Then, they sent their queries for specific files to that server, which replied with a list of IP addresses (i.e., Internet locations) of users having those files. At last, they established a connection with these users to download files. Of course, there was a risk of obtaining obsolete IP addresses since users would connect from different locations or would be assigned new IP addresses each time they connect to the Internet.

Gnutella

Gnutella appeared in March 2000. Like Napster, it was a file exchange application. However, it differed significantly from Napster in its way of establishing contact with other peers and querying for files. Instead of contacting a central server, the Gnutella application would try to connect to a predefined set of nodes to obtain a list of IP addresses of other nodes. It would then try to connect to these nodes to obtain more IP addresses until a sufficient set of successful connection was reached. Unsuccessful addresses were automatically discarded. This type of bootstrapping method made the application decentralized and independent of the current node network topography. Nodes could join and quit the network from anywhere without hampering the systems' capacity to establish

connections with other nodes as long as seed nodes were available.

Once the connection was established, each node could send file queries to the pool of connected nodes. They would forward these to the nodes they knew, which would forward these to nodes they knew, etc... etc... provoking a cascading effect. When a node possessed the researched file, it would notify the original node sending the query, so that a file transfer could be started between them. Each query would contain a positive number called time to live (TTL) which was decreased each time the query was forwarded to another node. When TTL reached 0, the query was not forwarded anymore. The list of visited nodes was kept in each copy of the propagated query to avoid loops. Unfortunately, it did not prevent nodes from receiving the same query twice, through different paths. The increasing number of users resulted into an incredible amount of useless traffic between nodes. The consequence was that the network congestion around servers was now propagated around all users.

The architecture of Gnutella presented a benefit over the architecture of Napster. The latter could easily be attacked or stopped since it was relying on a central server. With Gnutella, attacking or stopping a node was inefficient, since other nodes could take over and compensate for the missing node.

Kazaa

Kazaa appeared in March 2000 and introduced a new concept: super nodes. Instead of having each node maintaining its own list of shared files to share locally, they were uploaded to super nodes at regular intervals. User's queries were sent to super nodes. Then, these nodes would reply with the list of nodes offering the searched file. Queries were not propagated in all directions anymore. The user could then establish a connection with the remote node containing the requested file and start the transfer. This method induces significantly less network traffic than Gnutella-like applications. Super nodes are automatically chosen by the system according to their capacity (storage, band-width, etc...). In this model, some individuals in the community are being given more responsibilities to organize the life of the community.

BitTorrent

BitTorrent appeared in July 2001. It introduced a new way of exchanging files on a P2P network. Instead of focusing on a single peer for the transfer of large files, the query peer would obtain parts of it from different peers simultaneously, creating a torrent of data. This method allows faster download time compared to traditional P2P transfer methods. If you were unlucky and downloaded a file from a peer having a low bandwidth connection, you had no other option than to wait or to cancel the connection and try with another peer, hoping for a better connection. With this new method, even a set of low bandwidth peers can generate rapid transfer times.

Freenet

Freenet appeared in 1999 (at least conceptually). The objective of this P2P application was to let its user publish and exchange decentralized information in pure anonymity using cryptography and special routing functions between nodes, making it hard to trace peers querying for information. Other applications such as Napster, Gnutella and Kazaa do not provide anonymity. Users know who they are downloading data from and they also know where users queries are coming from, making it relatively easy to trace them.

2.2 Latest and most prominent P2P

At the beginning of the millennium, P2P had multiple initial objectives. They started with chatting, then with file sharing and avoiding central server computers in general. However, these early objectives obscured another goal of P2P application: distributed computing .

One of the most successful examples is the SETI@home project. Internet users can download a free program that would use the idle time of their computer to analyze radio telescope data. When finished, the results are sent back to a central server and new data is downloaded for analysis. Although this application is not P2P in its design, it illustrates how many computers can solve a large divisible problem. A real-life example of divisible problems is mowing the lawn. One person can do it alone or several can simultaneously do it by taking care of a part of the yard.

There are some indivisible problems, such as checking one's account balance when withdrawing money. You need to centralize all account transactions in one location in order to compute the balance and make sure there is money available when performing the withdrawal. However checking many balance accounts is a divisible problem: the account checking can be spread over a set of computers. Each account can be verified simultaneously. In general, indivisible problems are better served in a client/server model and divisible problems in a P2P model.

One will notice that, grid computing and a P2P network of computers performing distributed computing is virtually the same thing. Both systems have to satisfy the same core needs to locate resources, request services, access, exchange and collect information remotely.

A surprising evolution of P2P is Skype, the application allowing us to call for free using our computer anywhere around the world. This ground breaking technology has had a tremendous impact on the telecom industry. Today, companies are trying to develop P2P television, but they are facing issues with network bandwidth availability.

To summarize, P2P is the last extremity of a continuum starting from the mainframe-terminal concept and going forward to the client-server, multi-tier, SOA and cloud computing concepts.

2.3 P2P architectures

Decentralization is one of the major concept of p2p systems. This includes distributed storage, processing, information sharing and also control information. Based on the degree of decentralization in a p2p system, we can classify them into two categories:

2.3.1 Purely Decentralized

A pure p2p system is a distributed system without any centralized control. In such systems all nodes are equivalent in functionality. In such networks the nodes are named as ”servant” (SERver+cliENT), the term servent represents the capability of the nodes of a peer-to-peer network of acting at the same time as server as well as a client.

Gnutella, Freenet, Chord and CAN are instances of such systems. Pure p2p systems are inherently scalable. Scalability in the system is usually restricted by the amount of centralized operation necessary and such system largely avoid central instances or servers. This kind of

systems are inherently fault-tolerant, since there is no central point of failure and the loss of a peer or even a number of peers can easily be compensated. They also have a greater degree of autonomous control over their data and resources. On the other hand such systems present slow information discovery and there is no guarantee about quality of services. Also because of the lack of a global view at the system level, it is difficult to predict the system behavior.

2.3.2 *Hybrid Architecture*

In hybrid P2P systems, there is a central server that maintains directories of information about registered users to the network, in the form of meta-data. The end-to-end interaction (data exchange) is between two peer clients. There are two kinds of hybrid systems: centralized indexing and decentralized indexing. In centralized indexing (see Figure 1) a central server maintains an index of the data or files that are currently being shared by active peers. Each peer maintains a connection to the central server, through which the queries are sent. This architecture is used by Napster. Such systems with the central server are simple and they operate quickly and efficiently for discovery information. Searches are comprehensive and they can provide guarantee in searches. On the other hand they are vulnerable to censorship and malicious attack. Because of central servers they have a single point of failure. They are not inherently scalable, because of limitations on the size of the database and its capacity to respond to queries. As central directories are not always updated, they have to be refreshed periodically.

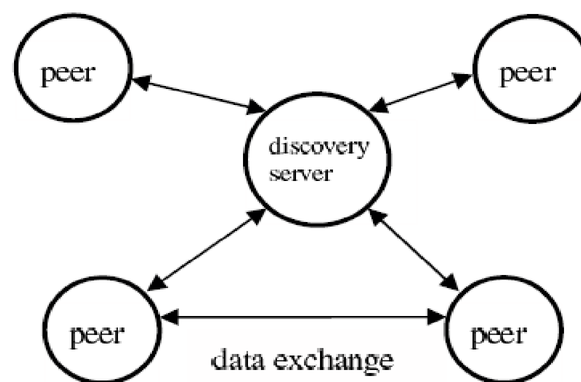

Figure 1. Centralized Indexing

In decentralized indexing (see Figure 2), a central server registers the users to the system and facilitates the peer discovery process. In these systems some of the nodes assume a more important role than the rest of nodes. They are called ”supernodes”. These nodes maintain the central indexes for the information shared by local peers connected to them and proxy search requests on behalf of these peers. Queries are therefore sent to SuperNodes, not to other peers. Kazaa and Morpheus are two similar decentralized indexing systems. In such systems peers are automatically elected to become SuperNodes if they have sufficient bandwidth and processing power and a central server provides new peers with a list of one or more SuperNodes with which they can connect.

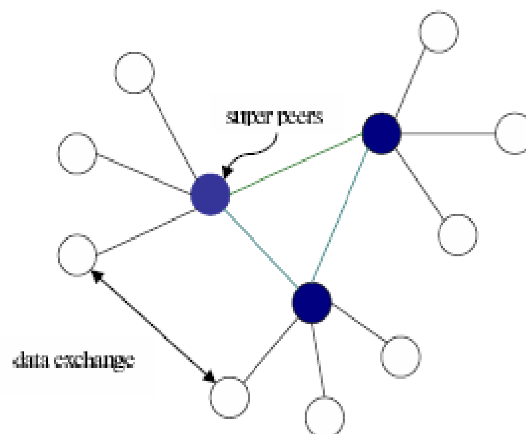

Figure 2. Distributed Indexing

More recent architectures, such as Gnutella also uses the concept of Super Nodes. As a node with enough CPU power joins the network, it immediately becomes a Super-Peer and establishes connections with other SuperPeers, forming a flat unstructured network of SuperPeers. It also sets the number of clients required for it to remain a SuperPeer. If it receives at least the required number of connections to client nodes within a specified time, it remains a SuperPeer. Otherwise it turns into a regular client node. If no SuperPeer is available, it tries to become a SuperPeer again for another probation period.

In comparison with purely decentralized systems, they reduce the discovery time and also they reduce the traffic on messages exchanging between nodes. In comparison with centralized indexing, they reduce the workload on central server but they present slower information discovery. Also in this kind of systems, there is still no unique point of failure as on single central sever. If one or more supernodes go down, the nodes connected to them can open new connection with others, and the network will continue to operate. In the case a large number or even all supernodes go down, the existing peers become supernodes themselves.

2.4 Discovery mechanisms for P2P systems

Distributed peer-to-peer systems often require a discovery mechanism to locate specific data within the system. P2P systems have evolved from first generation centralized structures to second generation flooding-based and then third generation systems based on distributed hash tables [26]:

2.4.1 Centralized indexes and repositories

This mechanism is used in hybrid systems. In this model the peers of the community connect to a centralized directory servers, which store all information regarding location and usage of resources. Upon request from a peer, the central index will match the request with the best peer in its directory that matches the request. The best peer could be the one that is cheapest, fastest, nearest, or most available, depending on the user needs. Then the data exchange will occur directly between the two peers. Napster uses this method. A central directory server maintains: an index with meta data (file name, time of creation etc.) of all files in the network, a table of registered user connection information (IP addresses, connection speeds etc.), a table listing the files that each user holds and shares in the network. In the beginning the client contacts the

central server and reports a list with the files it maintains. When server receives a query from a user, it searches for matches in its index, returning a list of users that hold the matching file. The user then opens a direct connection with the peer that holds the requested file, and downloads it.

2.4.2 Flooding broadcast of queries

This model is a pure p2p model in which each peer does not maintain any central directory and each peer publishes information about the shared contents in the P2P network. Since no single peer knows about all resources, peers in need for resources flood an overlay network queries to discover a resource, each request from a peer is flooded (broadcasted) to directly connected peers, which themselves flood their peers etc., until the request is answered or a maximum number of flooding steps occur. Flooding based search networks are built in an ad hoc manner, without restricting a priori which nodes can connect or what types of information they can exchange. Different broadcast policies have been implemented to improve search in P2P networks [27][28][29]. Original architecture of Gnutella uses the flooding broadcast to find the files in the network. It works as a distributed file storage system. There is four types of messages in the Gnutella protocol: Ping: a request for a certain host to announce itself. Pong: reply to a Ping message. It contains the IP and port of the responding host and number and size of files shared. Query: a search request. It contains a search string and the minimum speed requirements of the responding host. Query hits: reply to a Query message. It contains the IP and port and speed of the responding host, the number of matching files found and their indexed result set.After joining the Gnutella network(by using hosts such as gnutellahosts.com), a node sends out a Ping message to any node it is connected to. The nodes send back a Pong message identifying themselves, and also propagate the ping to their neighbors. Gnutella originally uses TTL-limited flooding (or broadcast) to distribute Ping and Query messages. At each hop the value of the field time-to-live(TTL) is decremented, and when it reaches zero the message is dropped. In order to avoid loops, the nodes use the unique message identifiers to detect and drop duplicate messages. This approach improves efficiency and preserve network band width. Once a node receives a QuerryHit message, indicating that the target file has been identified at a certain node, it initiates a direct out-of-network download, establishing a direct connection between the source and target node.

Although the flooding protocol might give optimal results in a network with a small to average number of peers, it does not scale well. Furthermore, accurate discovery of peers is not guaranteed in flooding mechanisms. Also TTL effectively segments the Gnutella network into subsets, imposing on each user a virtual horizon beyond which their messages cannot reach. If on the other hand the TTL is removed, the network would be swamped with requests.

2.4.3 Routing Model

The routing model adds structure to the way information about resources are stored using distributed hash tables. This protocol provide a mapping between the resource identifier and location, in the form of a distributed routing table, so that queries can be efficiently routed to the node with the desired resource. This protocol reduces the number of p2p hops that must be taken to locate a resource. The look-up service is implemented by organizing the peers in a structured overlay network, and routing a message through the overlay to the responsible peer. Several proposals have been recently put forth for implementing distributed P2P look-up services :

Freenet

Freenet [30] provides file-storage service rather than filesharing service. In this system each peer from the network is assigned a random ID and each peer also knows a given number of peers.

When a document is shared on such a system, an ID is assigned to the document based on a hash of the document's contents and its name. Each peer will then route the document towards the peer with the ID that is most similar to the document ID. This process is repeated until the nearest peer ID is the current peer's ID. Each routing operation also ensures that a local copy of the document is kept. When a peer requests the document from the p2p system, the request will go to the peer with the ID most similar to the document ID. This process is repeated until a copy of the document is found. Then the document is transferred back to the request originator, while each peer participating the routing will keep a local copy.

Chord

Chord [33] is a decentralized p2p lookup protocol that stores key/value pairs for distributed data items. Given a key, it maps key a node responsible for storing the key's value. In the steady state, in an N-node network, each node maintains routing information about $O(logN)$ other nodes, and resolves all lookups via $O(logN)$ messages to other nodes. Updates to the routing information for nodes leaving and joining require only $O(log^2N)$ messages.

Content Addressable Networks

CAN [31] is a mesh of N nodes in virtual d-dimensional dynamically partitioned coordinate space. Each peer keeps track of its neighbors in each dimension. When a new peer joins the network, it randomly chooses a point in the identifier space and contacts the peer currently responsible for that point. The contacted peer splits the entire space for which it is responsible into two pieces and transfers responsibility of half to the new peer. The new peer also contacts all of the neighbors to update their routing entities. The CAN discovery mechanism consists of two core operations namely, a local hash-based look-up of a pointer to a resource, and routing the look-up request to the pointer. The CAN algorithm guarantees deterministic discovery of an existing resource in $O(N^{1/d})$ steps.

Pastry

An approach similar to Cord was also used in Pasty [32]. In the Pastry each node network has a unique identifier (nodId) from a 128-bit circular index space. The pastry node routes a message to the node with a nodeId that is numerically closest to the key contained in the message, from its routing table of $O(logN)$, where N is the number of active Pastry nodes. The expected of routing steps is $O(logN)$. Pastry takes into account network locality; it seeks to minimizes the distance messages travel, according to a scalar proximity metric like the number of IP routing hops.

2.5 P2P networks structure

P2P networks can be classified by the degree to which these overlay networks contain some structure or are created ad-hoc. Structure refer to the way in which the content of the network is located with respect to the network topology. In structured networks, the topology is tightly controlled and the data are placed at specific locations. These systems provide a mapping between the data identifier and location, in the form of a distributed routing table, so that queries can be efficiently routed to the node with the desired data. In unstructured networks, the placement of the data is completely unrelated to the overlay topology and peers are connected directly to each other. They are refereed to as neighbors and have no information of each others data. In these systems, searching amounts to random search, in which various nodes are probed and asked if they have any match for the query. For instance, Gnutella is unstructured and Freenet, Chord and CAN are structured.

In many ways, the quality of a P2P system depends on the structural and behavioral properties of

its network. Unstructured systems are easy to implement and also they require little maintenance but they lack scalability. As the number of participant peers increases, the number of messages exchanged for a resource search grows. Flooding search protocol used in unstructured P2P networks is very sensitive to the number of edges in the network graph. If the number of links is to small, all nodes will not be reachable in a reasonable amount of time. Conversely, if there are too many links, numerous identical copies of the query message will arrive at many nodes from different directions, resulting in wasted bandwidth. In structured P2P systems peers maintain information about what resources neighboring peers offer. It increases the cost of maintenance efforts during changes in the overlay network when peers join or leave.

2.6 Benefits

The common benefits associated with P2P applications are:

Tapping into resources at the edge of the Internet. Instead of relying on a central server to perform many operations, P2P attempts to maximize the utilization of resources of client PCs (memory, processing power and storage capacities) instead.

Reduced network traffic. If more work is performed at the edge of the Internet or if resources are distributed between nodes, then there will be less traffic and network congestion around servers. However, we have seen that if the search mechanism for resources across peers is not well implemented, it can generate a lot of overhead traffic.

Cost savings. If work can be done by peers, then there is no need to buy a server to do it. Therefore, one can save money in material and maintenance.

Faster information delivery. We have seen that high volumes of data can be generated by downloading data parts from multiple peers simultaneously. This is more efficient than acquiring a bigger bandwidth between two entities where only one end is transmitting data.

Scalability . If extra processing power for a P2P application is needed, one just needs to add extra nodes which is easier than installing another server. This can be useful for divisible problems.

Self-organization. Nodes arrive and depart at frequent intervals in P2P systems. Despite this chaotic activity, P2P systems can re-organize themselves automatically.

Network fault tolerance. If one peer goes down the network is still alive, another one can take over. If a server is down that is not (always) true.

Pervasiveness. This is the capacity to reconnect with peers and services that have changed location on the Internet. It allows users to behave like nomads on the Internet.

2.7 Drawbacks

The common drawbacks associated with P2P applications are:

Non-deterministic services. Since peers connect and disconnect to the network more often than servers, there is a higher risk of resource or service unavailability. However, if the resources and

services are duplicated on another peer, this problem can be mitigated. If two peers request the same service or resources on a P2P network, they may obtain it from different peers via different routes, with different bandwidths, resulting in different service quality.

Content ownership infringement. Early P2P allowed fast distribution of any content, including copyright protected content. However, some control on who-exchanges-what-with-whom can be implemented now within P2P applications, limiting the massive illegal propagation of resources. Sharing content or resources is a matter of trust.

Absence of central control. The fact that P2P applications allow the exchange of direct information from one peer to another means that improper content can be transferred too. The flip side of this argument is that a complete central server-like type of control would still not prevent users from exchanging this kind of information. They would simply do it by other means. There is a fundamental conceptual flaw with the idea that control needs to be central. Control is about trust. Whom do you trust? Why? And what privileges do you grant to that person or entity?

3 Analysis of available P2P technologies

A number of tools and platforms that had considerable momentum in the previous decade are not being actively developed anymore and thus the real choice between JXTA and other technologies comes down to JXTA vs. a custom tailored solution using a combination of established technologies.

3.1 JXTA

JXTA is a set of open, generalized peer-to-peer (P2P) protocols that allow any networked device — sensors, cell phones, PDAs, laptops, workstations, servers and supercomputers — to communicate and collaborate mutually as peers. The JXTA protocols are programming language independent, and multiple implementations, also known as bindings, exist for different environments. Their common use of the JXTA protocols means that they are all fully interoperable.

JXTA, pronounced 'juxta', originates from the word juxtapose, which means to place something side by side. JXTA reflects the operations by which peers establish temporary associations to form a P2P network; they juxtapose themselves to each other. JXTA is not a software design philosophy and it is not a software application. It is a set of protocols that software developers can implement using their preferred technology to establish P2P connections with other peers using identical technologies or different implementations of JXTA.

For example, a group of developers can implement JXTA in Visual Basic under Windows XP. Another group could do the same in C++ under Linux and a third group could implement the JXTA layer on a hand-held device in Java. They would all be able to find each other on the Internet and to start exchanging any kind of information or services between them, despite the use of different underlying technologies.

A primary design principal of JXTA is to provide a platform that embodies the basic P2P network functions. As such, JXTA overcomes potential shortcomings of many existing P2P systems:

- *Interoperability* — JXTA technology is designed to enable peers provisioning P2P services to locate and communicate with one another independent of network

addressing and physical protocols.

- *Platform independence* — JXTA technology is designed to be independent of programming languages, network transport protocols, and deployment platforms.
- *Ubiquity* — JXTA technology is designed to be accessible by any device with a digital heartbeat, not just PCs or a specific deployment platform.

One common characteristic of peers in a P2P network is that they often exist on the edge of the regular network, the edge often being occasionally connected devices that are assigned non static addresses (e.g. DHCP). Because they are subject to unpredictable connectivity with potentially variable network addresses, they are outside the standard scope of DNS. JXTA empowers peers on the edge of the network by provisioning a globally unique peer addressing scheme that is independent of traditional name services. Through the use of JXTA IDs, a peer can migrate across physical networks, changing transports and network addresses, even being temporarily disconnected, and still be addressable by other peers.

The JXTA protocols are designed to be independent of transport protocols and make few assumptions about network transportation mechanisms between computers and electronic devices. In other words, JXTA does not take the responsibility of explaining how messages should physically be exchanged between peers or from a technical point-of-view.

JXTA imposes a common structured language to issue and exchange messages between peers:XML (Extensible Markup Language). Although this language is readable by human beings, which is a benefit, its verbosity has often been pointed as a weakness regarding application performance. XML documents are usually bigger than traditional binary data documents containing the same amount of information. This issue can be me mitigated by the use of data compression within XML documents.

The fact that JXTA defines its protocols independently from any other technologies and that it has chosen a neutral technology to communicate messages between peers implementing its protocols guarantees its universality. Of course, its implementation on specific platforms and the choice of a network transportation layer between peers creates specific technical issues which have to be solved by each implementation of JXTA locally. This preserves its universality.

Conceptually, JXTA is consists of three logical layers:

1. ***Platform.*** This layer is the base of JXTA and contains the implementation of the minimal and essential functionalities required to perform P2P networking. Ideally, JXTA-enabled peers will implement all JXTA functionalities, although they are not required to. This layer is also known as the core layer.
2. ***Services.*** This layer contains additional services that are not absolutely necessary for a P2P system to operate, but which might be useful. For example: file sharing, PKI infra-structures, distributed files systems, etc... These services are not part of the set of services defined by JXTA.
3. ***Applications.*** P2P applications are built on top of the service layer. However, if I develop a file sharing application and let other JXTA based applications make requests to my application, the other applications will perceive me as a service. Therefore, the border between a service and an application depends on one's perspective.

JXSE is the open Source Java implementation of the JXTA protocols standard edition.

3.2 Microsoft Windows P2P

Microsoft has not been oblivious to the emergence of P2P, and has been developing its own tools and technologies to use it. You can use the Microsoft Windows Peer - to - Peer Networking platform as a communication framework for P2P applications. This platform includes the important components Peer Name Resolution Protocol (PNRP) and People Near Me (PNM). Also, version 3.5 of the .NET Framework introduced a new namespace, System.Net.PeerToPeer , and several new types and features that you can use to build P2P applications yourself with minimal effort.

Microsoft .NET framework can be used in the development of P2P systems. Features include:

- All of the P2P code is based on the .NET framework.
- Messages sent between peers is serialized as XML.
- Objects can be shared and accessed by peers.
- A discovery service has been implemented using .NET.

There are Microsoft resources showing the peer discovery as well as a simple chat application. Of course, the focus of the Microsoft initiative are web services, and several examples illustrate building services that peers can use .

The Microsoft Windows Peer - to - Peer Networking platform is Microsoft ' s implementation of P2P technology. It is part of Windows XP SP2, Windows Vista, and Windows 7, and is also available as an add - on for Windows XP SP1. It includes two technologies that you can use when creating .NET P2P applications:

- The Peer Name Resolution Protocol (PNRP), which is used to publish and resolve peer addresses
- The People Near Me server, which is used to locate local peers (currently for Vista and Windows 7 only)

You can of course use any protocol at your disposal to implement a P2P application, but if you are working in a Microsoft Windows environment it makes sense to at least consider PNRP. There have been two versions of PNRP released to date. PNRP version 1 was included in Windows XP SP2, Windows XP Professional x64 Edition, and Windows XP SP1 with the Advanced Networking Pack for Windows XP. PNRP version 2 was released with Windows Vista, and was made available to Windows XP SP2 users through a separate download (see KB920342 at support.microsoft.com/kb/920342). Windows 7 also uses version 2. Version 1 and version 2 of PNRP are not compatible, and this chapter covers only version 2.

In itself, PNRP doesn' t give you everything you need to create a P2P application. Rather, it is one of the underlying technologies that you use to resolve peer addresses. PNRP enables a client to register an endpoint (known as a *peer name*) that is automatically circulated among peers in a cloud. This peer name is encapsulated in a PNRP ID. A peer that discovers the PNRP ID is able to use PNRP to resolve it to the actual peer name, and can then communicate directly with the associated client.

PNRP IDs are 256 - bit identifiers. The low - order 128 bits are used to uniquely identify a particular peer, and the high - order 128 bits identify a peer name. The high - order 128 bits are a hashed combination of a hashed public key from the publishing peer and a string of up to 149 characters that identifies the peer name. The hashed public key (known as the *authority*) combined with this string (the *classifier*) are together referred to as the P2P ID. It is also possible to use a value of 0 instead of a hashed public key, in which case the peer name is said to be *unsecured* (as opposed to *secured* peer names, which use a public key).

The PNRP service on a peer is responsible for maintaining a list of PNRP IDs, including the ones that it publishes as well as a cached list of those it has obtained by PNRP service instances elsewhere in the cloud. When a peer attempts to resolve a PNRP ID, the PNRP service either uses a cached copy of the endpoint to resolve the peer that published the PNRP or it asks its neighbors if they can resolve it. Eventually a connection to the publishing peer is made and the PNRP service can resolve the PNRP ID.

Note that all this happens without you having to intervene in any way. All you have to do is ensure that peers know what to do with peer names after they have resolved them using their local PNRP service.

Peers can use PNRP to locate PNRP IDs that match a particular P2P ID. You can use this to implement a very basic form of discovery for unsecured peer names. This is because if several peers expose an unsecured peer name that uses the same classifier, the P2P ID will be the same. Of course, because any peer can use an unsecured peer name you have no guarantee that the endpoint you connect to will be the sort of endpoint you expect, so this is only really a viable solution for discovery over a local network.

A cloud is maintained by a *seed server,* which can be any server running the PNRP service that maintains a record of at least one peer. Two types of clouds are available to the PNRP service:

- **Link local** — These clouds consist of the computers attached to a local network. A PC may be connected to more than one link local cloud if it has multiple network adapters.
- **Global** — This cloud consists of computers connected to the Internet by default, although it is also possible to define a private global cloud. The difference is that Microsoft maintains the seed server for the global Internet cloud, whereas if you define a private global cloud you must use your own seed server. If you use your own seed server you must ensure that all peers connect to it by configuring policy settings.

With Windows 7, PNRP makes use of a new component called the *Distributed Routing Table (DRT).* This component is responsible for determining the structure of the keys used by PNRP, the default implementation of which is the PNRP ID previously described. By using the DRT API it is possible to define an alternative key scheme, but the keys must be 256 - bit integer values (just like PNRP IDs). This means that you can use any scheme you want, but you are then responsible for the generation and security of the keys. By using this component you can create new cloud topologies beyond the scope of PNRP, and indeed, beyond the scope of this chapter as this is an advanced technique.

Windows 7 also introduces a new way of connecting to other users for the Remote Assistance application: Easy Connect. This connection option uses PNRP to locate users to connect to. Once a session is created, through Easy Connect or by other means (for example an e - mail invitation), users can share their desktops and assist each other through the Remote Assistance interface.

3.3 The Peer-to-Peer Trusted Library

One of the well-known libraries is called the Peer-to-Peer Trusted Library (PtPTL) [3][4]. This library is open source, and its goal is to provide innovation in the security arena as it relates to P2P systems.

P2P has been described as “an anarchistic threat to the current Internet” (David Streitfeld, *The Washington Post*, July 18, 2000), and Marc Andreesen has called P2P software a “benevolent

virus.”

The potential security concerns for P2P software can be categorized as follows:
- · Reputation - copyright infringement
- · Denial of Service - bandwidth and storage consumption
- · Security Holes
- · Confidentiality - file sharing
- · Malware - trojan horse and virus distribution
- · Information Gathering - disclosure of IP and MAC addresses, connection speed

The PtPTL is designed to provide the following:
- Digital certificates
- Peer authentication
- Secure storage
- Public-key encryption
- Digital signatures
- Digital envelopes
- Symmetric-key encryption

PtPTL conforms to, and includes support for, the following standards:
- X.509 digital signatures
- PKCS#1 (RSA cryptography)
- PKCS#5 (password-based cryptography)
- PKCS#7 (digital envelopes)
- PKCS#12 (personal information exchange)
- RFC 1421 (privacy enhanced mail format)
- Various standard symmetric encryption algorithms
- HTTP

The code is designed to execute on both the Windows and Linux operating systems. Numerous examples are provided, and full API documentation is available. As a replacement to secure communication using SSL, the PtPTL provides support for more than just client-server network topologies. Note that PtPTL is not a P2P system or toolkit—it is designed to add trust to a P2P system.

3.4 JINI

Jini technology [5] is a service oriented architecture that defines a programming model which both exploits and extends Java technology to enable the construction of secure, distributed systems consisting of federations of well-behaved network services and clients. Jini technology can be used to build adaptive network systems that are scalable, evolvable and flexible as typically required in dynamic computing environments. Jini offers a number of powerful capabilities such as service discovery and mobile code.

The term Jini refers to both a set of specifications and an implementation; the latter is referred to as the Jini Starter Kit. Both the specifications and the Starter Kit have been released under the Apache 2.0 license and have been offered to the Apache Software Foundation's Incubator.

Jini provides facilities for dealing with some of the fallacies of distributed computing, problems of system evolution, resilience, security and the dynamic assembly of service components. Code

mobility is a core concept of the platform and provides many benefits including non-protocol dependence.

One of the goals of Jini is to shift the emphasis of computing away from the traditional disk-drive oriented approach, to a more network oriented approach. Thus resources can be used across a network as if they were available locally. Jini is based on Java, and is similar to Java Remote Method Invocation but more advanced. Jini allows more advanced searching for services, through a process of discovery of published services (making Jini akin to the service-oriented architecture concept).

There are three main parts to a Jini scenario. These are the client, the server, and the lookup service.

The service is the resource which is to be made available in the distributed environment. This can include physical devices (such as printers or disk drives) and software services (for example a database query or message service). The client is the entity which uses the service.

JXTA effectively abstracts the network allowing for P2P, Jini, on the other hand is a federation of distributed software components, which sometimes do communicate in a P2P fashion (and sometimes do NOT). To the casual observer, both have some overlapping functions, but this is not actually the case. For example, both Jini and JXTA have lookup facilities but in Jini one is looking for an Java Interface (think of this as looking up for a function, a service) while in JXTA one looks for a peer by some name or inside a group. The JXTA type of lookup is more like trying to find an IP based on a name using a DNS, since JXTA abstracts the network this is a more than required functionality.

So JXTA connects the peers by allowing them to find each other and providing communications channels called pipes, all this is done with a protocol based on XML messages and **it can cross firewalls easily**. Over these pipes one can communicate using the protocol one desires, and peers can be running any programing language on any platform.
In conclusion JXTA is a powerful communication tool, and is very useful in that it allows crossing of firewalls.

3.5 Enterprise service bus (ESB) - Mule

An **enterprise service bus** (ESB) is a software architecture model used for designing and implementing the interaction and communication between mutually interacting software applications in Service Oriented Architecture. As a software architecture model for distributed computing it is a speciality variant of the more general client server software architecture model and promotes strictly asynchronous message oriented design for communication and interaction between applications. Its primary use is in Enterprise Application Integration of heterogenous and complex landscapes.

An ESB transports the design concept of modern operating systems to networks of disparate and independent computers. Like concurrent operating systems an ESB caters for commonly needed commodity services in addition to adoption, translation and routing of a client request to the appropriate answering service.
The prime duties of an ESB are:

- Monitor and control routing of message exchange between services
- Resolve contention between communicating service components

- Control deployment and versioning of services
- Marshal use of redundant services
- Cater for commonly needed commodity services like event handling and event choreography, data transformation and mapping, message and event queuing and sequencing, security or exception handling, protocol conversion and enforcing proper quality of communication services

An ESB generally provides an abstraction layer on top of an implementation of an enterprise messaging system. In order for an integration broker to be considered a true ESB, it would need to have its base functions broken up into their constituent and atomic parts. The atomic components would then be capable of being separately deployed across the bus while working together in harmony as necessary.

ESB is a modular and component based architecture. It assumes that services are generally autonomous and availability of a service at a certain moment of time cannot be guaranteed. Therefore messages need to be routed consequently through the message bus for buffering (message queuing to allow inspection and enhancement of content as well as filtering, correction and rerouting of message flow.

Mule is a lightweight ESB and integration framework. It can handle services and applications using disparate transport and messaging technologies. The platform is Java-based, but can broker interactions between other platforms such as .NET using web services or sockets. The architecture is a scalable, highly-distributable object broker that can seamlessly handle interactions across legacy systems, in-house applications and almost all modern transports and protocols.

On the other hand, JXTA is open source and sents text/binary messages or streams over unreliable disparate networks to peers belonging to specific groups. Strictly speaking, JXTA is a set of open XML-based protocols used to create logical networks on top of physical networks. JXTA has several bindings that implement these open XML protocols to accommodate different platforms, such as jxta-jxse, jxta-c and jxta-jxme. Mule is used to send, transform and route text/binary/POJOs to one or many endpoints on unreliable disparate networks. Both Mule and JXTA have overlapping features such as providing network independence or a platform to abstract the network from the application. They also provide communication and services to a group of authorized peers over secure channels and the ability to emulate various network topologies: P2P, client/server, service buses.

The most obvious difference between the two technologies is that Mule is a "high-level" technology designed for quick implementation to solve many distributed application problems like having two legacy applications communicate over a network with each other in different data formats or protocols. JXTA, in comparison, is a "low-level" platform that provides a set of its own networking protocols that the developer must implement to communicate with its remote peers. As you might expect, JXTA has a higher learning curve and has more parts to manage than Mule does, but as you also might expect, JXTA provides finer control over the network semantics and can reach peers through more firewall and NAT configurations than Mule

3.6 Unmanaged Internet Architecture (UIA)

The Internet's architecture, designed in the days of large, stationary computers tended by technically savy and accountable administrators, fails to meet the demands of the emerging ubiquitous computing era. Nontechnical users now routinely own multiple personal devices, many

of them mobile, and need to share information securely among them using interactive, delay-sensitive applications.

Unmanaged Internet Architecture (UIA) is a novel, incrementally deployable network architecture for modern personal devices, which reconsiders three architectural cornerstones: naming, routing, and transport. UIA augments the Internet's global name system with a *personal name system*, enabling a user to build personal administrative groups easily and intuitively, to establish secure bindings between his and other users' devices, and to name his devices and his friends much like using a cell phone's address book. To connect personal devices reliably, even while mobile, behind NATs or firewalls, or connected via isolated ad hoc networks, UIA gives each device a persistent, location-independent *identity*, and builds an *overlay routing service* atop IP to resolve and route among these identities. Finally, to support today's interactive applications built using concurrent transactions and delay-sensitive media streams, UIA introduces a new *structured stream* transport abstraction, which solves the efficiency and responsiveness problems of TCP streams and the functionality limitations of UDP datagrams.

UIA is a distributed name system and ad-hoc routing infrastructure which provides zero-configuration connectivity among users' mobile devices without the use of centralized servers. Each user has a local namespace which is shared among all devices and is always available on every device. Users can assign personal names to each of their devices, and can also name other users and access their friends' namespaces. UIA devices automatically maintain connectivity with other named devices, both in ad-hoc networks and **globally** on the Internet when available.

UIA [1] provides strong permanent location independent device identifiers, and allows users to securely bind personal names to devices. Each device creates a unique public/private keypair, and hashes the public key to create an endpoint identifier (EID), which acts as the permanent device address. UIA constructs an overlay network and offers a traditional socket API to establish connections. The UIA router forwards connections over the authenticated and encrypted overlay network to the destination. UIA's routing overlay supports IP mobility along with seamless operation though NATs and most firewalls.

3.7 MACEDON and Mace

MACEDON is an infrastructure to simplify the design, development, evaluation, and comparison of large-scale overlays. In MACEDON, researchers specify algorithm behavior in terms of event-driven finite state machines (FSMs) consisting of system states, events (e.g. message reception, remote node failure, etc.), and transitions indicating the actions to take in response to events. From this high level specication, MACEDON generates code for a variety of experimentation infrastructures leveraging shared (but extensible) libraries. The libraries implement much of the base overlay maintenance functionality, such as thread and timer management, network communication, debugging, and state serialization. As such, improvements in system support can be equally applied to all protocols. Ultimately, these system mechanisms enable fair comparisons of the merits of individual algorithms.

MACEDON [2][6] and Mace [7] are overlay construction software which support multiple routing algorithms. A user describes an algorithm in MACEDON language, which is like C/C++ but specific to the overlay description. MACEDON translates the description into executable C++ code. The generated code communicates using TCP or UDP. MACEDON provides distributed hash table (DHT) implementations, i.e., Chord and Pastry. MACEDON introduces a domain-specific language and thus, involves a higher learning cost for dedicated language.

Experiments with MACEDON have been performed on an Internet emulator, ModelNet, where the number of underlying computers ranged from 2 to 50 in the emulation. The original length of IDs in Chord is 160 bits and 128 bits in Pastry, but both are 32 bits in MACEDON. The integer type int the dedicated language provides is 32 bits and the shortened ID length might be a natural consequence of this. Mace [7] is a successive project following MACEDON.

Overlay algorithms typically target specific types of applications. An important characteristic of their implementation is the API they export. For example, a multicast overlay must export a send function to disseminate data through the overlay.

A standard API enables MACEDON applications to select underlying overlays without modication. In general, overlays support multicast or route primitives that route data from a source to destination(s) through the overlay. Typically, overlays provide upcalls at each routing hop so that intermediate nodes can perform application-specific functionality. For example, an intermediate Scribe node receiving a join request for a group will add the group to its list of multicast sessions and propagate the request toward the destination, thus building a reverse-path distribution tree.

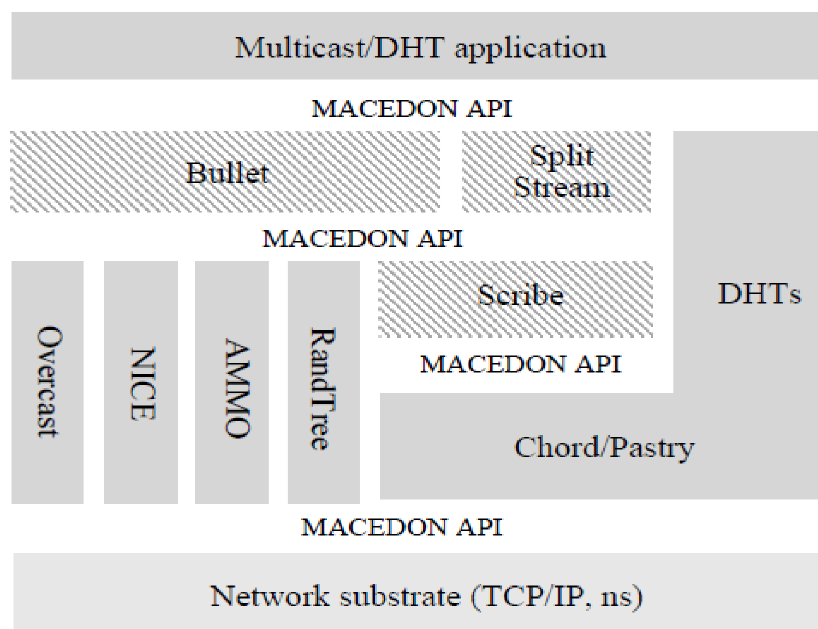

Figure 3. The MACEDON protocol stack

Protocol layering (Figure 3) is central to implementing algorithms in MACEDON. The MACEDON protocol stack is divided into three components: application, multiple protocol layers, and network substrate (ns orTCP/IP). Much like the TCP/IP stack, higher layers in MACEDON use the services of lower layers. Bullet, for example, uses a simple randomly constructed tree, RandTree, for baseline data distribution.

```
typedef int (*macedon_forward_handler)
   (char *msg, int size, int type,
    int nextHop, macedon_key nextHopKey);
typedef void (*macedon_deliver_handler)
   (char *msg, int size, int type);
typedef void (*macedon_notify_handler)
   (int type, int size, int *neighbors);
typedef int (*macedon_upcall_handler)
   (int operation, void *arg);
macedon_init(macedon_key bootstrap, int prot);
void macedon_register_handlers(
  macedon_forward_handler,macedon_deliver_handler,
  macedon_notify_handler,macedon_upcall_handler);
int macedon_create_group(macedon_key groupID);
void macedon_join(macedon_key groupID);
void macedon_leave(macedon_key groupID);
int macedon_route(macedon_key dest, char *msg,
     int size, int priority);
int macedon_multicast(macedon_key groupID,
     char *msg, int size, int priority);
int macedon_anycast(macedon_key groupID,
     char *msg, int size, int priority);
int macedon_routeIP(int dest, char *msg,
     int size, int priority);
```

Figure 4. MACEDON API

Figure 4 illustrates a simplied version of the API that MACEDON overlays export. We provide an extensible upcall and downcall mechanism to perform protocol-specific collaboration across layers in the stack. As instances of this mechanism, we describe forward(), deliver(), and notify() (extensible upcalls are handled using the generic handler). A node calls forward() once it makes a message routing decision. Intermediate nodes can change the message or its destination or quash the message altogether. The notify() upcall allows lower-layer protocols to inform higher layers of changes in neighbor lists (a higher layer may require this direct knowledge). An application optionally registers its upcall handlers with the macedon_register_handlers() function. At least one handler is necessary if the application is to receive any data through the overlay (having null handlers would be used when evaluating just the construction process of dierent overlays).

Figure 4 also shows macedon_init() that initializes an overlay identied by the application-specied well-known protocol value (akin to protocol values in IP). Once an application initializes and registers its handlers, it can send and receive data. For unicast data, the overlay must implement routing functionality that determines which neighbor receives data packets next. The macedon_route() function accepts a message and destination in the form of a macedon key, meaning it is not necessarily an IP address (it could be a hash of an IP address or name). A similar primitive is macedon_routeIP() that enables native IP-based communication with an IP host.

Multicast primitives include macedon_create_group() to create sessions. Its sole input is the value, or handle, associated with the session (group). Receivers join and leave a session with macedon_join() and macedon_leave(), specifying the group value. Similar to macedon_route(), macedon_multicast() requires a session's ID instead of a node's destination address. macedon_collect() introduces a new primitive to traditional overlay APIs. It essentially performs the opposite of multicast, where data originates at non-root nodes and is collected via the distribution tree toward the root. Intermediate nodes can summarize data in an application-specific manner, ultimately delivering a global summary to the tree's root. We believe that a number of applications could benet from this communication paradigm.

Mace is a software package for building distributed systems. It builds upon the ideas from its parent project, MACEDON, by broadening the scope of what can be designed with it, and by removing many limitations of the original system.

Mace includes a compiler that translates service specifications into *C++ code*, libraries designed to be linked together with generated services, a distribution of existing services ready to be used by other services or applications, and a few basic applications to run the services contained within.

Mace seeks to transform the way distributed systems are built by providing designers with a simple method for writing complex but correct and efficient implementations of distributed systems. To that end, we are always considering new libraries and language features which could be used to make building, designing, debugging, or verifying distributed systems more powerful, flexible, simple, or natural.

Constructing distributed systems would be simplified by the ability to compose simple distributed computing primitives into more complex behavior. For instance, many distributed applications would benefit from failure detection, consensus, multicast, barriers, and key-based routing. However, without well-defined API's, it is difficult to reuse implementations or to leverage the benefits of an improved implementation of a given logical subsystem. In this paper, we describe initial efforts to define required API's to support complex, multi-layer distributed systems.

Current programming languages are not well suited to the requirements of distributed systems. While there are communication libraries and class hierarchies in languages such as C++, Java, and Python, they typically target client/server communications (e.g., HTTP or XMLRPC) and still provide relatively primitive support for failure detection and recovery. Further, we observe that the higher-level structure of many distributed systems is logically event and state-based. Each node maintains some state that may be modified as a result of a series of events, typically message reception and timer expiration. Individual nodes respond to events by modifying their state and perhaps transmitting their own message to one or more destinations. While this high level structure is simple to describe, it is error prone to implement. Further, managing asynchrony still remains a challenge. Delivering high performance often requires careful consideration of appropriate locking primitives, ensuring that individual operations do not block, and assigning the appropriate number of threads to handle logically concurrent tasks. Of course, all of this can be programmed in existing languages such as C++ and, to a lesser extent, Java. Providing the appropriate language primitives can both significantly simplify the code and reduce opportunities for errors.

3.8 Ezel

EZEL is the Easy Entry Library for JXTA. It's goal is to enable a client-server developer, who has little or no JXTA or P1P experience, to create a JXTA service in a single afternoon.

This library provides a client/server-like API that hides implementation details and provides reasonable and functional defaults. Complexity for the uninitiated is greatly reduced by folding away the options. It also encapsulates many typical techniques that are used in everyday JXTA programming (creating and publishing advertisement, searching for advertisements, creating and working with structured documents, working with peergroups, etc.) in several collections of APIs, making it easier for new developers to build JXTA applications.

Ezel seems to be an abandoned project and is no longer supported in the current version of jxta.

3.9 Microsoft Groove

Microsoft Groove is a collaboration suite ideal for small businesses and companies with no single physical base. It has been used by emergency relief agencies and top consultants and is a valuable tool for anyone who needs to work offline or within a disparate community.

At its core Groove is a simple idea: to create a shared workspace allowing users to distribute files and folders across a team. The cache of files that is built up means that members of the team do not have to be online to examine and amend crucial information, thus making the process convenient and hassle free.

Another advantage of Groove is that it bypasses constrictive security clearance issues. Not all consultants working for a particular company will have the same levels of security clearance, which can make communication and interaction difficult. As Groove is a peer-to-peer platform it works without a server, users simply invite others to join the group and when they accept they become part of the workspace.

Once in the virtual workspace all active team members can edit and amend documents in synchronicity. Multiple versions of the initial document appear with the alterations that each team member is making at that particular time. Various tools can be deployed in line with the workspace being utilised, such as a calendar, web browser and sketchpad and there is also access to Microsoft SharePoint's document library.

With team members potentially scattered across the globe and the greater freedom afforded by the peer-to-peer model you could be forgiven for thinking that Groove was lacking in cohesion and structure. However, there is a Microsoft Groove server available to team leaders and those organising workgroups. This server enables centralised control of virtual workspaces within Groove allowing for a focused approach to file sharing and amendment.

In an age of ever increasing globalisation where colleagues are not all sat at office desks during the working day Microsoft Groove is an invaluable program that brings people together quickly and conveniently to achieve a shared business goal. Collaboration software is now all around us with social networking sites like Facebook and Twitter operating on similar principles. Groove harnesses this hugely popular mass community interaction and utilises it for a business purpose.
Yet it would be a mistake to think that just because you can create a group on Facebook you can master Groove just like that. While far from inaccessible Groove is nonetheless a strikingly

different addition to Microsoft Office and a high-quality training course is advised in order to get the most out of its many functions.

3.10 Summary

For the majority of developed P2P network protocols, implementations and API's exist to aid application developers. The complexity of these APIs varies significantly, from simple network-oriented functionality provided by Gnutella [15] APIs such as Jtella [16], and Chord[8] APIs such as Accord[17], to the more complex and application-oriented functionality provided by JXTA [13] and Groove[18]. Each of these APIs require that developers possess detailed understanding of the underlying P2P technology. Due to the widely recognized [10] lack of standardisation within the P2P community, the structure of these APIs varies considerably, to the extent that, it is rarely the case that experience and understanding of one API can be readily applied to another.

To address the lack of standardisation of P2P technologies, recent work has considered building abstractions of underlying P2P technologies to create common interfaces for developers. Notable attempts to provide abstractions of heterogeneous P2P technologies include the Common API for Structured P2P Systems [19], PROST [20] and the Open Overlays project [21].

The Common API for Structured P2P Systems provides a consistent abstraction for structured overlays such as Pastry [22], Past [23] and SplitStream [24]. The Common API for Structured P2P Systems provides three different abstractions, one for each major area of system functionality. These abstractions include: distributed hash table, distributed object location and retrieval, and cast (i.e. multicast and anycast).

Prost provides an abstraction of overlay networks by implementing the previously described common API upon which a supporting infrastructure for pluggable services is layered. The design of PROST is influenced heavily by lower level programmable networking approaches. All applications and services for PROST are written as plug-ins known as peer-lets. PROST also allows these plug-ins to be dynamically deployed, installed and instantiated.

The Open Overlays project provides a common abstraction in which diverse overlay networks may be modelled using a consistent abstraction provided by the 'overlay' component framework. This framework forms a powerful building block which can be used to assemble systems composed of heterogeneous overlays. For example, using the open overlays component framework, any unstructured overlay network could be layered on top of any structured overlay. Open Overlays is implemented using the run-time reconfigurable OpenCOM middleware.

The aforementioned approaches, however, focus on providing support for the P2P network developer rather than the P2P application developer. In contrast Ezel [25] and Groove provide more application-centric APIs. Ezel implements an abstraction of JXTA, reducing the complexity of the standard API by replacing it with a simpler cut-down version. However, while Ezel does reduce JXTA's complexity, it still requires the developer to understand JXTA's core concepts and principles in order to be able to use it (for example, understanding how pipes are used for message communication). Groove is more sophisticated in that it provides an integrated development environment in which P2P applications can be created. Groove provides a higher level of abstraction, removing the need for developers to understand the underlying technology. However, Groove achieves this by constraining what the developer can build, with the primary focus being on groupware applications such as Instant Messengers and shared workspaces. For applications that fall outside this domain, for example, distributed computation, the usefulness of Groove is limited.

4 Platform selection process and criteria

The entire set of the above candidates P2P tools and platform have been evaluated in order to select the most appropriate one as a base technology for P2P. This section presents the criteria used for evaluating them. The employed criteria includes a subset of the functional and non functional requirements of PeerAssist, which are mainly related to networking functions such as connectivity, communication grouping, interoperability, openness, serviceability, security, trust, scalability, efficiency, etc.

4.1 Connectivity, communication, grouping

A P2P system enables entities at the edges of the network to communicate and share services and resources without the need of centralized control. The selected technology will provide or facilitate the following:

R1. The system shall allow users to search for peers.

R2. The system shall allow users to communicate with each other through specific channels.

R3. The system shall allow users to create and participate in groups of users.

R7. The system shall find and propose matching peers to join an open group.

R8. The user must be allowed to select specific peers for a closed, private community.

R9. The users must be allowed to accept or reject group membership invitations.

R10. The users must be allowed to join open (public access) groups, even if they were not initially matched and invited by the system.

R11. The users must be allowed to leave a group at any time.

R14. The system shall allow users to search for groups.

R15. The system shall allow the owner of a group to delete it, in which case notifications are sent to its members.

R59. The system shall support the creation of P2P communities based on semantically retrieved information.

Every client participating in a P2P network application must be able to perform the following operations to overcome these problems:

- It must be able to *discover* other clients.
- It must be able to *connect* to other clients.
- It must be able to *communicate* with other clients.

The discovery problem has two obvious solutions. You can either keep a list of the clients on the server so clients can obtain this list and contact other clients (known as *peers*), or you can use an infrastructure (for example PNRP) that enables clients to find each other directly.

The connection problem is a more subtle one, and concerns the overall structure of the networks used by a P2P application. If there exists one group of clients, all of which can communicate with one another, the topology of the connections between these clients can become extremely complex. Performance can often be improved by having more than one group of clients, each of which consists of connections between clients in that group, but not to clients in other groups. If these groups can be made locale - based one will get an additional performance boost, because clients can communicate with each other with fewer hops between networked computers.

Communication is perhaps a problem of lesser importance, because communication protocols such as TCP/IP are well established and can be reused here. Discovery, connection, and communication are central to any P2P implementation.

Groups of peers that are connected to each other are known by the interchangeable terms *meshes, clouds,* or *graphs.* A given group can be said to be *well - connected* if at least one of the following statements applies:

- There is a connection path between every pair of peers, so that every peer can connect to any other peer as required.
- There are a relatively small number of connections to traverse between any pair of peers.
- Removing a peer will not prevent other peers from connecting to each other.

4.2 Service, Interoperability, Openness and Extensibility

A modular P2P overlay architecture will be built that resides between the network and the service layer. The P2P layer will be responsible for the transparent and efficient communication of the SOAP messages described in each of the services. This network overlay will provide efficient routing and the formation and maintenance of virtual communities.

R13. The communities shall provide facilities to enable interaction between users: communication channels, data sharing, etc.

R16. The system shall allow users or 3rd parties to publish services.

R17. The system shall allow users or 3rd parties to advertise services.

R18. The system shall allow users to search for services.

R19. The system shall allow users to use (i.e. book) a service.

R20. The system shall allow users to rate a service.

R21. The system shall allow users to search content in the platform.

R22. The system shall allow users to publish content in the platform.

R23. The system shall provide users with content suggestions.

R24. The system shall allow users to advertise items (events, communities…).

R25. The system shall allow users to receive advertisements based on filtering criteria.

R30. The system shall allow the user to edit the data on his/her profile.

R31. The system shall allow the user to add, delete or modify peers on his/her contact list.

R32. The system shall help the user to perform tasks through a Personal Assistant.

R60. The system shall provide remote service discovery and management. Service management and discovery will be independent of the network layer.

R61. The system shall provide identity management.

R65. The P2P layer shall provide an application agnostic overlay.

R66. It should be a tailored overlay to the needs raised by the services and applications running on top.

R67. It will be implemented in the service layer using technologies already in place through the Service Oriented Architecture (SOA). Through this mechanism services can be implemented in an efficient cross platform manner that does not rely on the underlying network infrastructure.

4.3 System architecture

The P2P layer is a distributed system architecture paradigm that will provide all desired system characteristics. P2P networks are typically used for largely connecting nodes via ad-hoc connections. The system that will be built, will be secure, scalable and efficient and it will support a wide range of end-user devices as described in the following sections.

R34. The system shall be reliable.

4.4 Efficiency and scalability

The P2P architecture, and the jxta 2 network topology (a smaller population of rendezvous peer among the edge peers of the system), reduce the network traffic and makes the system work in a high performance and scalable manner.

R33. The system's latency shall be within acceptable limits.

R64. The P2P overlay network shall be scalable, decentralized, extensible and flexible.

4.5 Security and trust

JXTA implements TLS version 1.0 for transportation of messages between peer endpoints of the JXTA network. This model is a clear message over communications channel mode. JXSE guarantees that if a message between two peers has to be transmitted via a relay peer or via

other peers, these will not have access to the content of the message. JXTA security allows advertisements and messages to be signed when stored or communicated between peers.

R6. All types of groups can be open (free access) or closed (only selected users are allowed). However, the system may impose access restrictions in specific cases.

R35. The system shall be safe and secure on a technical layer and furthermore shall foster trust mechanisms on a conceptual level.

R62. The system shall provide fundamental security services such as authentication, confidentiality and integrity.

R63. The system shall support the enforcement of security policies.

R79. The system shall provide users authentication

R80. The user shall be able to set what personal information wants to share and with whom he/she wants to share it

R81. The personal user's information shall be protected from unauthorized accesses

4.6 OSGI

The Open Services Gateway initiative framework is a module system and service platform for the Java programming language that implements a complete and dynamic component model. Applications or components (coming in the form of bundles for deployment) can be remotely installed, started, stopped, updated and uninstalled without requiring a reboot; management of Java packages/classes is specified in great detail. Application life cycle management (start, stop, install, etc.) is done via APIs that allow for remote downloading of management policies. The service registry allows bundles to detect the addition of new services, or the removal of services, and adapt accordingly.

R60. The system shall provide remote service discovery and management. Service management and discovery will be independent of the network layer.

R64. The P2P overlay network shall be scalable, decentralized, extensible and flexible.

R65. The P2P layer shall provide an application agnostic communication overlay.

R66. It should be a tailored overlay to the needs raised by the services and applications running on top.

R74. The services platform must support service life cycle management at runtime.

R87. The service platform must support communication using P2P networking.

R78. It may be supported to contact a central location for obtaining new services, security updates etc. apart from the P2P network.

4.7 Support a wide range of end-user devices

The JXSE implementation of the JXTA platform and the JXME version for mobile devices, allows different components of the system to run on a wide range of end user devices.

R68. The system shall support a wide range of end-user terminals in terms of processing power and display capabilities.

R69. The hardware that will run the service platform must have adequate network interfaces to communicate with other devices in the home network and the P2P network.

R72. Handheld device should be suitable (simple enough and lightweight) for use by the elderly.

4.8 Why JXTA ?

JXTA is an open network computing platform designed for peer-to-peer (P2P) computing by providing basic building blocks and services required to enable and “anything, anywhere” application connectivity.

The name “JXTA” is not an acronym. It is short hand for *juxtapose*, as in side by side. It is a recognition that P2P is juxtaposed to client-server or Web-based computing, which is today’s traditional distributed computing model.

JXTA provides a common set of open protocols backed with open source reference implementations for developing peer-to-peer applications. The JXTA protocols standardize the manner in which peers:

- Discover each other
- Self-organize into peer groups
- Advertise and discover network resources
- Communicate with each other
- Monitor each other

The JXTA protocols are designed to be independent of programming languages and transport protocols alike. The protocols can be implemented in the Java programming language, C/C++, .NET, Ruby, and numerous other languages. Furthermore, they can be implemented on top of TCP/IP, HTTP, Bluetooth, and other network transports while maintaining global interoperability.

The JXTA protocols enable developers to build and deploy interoperable P2P services and applications. Because the protocols are independent of both programming language and transport protocols, heterogeneous devices with completely different software stacks can interoperate with one another. Using JXTA technology, developers can write networked, interoperable applications that can:

- Find other peers on the network with dynamic discovery across firewalls and NATs
- Easily share resources with anyone across the network
- Create a group of peers that provide a service
- Monitor peer activities remotely
- Securely communicate with other peers on the network

Information on the JXTA technology can be found at the Project JXTA web site. Resources include project information, documentation, mailing lists, source code, binaries, documentation, and tutorials.

5 JXTA

5.1 Overview

The JXTA software architecture is divided into three layers, as shown in Figure 6.

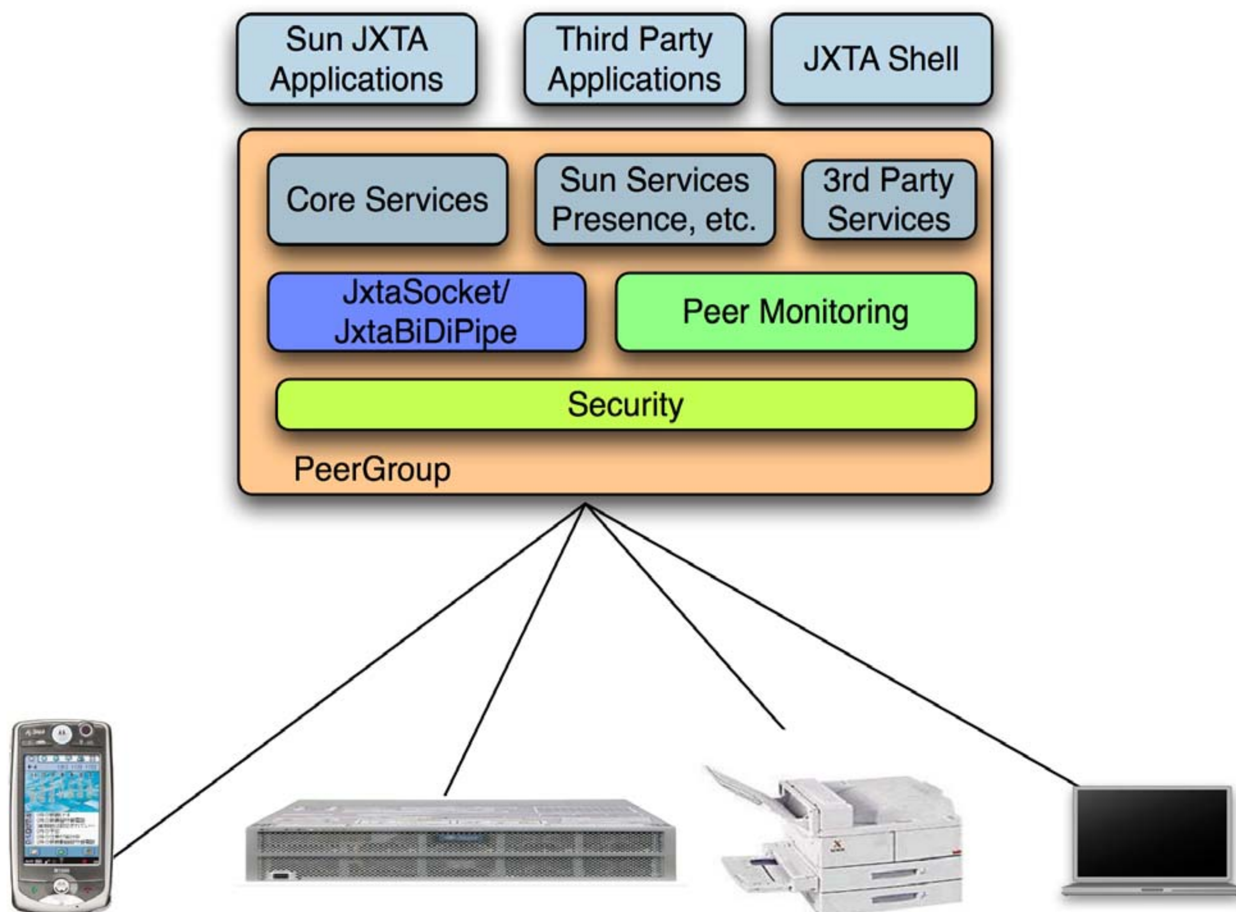

Figure 6. JXTA Software Architecture

- *JXTA Core*

  The JXTA core encapsulates the minimal and essential primitives that are common to P2P networking. It includes building blocks to enable key mechanisms for P2P applications, including discovery, communication transports (including firewall and NAT traversal), the creation of peers and peer groups, and associated security primitives.

- *Services Layer*

  The services layer includes network services that may not be absolutely necessary for a P2P network to operate, but are common or desirable in a P2P environment. Examples of network services include searching and indexing, directory, storage systems, file sharing, distributed file systems, resource aggregation and renting, protocol translation, authentication, and PKI (Public Key Infrastructure) services.

- *Applications Layer*

  The applications layer includes implementation of integrated applications, such as P2P instant messaging, document and resource sharing, entertainment content management and delivery, P2P E- mail systems, distributed auction systems, and many others.

The boundary between services and applications is not rigid. One customer's application can be

viewed as a service to another customer. The entire system is designed to be modular, allowing developers to pick and choose a collection of services and applications that suits their needs.

The JXTA network consists of a series of interconnected nodes, or *peers*. A peer may be any type of device from a sensor to a supercomputer or even a virtual process. Multiple peers may run on a single physical device and, potentially, multiple physical devices could cooperate to act as a single peer. The peers may be connected by any suitable networking protocol including TCP/IP, HTTP, Bluetooth, GSM, etc.

Each peer provides a set of *services* and *resources* which it makes available to other peers. Services are interactive programs and can include databases, authentication systems, chat servers or almost any program that can be networked. Two types of services are common within JXTA networks, *peer services* and *group services*. Peer services are those provided by a single peer. Group services are services which are provided in either a federated, redundant or cooperative way by the "whole group". Each Peer service instance is normally independent of other instances. Actions taken with one instance have no effect upon other instances. Each Peer group service instance is normally a participant in a common instance. Actions taken with one instance may (likely) have effects upon all instances.

All JXTA peers implement a small number of required *core services* and commonly also provide several additional *standard services*. Each Peer Group includes as part of it's definition the set of Group services which each peer must run in order to participate in the peer group.

A peer's resources are normally static (non-interactive) content which the peer either controls, owns or even merely has a copy of. Resources can include files, documents, media, advertisements, indexes but can also include real world resources such as switches, sensors and printers.

JXTA peers advertise their services and resources using XML documents called *advertisements.* Advertisements enable peers on the network to discover resources and services and to determine how to connect to and interact with those services.

Peers can organize themselves into *peer groups.* A Peer group, loosely defined, is any set of peers that provision and leverage a common set of services for a common purpose. There are two key aspects to this definition-common services and common purpose. Two peer groups might have the same set of services, for example a chat application, but different purposes, for example politics chat and sports chat. Peer groups can be defined on almost any basis that developers or deployers choose. For the preceding example the peer group could be redefined as providing a chat application for multiple topics but located within an organization, for example a university department. When defining a peer group the first two questions which must always be answered are; "What peers are members of this group?", and "What application or service are the peers cooperating to provide?".

JXTA peers use *sockets* and *pipes* to send *messages* to one another. JXTA sockets are reliable bi-directional connections used for applications to communicate reliably. Pipes are an asynchronous and unidirectional message transfer mechanism used for service communication. Messages are simple XML documents whose envelope contains routing, digest, and credential information. Pipes are bound to specific *endpoints*, such as a TCP port and associated IP address.

Four essential aspects of the JXTA architecture that distinguish it from other distributed network models are:

- The use of XML documents (advertisements) to describe network resources.
- Abstraction of pipes to peers, and peers to endpoints, without reliance upon a central naming/ addressing authority such as DNS.
- A uniform peer addressing scheme (IDs).

- A decentralized search infrastructure based on Distributed Hash Table (DHT) for resource indexing.

5.2 Peer

A *peer* is any networked entity that implements one or more of the JXTA protocols. Peers can reside on sensors, phones, and PDAs, as well as PCs, servers, and supercomputers. Each peer operates independently and asynchronously from all other peers and is uniquely identified by a Peer ID.

Peers publish one or more network addresses for use with the JXTA protocols. Each published address is advertised as a *peer endpoint,* which identifies the network address. Peer endpoints are used by peers to establish direct point-to-point connections between two peers.

Direct point-to-point network connections are not always available between peers. Intermediary peers may be used to route messages to peers that are separated due to physical network boundaries. The network boundaries can be natural boundaries such as Ethernet and Bluetooth networks or artificially created due to network configuration. Artificial barriers can include NAT, firewalls and proxies. The use of enlisted intermediate peers can and will change over time with no impact on the JXTA application.

Peers are typically configured to spontaneously discover each other on the network to form relationships known as peer groups, which can be transient or persistent in nature.

JXTA peers can be divided into three main types:

- *Minimal-Edge peers*: Peers that implement only the required core JXTA services and may rely on other peers to act as their proxy for other services to fully participate in a JXTA Network. The proxy peers act as proxy for the non-core services. Typical minimal-edge peers include sensor devices and home automation devices,
- *Full-Edge Peer*: Peers that implements all of the core and standard JXTA services and can participate in all of the JXTA protocols. These peers form the majority of peers on a JXTA network and can include phones, PC's, servers, etc.
- Super-Peer: Peers that implement and provision resources to support the deployment and operation of a JXTA network. There are three key JXTA Super Peer functions. A single peer may implement one or more of these functions.
    - **Relay**: Used to store and forward messages between peers that do not have direct connectivity because of firewalls or NAT. Only peers which are unable to receive connections from other peers require a relay.
    - **Rendezvous**: Maintains global advertisement indexes and assists edge and proxied peers with advertisement searches. Also handles message broadcasting.
    - **Proxy**: Used by minimal-edge peers to get access to all the JXTA network functionalities. The proxy peer translates and summarizes requests, responds to queries and provides support functionality for minimal-edge peers.

These categories describe the most common peer confiugrations. Depending upon the application and peer capabilities it may make sense to deploy the peers with a mix of functionality. For example, it my be reasonable to deploy peers with full Discovery and Pipe functionality but require a proxy for running group services.

5.3 Peer group

A *peer group* is a collection of peers that have agreed upon a common set of services, or interests. Peers self-organize into peer groups, each of which is uniquely identified by a peer group ID. Each peer group establishes its own membership policy including open (anybody can join) to highly secure and protected (requiring credentials to gain membership).

Peers can belong to more than one peer group simultaneously. By default, the first group that is instantiated is the Network Peer Group. All peers belong to the Network Peer Group and may choose to join additional peer groups at any time.

The JXTA protocols describe how peers may publish, discover, join, and monitor peer groups; they do not dictate when or why peer groups are created. A group join is simply instantiating all the peer group services defined by the peer group. There are several motivations for creating peer groups:

- *To create a secure environment*

  Groups create a local domain of control in which a specific security policy can be enforced. The security policy may be as simple as a plain text user name/password exchange, or as sophisticated as public key cryptography. Peer group boundaries permit member peers to access and publish protected content. Peer groups form logical regions whose boundaries limit access to the peer group's resources.

- *To create a scoping environment*

  Groups allow the establishment of a local domain of specialization. For example, peers may group together to implement a document sharing network or a CPU sharing network. Peer groups serve to subdivide the network into abstract regions providing an implicit scoping mechanism. Peer group boundaries define the search scope when searching for a group's content.

- *To create a monitoring environment*

  Peer groups permit peers to monitor a set of peers for any special purpose (e.g., heartbeat, traffic introspection, or accountability).

Groups can also form a hierarchical parent-child relationship, in which each group has a single parent. Search requests are propagated within the group. The advertisement for the group is published in the parent group in addition to the group itself.

5.4 Service

The JXTA protocols are implemented with the help of services and modules. These are the basic entities representing, 'things' a JXTA peer must know in order to operate on the JXTA network. Services and modules are constituents of the glue that makes the JXTA network stick together. The other constituents are standard messages defined in JXTA. At first, the distinction between modules and services in JXTA is not obvious. The relationship between both concepts is not explicitly described in the protocol specifications. However, a module is defined as “an abstraction used to represent any piece of code used to implement a behavior in the JXTA world”.

The implicit link between modules and services is that each service is ultimately implemented as a module. Services can be immediately loaded and available on the local peer, or can be accessed remotely using a pipe or another proxy module. Eventually, an authentication module can be used to check the communication with the service. The code can also be fetched from a remote location and loaded later. The publication of a service advertisement should contain all necessary information explaining how to use it or invoke it.
The JXTA specification 2.0 mention two types of services:

- Peer Services, of which individual instances run on each peer. If a peer goes down, the individual service goes down too. Each instance of the service should publish its own advertisement.
- Peer Group Services are published within peer group advertisements. Instances of these services run on each peer participating in the peer group. Typically, these services may communicate with each other.

All these types and sub-types of services, together with their publication processes, can be confusing. One should remember that, in practice, peers are modules creating peer group modules and that services (which are modules too) are attached to peer groups. A peer communicates and operates with other peers using the services attached to the peer groups it has created. Customized or additional services can be loaded on peer groups created by the peer too.

JXTA defines core and standard services, these implement the protocols that we will describe later:

- **Access Service.** This is the service verifying the credentials and information of a request to access resources. .
- **Discovery Service.** This is the service allowing tribes to search for other tribes or peer groups within a peer group. Technically speaking, they will search for the advertisements representing them. The discovery service can also help searching for other types of resources, such as routes to islands or trading routes between tribes. Newcomers to JXTA often believe that discovery means finding out whether a peer is connected online. But in the JXTA paradigm, discovery means discovery of advertisements describing resources (such as peers, peergroups, services, etc...) within a given peer group. It is not the instance of these resources themselves. It is like confusing a Class and an object instance of this class in the Java programming language.
- **Endpoint Service**. This service is responsible for transmitting a message from one peer to another peer.
- **Membership Service.** This is the service used to allow or reject a new request for membership in a peer group. It can be as simple as always approving a new member or more complex, like using a voting procedure. A tribe willing to join a group must first find one of its members and request to join the group.
- **Peer Info Service.** This service helps peers find about the status of other peers in a peer group.
- **Pipe Service.** This service creates trading routes between one or many tribes (not islands) belonging to the same peer group.
- **Rendezvous Service.** This service is operated by peers acting as rendezvous to facilitate the efficient forwarding of queries to peers belonging to the peer group.
- **Resolver Service.** This service is used to address queries made by a tribe leader to another tribe leader and to collect responses.

Some services implement core specifications that all JXTA implementations should deliver. Other services are considered standard and should preferably be implemented, but this is not mandatory. Remaining services are not mandatory

Service/Functionalities	Requirement
Endpoint service	Core
Resolver service	Core

Discovery service	Standard
Peer information service	Standard
Pipe service	Standard
Rendezvous service	Standard

5.5 Modules

JXTA modules are a low-level JXTA abstraction used to represent any piece of "code" and the interface (API) which that code provides. Modules are used to implement services, message transports and other loadable bits of JXTA code. Most JXTA developers typically don't have to deal with modules as distribution. That includes the initial set of services required by most applications. The module abstraction does not specify the physical "code" implementation, as it can be provided as a Java class, a Java jar, a dynamic library DLL, a set of XML messages, or a script. The implementation of the module behavior is left to module implementer. For instance, modules can be used to represent different implementations of a network service on different platforms, such as the Java platform, Microsoft Windows, or the Solaris Operating Environment.

Modules provide a generic abstraction to allow a peer to instantiate a function or service. When a peer joins a peer group they may find new behaviors that they may want to instantiate. For example, when joining a peer group, a peer may be required to provide a new search service that is only used in this peer group. In order to join this group, the peer must instantiate this new search service. The module framework enables the representation and advertisement of platform-independent behaviors, and allows peers to describe and instantiate any type of implementation of a behavior. For example, a peer has the ability to instantiate either a Java or a C implementation of the specified behavior.

The ability to describe and publish platform-independent behavior is essential to support the development of new peer group services which are provisioned by a heterogeneous cadre of peers. The module advertisement enables JXTA peers to describe a behavior in a platform-independent manner. In fact, JXTA uses module advertisements to self-describe it's own services.

The module abstraction includes a module class, module specification, and module implementation:

- *Module Class*

  The module class is primarily used to advertise the existence of a behavior. The class definition represents an expected behavior and an expected binding to support the module. Each module class is identified by a unique ID, the ModuleClassID.

- *Module Specification*

  The module specification is primarily used to access a module. It contains all the information necessary to access or invoke the module. For instance, in the case of a service, the module specification may contain a pipe advertisement to be used to communicate with the service.

  A module specification is one approach to providing the functionality that a module class implies. There can be multiple module specifications for a given module class. Each module specification is identified by a unique ID, the ModuleSpecID. The ModuleSpecID contains the ModuleClassID (i.e., the ModuleClassID is embedded in a ModuleSpecID), indicating the associated module class.

  A module specification implies network compatibility. All implementations of a given module specification must use the same protocols and are compatible, although they may be

written in a different language.

- *Module Implementation*

  The module implementation is the implementation of a given module specification. There may be multiple module implementations for a given module specification. Each module implementation contains the ModuleSpecID of the associated specification it implements.

Modules are used by peer group services, and can also be used by stand-alone services. JXTA services can use the module abstraction to identify the existence of the service (its Module Class), the specification of the service (its Module Specification), or an implementation of the service (a Module Implementation). Each of these components has an associated advertisement which can be published and discovered by other JXTA peers.

As an example, consider the JXTA Discovery Service. It has a unique ModuleClassID, identifying it as a discovery service — its abstract functionality. There can be multiple specifications of the discovery service, each possibly incompatible with each other. One may use different strategies tailored to the size of the group and its dispersion across the network, while another experiments with new strategies. Each specification has a unique ModuleSpecID, which references the discovery service ModuleClassID. For each specification, there can be multiple implementations, each of which contains the same ModuleSpecID.

In summary, there can be multiple specifications of a given module class, and each may be incompatible. However, all implementations of any given specification are assumed to be compatible.

5.6 Message

JXTA services and applications communicate using JXTA Messages. JXTA Messages are the basic unit of data exchange between peers. Each JXTA protocol is defined as a set of messages which the participating peers exchange. Messages are sent between peers using the Endpoint Service and the Pipe Service as well as JxtaSocket and other approaches. Most applications do not need to use unidirectional pipe or the JXTA Endpoint Service directly. Instead, applications and services commonly use the JXTA Socket and JxtaBiDiPipe communication channels to send, and receive messages.

The JXTA protocols are specified as a set of messages exchanged between peers. The use of XML messages to define protocols allows many different kinds of peers to utilize a given protocol. Because the data is tagged, each peer is free to implement the protocol in a manner best suited to its abilities and role. If a peer only needs some subset of the message, the XML data tags enable that peer to identify the parts of the message that are of interest. For example, a peer that is highly constrained, and has insufficient capacity to process some or most of a message, can use data tags to extract the parts that it can process and ignore the remainder. Each software platform binding describes how a message is converted to and from a native data structure such as a Java object or a C structure.

The JXTA protocols define two “on-wire” representations for messages: XML and binary. These on-wire representations are the data format used for transmitting the message between peers. Different on-wire formats are used to take best advantage of the characteristics of the underlying network transport.

5.7 Pipes

JXTA peers use *pipes* to send messages to one another. Pipes are an asynchronous, unidirectional and non-reliable (with the exception of unicast secure pipes) message transfer mechanism used for communication and data transfer. Pipes are virtual communication channels

and may connect peers that do not have a direct physical link, resulting in a logical connection. Pipes can be used to send any type of data including XML and HTML text, images, music, binary code, data strings and Java Objects.
The pipe endpoints are referred to as the receiving *input pipe* and the sending *output pipe*. Pipe endpoints are dynamically bound to peer endpoints when the pipe is opened. Peer endpoints correspond to available peer network interfaces with an example being a TCP port and associated IP address, that can be used to send and receive messages. JXTA pipes can have endpoints that are connected to different peers at varying times, or may not be connected at all. All pipe resolution and communication is done within the scope of a peer group. That is, the output and input pipes must belong to the same peer group.
Pipes offer two modes of communication, point-to-point and propagate, as seen in the diagram below. JXSE (open Source Java implementation of the JXTA protocols standard edition) also provides secure unicast pipes, a secure variant of the point-to-point pipe.

- *Point-to-point Pipes*

  A point-to-point pipe connects exactly two pipe endpoints together, an input pipe on one peer receives messages sent from the output pipe of another peer. It is also possible for multiple peers to bind to a single input pipe.

- *Propagate Pipes*

  A propagate pipe connects one output pipe to multiple input pipes. Messages flow from the output pipe, the propagation source, into the input pipes.

- *Secure Unicast Pipes*

  A secure unicast pipe is a type of point-to-point pipe that provides a secure and reliable communication channel.

Unidirectional pipes are a very-low level JXTA communication programming abstraction. It is recommended that developers use the higher-level communication abstraction provided by the JxtaSocket and JxtaBiDipipe services described in the next section.

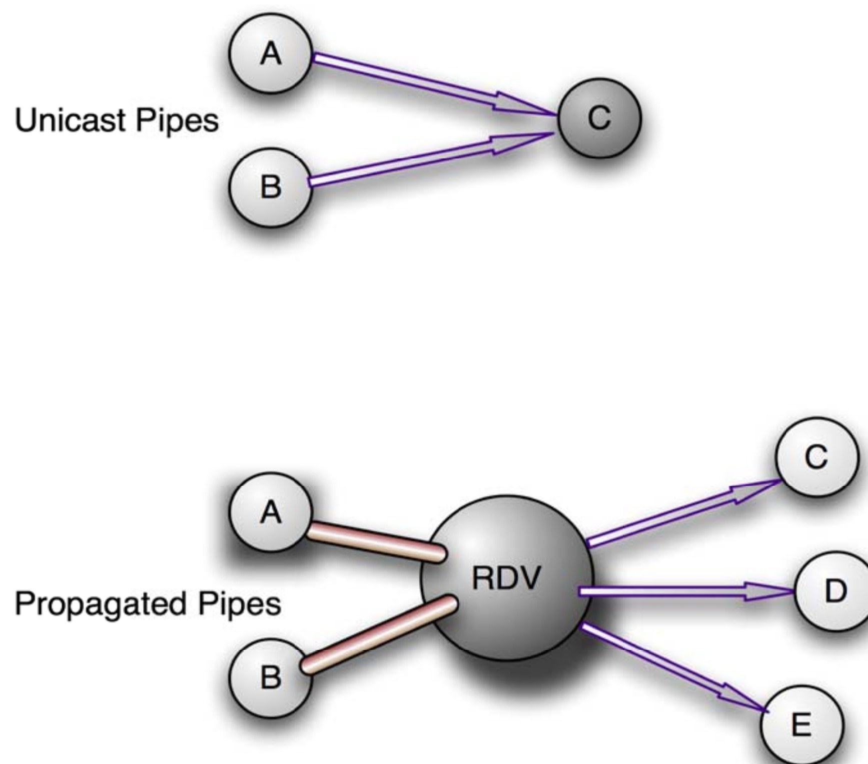

Figure 7. Unicast and Propagate Pipes

(Bidirectional reliable communication channels)

The basic JXTA pipes provide unidirectional, unreliable communication channels. In order to make pipes more useful to services and applications it is necessary to implement bidirectional and reliable communication channels on top of the pipe primitives. JXSE provides functionality to meet the level of service quality required by most applications:

- Reliability
- Ensures message sequencing
- Ensures delivery
- Exposes message and stream interfaces
- Security
- JxtaSocket and JxtaServerSocket :
  - Sub-class java.net.Socket and java.net.ServerSocket respectively
  - Are built on top of pipes, endpoint messengers, and the reliability library
  - Provide bidirectional, reliable and secure communication channels
  - Expose a stream based interface.
  - Provide configurable internal buffering and message chunking
  - Does not implement Nagle's algorithm, therefore streams must be flushed as needed
- JxtaBiDiPipe and JxtaServerPipe provides:
  - Are built on top of pipes, endpoint messengers, and the reliability library
  - Provide bidirectional, reliable, and secure communication channels
  - Expose a message based interface
  - Does not provide message chunking. Applications need to ensure message size does not exceed the standard message size limitation of 64K.

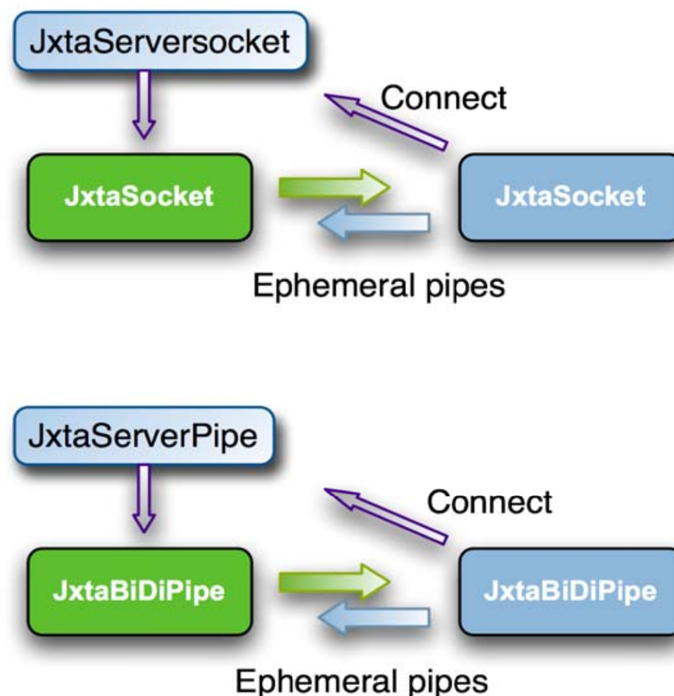

Figure 8. Pipe Connections

JxtaServerSocket and JxtaServerPipe expose an input pipe to process connection requests and negotiate communication parameters. JxtaSocket and JxtaBiDiPipe, on the other hand, bind to respective private dedicated pipes independent of the connection request pipe.

5.8 Advertisement

All JXTA network resources — such as peers, peer groups, pipes, and services — are represented as *advertisements*. Advertisements are language-neutral meta-data structures represented as XML documents. The JXTA protocols use advertisements to describe and publish the existence of a peer's resources. Peers discover resources by searching for their corresponding advertisements, and may cache any discovered advertisements locally.
Each advertisement is published with a lifetime that specifies the availability of its associated resource. Lifetimes enable the deletion of obsolete resources without requiring any centralized control. An advertisement can be republished (before the original advertisement expires) to extend the lifetime of a resource.

The JXTA protocols define the following advertisement types:

- *Peer Advertisement* — describes the peer's resources. The primary use of this advertisement is to hold specific information about the peer, such as its name, peer ID, available endpoints, and any run-time attributes which individual group services want to publish (such as being a rendezvous peer for the group).
- *Peer Group Advertisement* — describes peer group-specific resources, such as name, peer group ID, description, specification, and service parameters.
- *Pipe Advertisement* — describes a pipe communication channel, and is used by the pipe service to create the associated input and output pipe endpoints. Each pipe advertisement contains an optional symbolic ID, a pipe type (point-to-point, propagate, secure, etc.) and a unique pipe ID.
- *Module Class Advertisement* — describes a module class. Its primary purpose is to formally document the existence of a module class. It includes a name, description, and a unique ID (ModuleClassID).
- *ModuleSpecAdvertisement* — defines a module specification. Its main purpose is to provide references to the documentation needed in order to create conforming implementations of that specification. A secondary use is, optionally, to make running instances usable remotely, by publishing information such as a pipe advertisement. It includes name, description, unique ID (ModuleSpecID), pipe advertisement, and parameter field containing arbitrary parameters to be interpreted by each implementation.
- *ModuleImplAdvertisement* ¯ defines an implementation of a given module specification. It includes a name, associated ModuleSpecID, as well as code, package, and parameter fields which enable a peer to retrieve data necessary to execute the implementation.
- *Rendezvous Advertisement* — describes a peer that acts as a rendezvous peer for a given peer group.
- *Peer Info Advertisement* — describes the peer info resource. The primary use of this advertisement is to hold specific information about the current state of a peer, such as uptime, inbound and outbound message count, time last message received, and time last message sent.

Each advertisement is represented by an XML document. Advertisements are composed of a series of hierarchically arranged elements. Each element can contain its data or additional elements. An element can also have attributes which are comprised of name-value string pairs.

An attribute is used to store meta-data, which helps to describe the data within the element.

An example of a pipe advertisement is included in Figure 9.

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PipeAdvertisement>
<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
 <Id>
  urn:jxta:uuid-59616261646162614E504720503250338E3E786229EA460DADC1A176B69B731504
  </Id>
  <Type>
   JxtaUnicast
  </Type>
  <Name>
   TestPipe
  </Name>
  </jxta:PipeAdvertisement>
```

Figure 9. A Pipe Advertisement

The complete specification of the JXTA advertisements is given in the *JXTA Protocols Specification* (see.http://jxta-spec.dev.java.net). Services or peer implementations may subtype any of the above advertisements to create their own application advertisements.

5.9 Security

Dynamic P2P networks, such as the JXTA network, need to support different levels of resource access. JXTA peers operate in a role-based trust model, in which an individual peer acts under the authority granted to it by another trusted peer to perform a particular task.

Five basic security requirements must be provided:

- *Confidentiality* — guarantees that the contents of a message are not disclosed to unauthorized individuals.
- *Authentication* — guarantees that the sender is who he or she claims to be.
- *Authorization* — guarantees that the sender is authorized to send a message.
- *Data integrity* — guarantees that the message was not modified accidentally or deliberately in transit.
- *Refutability* — guarantees that the message was transmitted by a properly identified sender and is not a replay of a previously transmitted message.

XML messages provide the ability to add meta-data such as credentials, certificates, digests, and public keys to JXTA messages, enabling these basic security requirements to be met. Message digests and signatures guarantee the data integrity of messages. Messages may also be encrypted and signed for confidentiality and refutability. Credentials can be used to provide message authentication and authorization.

A credential is a token that is used to identify a sender, and it can be used to verify a sender's right to send a message to a specified endpoint. The credential is an opaque token that must be presented each time a message is sent. The sending address placed in a JXTA message envelope is cross-checked with the sender's identity in the credential. Each credential's implementation is specified as a plug-in configuration, which allows multiple authentication configurations to co-exists on the same network.

It is the intent of the JXTA protocols to be compatible with widely accepted transport-layer security mechanisms for message-based architectures, such as Secure Sockets Layer (SSL) and Internet Protocol Security (IPSec).

5.10 IDs

Peers, peer groups, pipes and other JXTA resources need to be uniquely identifiable. A JXTA ID uniquely identifies a resource and serves as a canonical way of referring to that resource. Currently, there are six types of JXTA entities which have JXTA ID types defined: peers, peer groups, pipes, content, module classes, and module specifications.
URNs are used to express JXTA IDs. URNs are a form of URI that “are intended to serve as persistent, location-independent, resource identifiers”. Like other forms of URI, JXTA IDs are presented as text.

An example JXTA peer ID is:
urn:jxta:uuid-59616261646162614A78746150325033F3BC76FF13C2414CBC0AB663666DA53903

An example JXTA pipe ID is:
urn:jxta:uuid-59616261646162614E504720503250338E3E786229EA460DADC1A176B69B731504

Every JXTA ID has an *ID Format*. The format describes how the ID was generated and how it may be manipulated by programs. Every ID indicates it's format immediately after the urn:jxta: prefix. There are two common JXTA ID Formats, uuid and jxta, though others exist. The jxta format is used for special common identifiers such as the IDs of the World Peer Group and the Network Peer Group. The uuid format is used for most other IDs. The uuid format provides randomly generated unique IDs and is based upon DCE GUID/UUIDs. The portion of a JXTA ID which follows the ID Format is specific to each ID Format and is often opaque—aren't meant to be able be decoded directly from the URI.

5.11 Network architecture

The JXTA network is an ad-hoc, multi-hop, and adaptive network composed of connected peers. Connections in the network may be transient and, as a result, message routing between peers is non-deterministic. Peers may join or leave the network at any time; which results in ever changing routing information.
The only common aspect that various JXTA applications share is that they communicate using JXTA protocols. The organization of the network is not mandated by the JXTA framework, but in practice four kinds of peers are typically used (see Figure 10):

- *Minimal edge peer*

A minimal edge peer can send and receive messages, but does not cache advertisements or route messages for other peers. Peers on devices with limited resources (e.g., a PDA or cell phone) would likely be minimal edge peers.

- *Full-featured edge peer*

A full-featured peer can send and receive messages and will typically cache advertisements. A simple peer replies to discovery requests with information found in its cached advertisements, but it does not forward any discovery requests. In any JXTA deployment most peers are likely to be edge peers.

- *Rendezvous peer*

A rendezvous peer is an infrastructure peer, it aids other peers with message propagation, discovery of advertisements and routes, and most importantly it maintains a topology map of of other infrastructure peers, which then used for controlled propagation, and maintenance of the distributed hash table. Each peer group maintains its own set of rendezvous peers and may have as many rendezvous peers as needed. Edge peers send search and discovery requests to their

rendezvous peer which in turn may forward requests it cannot answer to other known rendezvous peers using the topology mapped distributed hash table.

- *Relay peer*[1]

A relay peer s an infrastructure peer, it aids non addressable (firewalled/NAT'd) peers with message relaying. A peer may request an in memory message box from a relay peer to facilitate message relaying whenever needed.

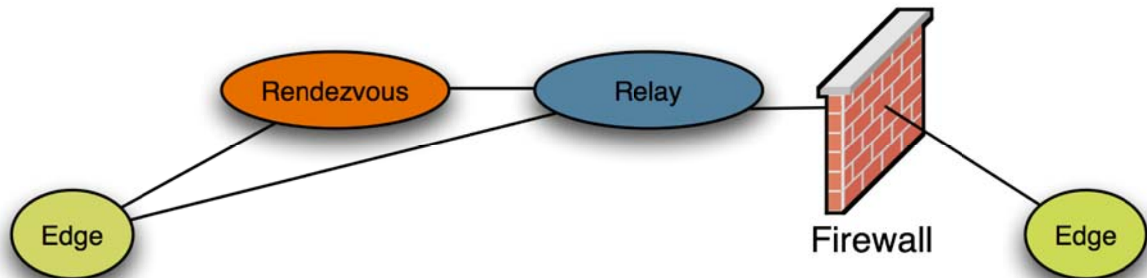

Figure 10. The four type of nodes

5.11.1 Shared Resource Distributed Index (SRDI)

JXSE supports a shared resource distributed index (SRDI) service to provide an efficient mechanism for propagating query requests within the JXTA network. Rendezvous peers maintain an index of advertisements published by edge peers. When edge peers publish new advertisements, they use the SRDI service to push advertisement indexes to their rendezvous. With this rendezvous-edge peer hierarchy, queries are propagated between rendezvous only, which significantly reduces the number of peers involved in the search for an advertisement.
Each rendezvous maintains its own list of known rendezvous in the peer group. A rendezvous may retrieve rendezvous information from a predefined set of bootstrapping, or seeding, rendezvous. Rendezvous periodically select a given random number of rendezvous peers and send them a random list of their known rendezvous. Rendezvous also periodically purge non-responding rendezvous. Thus, they maintain a loosely-consistent network of known rendezvous peers.
When a peer publishes a new advertisement, the advertisement is indexed by the SRDI service using keys such as the advertisement name or ID. Only the indexes of the advertisement are pushed to the rendezvous by SRDI, minimizing the amount of data that needs to be stored on the rendezvous. The rendezvous also pushes the index to additional rendezvous peers (selected by the calculation of a hash function of the advertisement index).

5.11.2 Queries

An example configuration is shown in Figure 11. Peer A is an edge peer and is configured to use Peer R1 as its rendezvous. When Peer A initiates a discovery or search request, it is initially sent to its rendezvous peer — R1, in this example — and also via multicast to other peers on the same subnet. Local network queries (i.e., within a subnet) are propagated to local network peers using what a transport defines as the broadcast or multicast method. Peers receiving the query respond directly to the requesting peer if they contain the information in their local cache.

1 Relay peers were referred to as router peers in early JXTA documentation.

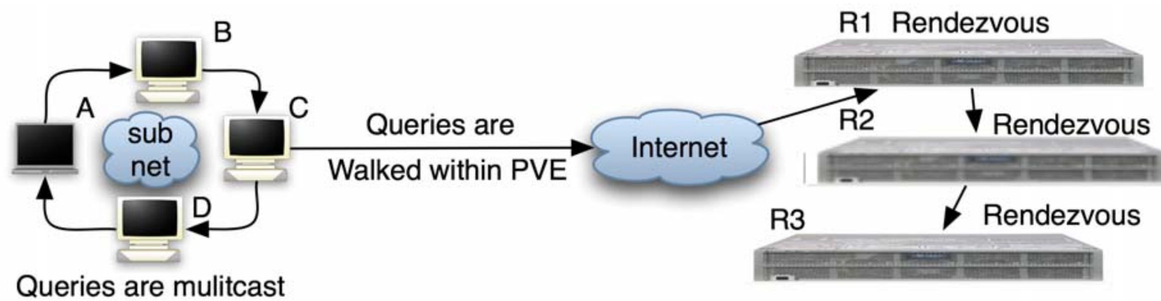

Figure 11. Request Propagation via Rendezvous Peers

Queries beyond the local network are sent to the connected rendezvous peer. The rendezvous peer attempts to satisfy the query against its local cache. If it contains the requested information, it replies directly to the requesting peer and does not further propagate the request. If it contains the index for the resource in its SRDI cache, it will notify the peer that published the resource and that peer will respond directly to the requesting peer. The rendezvous is unable to respond directly to the querying peer because the rendezvous stores only the index for the advertisement and not the advertisement itself.

If the rendezvous peer does not contain the requested information, a default limited-range walker algorithm is used to walk the set of rendezvous nodes looking for a rendezvous that contains the index. A query path may be altered by a network map function to reduce the TTL of a query; A hop count is used to specify the maximum number of times the request is mapped/forwarded to avoid ping-pong effects which can occur in unstable or very dynamic networks. Once the query reaches the peer, it replies directly to the originator of the query.

SRDI uses a SHA1 hash addressing scheme, where the 160 bit hash address space is divided amongst a ordered list of rendezvous nodes. When indexes are received they are hashed to determine their replication address, then replicated on their destination replica rendezvous.
Figure 12 depicts a logical view of how the SRDI service works. Once Node A publishes a set of advertisements, a set of indexes in the form of an SRDI message is sent to its rendezvous, RDV1, where such Indexes are stored, then replicated (based on their hash mapping) on rendezvous 2, 3, and 4. Node C then issues a query for advertisement A, which is walked to rendezvous 2, then mapped to rendezvous 3, then finally forwarded node A.

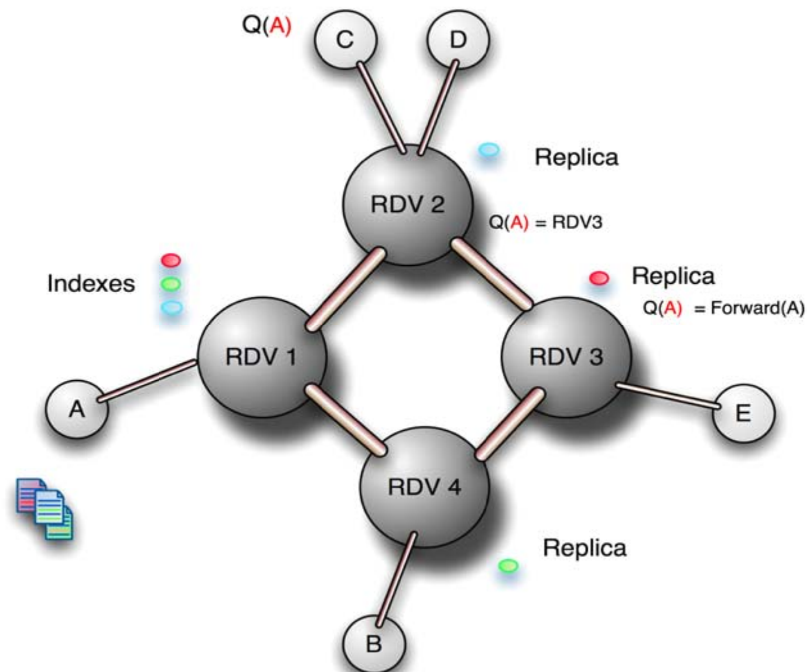

Figure 12. SRDI Operation

5.12 Firewalls and NAT

A peer behind a firewall can send a message directly to a peer outside a firewall, but a peer outside the firewall cannot establish a direct connection with a peer behind the firewall. The same is true for peers which are behind a NAT device.

In order for JXTA peers to communicate with each other across a firewall, the following conditions must exist:

- At least one peer in the peer group inside the firewall must be aware of at least one peer outside of the firewall.
- The peer inside and the peer outside the firewall must be aware of each other and must support a common transport (HTTP or TCP).
- The firewall, at the very least, has to allow outbound HTTP or TCP connections. Figure 4-3 depicts a typical message routing scenario through a firewall. In this scenario, JXTA Peers A and B want to pass a message, but the firewall prevents them from communicating directly. JXTA Peer A first makes a connection to Peer C using a protocol such as HTTP that can penetrate the firewall. Peer C then makes a connection to Peer B using a protocol such as TCP/IP. A virtual connection is now made between Peers A and B.

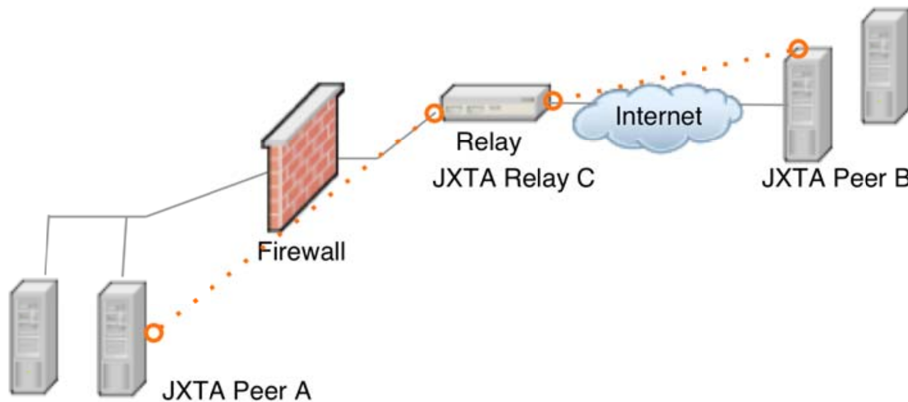

Figure 13. Message Routing Scenario Across a Firewall

5.13 JXTA protocols

JXTA defines a series of XML messages, or *protocols*, for communication between peers. Peers use these protocols to discover one another, advertise and discover network resources, and communication and route messages.

There are six standard JXTA protocols[2]:

- *Peer Discovery Protocol (PDP)* — used by peers to advertise their own resources (e.g., peers, peer groups, pipes, or services) and discover resources from other peers. Each peer resource is described and published using an advertisement.
- *Peer Information Protocol (PIP)* — used by peers to obtain status information (uptime, state, recent traffic, etc.) from other peers.
- *Peer Resolver Protocol (PRP)* — enables peers to send a generic query to one or more peers and receive a response (or multiple responses) to the query. Queries can be directed to all peers in a peer group or to specific peers within the group. Unlike PDP and PIP, which are used to query specific predefined information, this protocol allows peer services to define and exchange any arbitrary information they need.
- *Pipe Binding Protocol (PBP)* — used by peers to establish a virtual communication channel, or *pipe*, between one or more peers. The PBP is used by a peer to bind two or more ends of the connection (pipe endpoints).
- *Endpoint Routing Protocol (ERP)* — used by peers to find routes (paths) to destination ports on other peers. Route information includes an ordered sequence of relay peer IDs that can be used to send a message to the destination. (For example, the message can be delivered by sending it to Peer A which relays it to Peer B which relays it to the final destination.)
- *Rendezvous Protocol (RVP)* — used by edge peers to resolve resources, propagate messages, and advertise local resources. used by rendezvous peers to organize with other rendezvous peers, share the distributed hash table address space, and propagate messages in controlled fashion (message walkers).

2 For a complete description of the JXTA protocols, please see the *JXTA Protocols Specification*, available for download from http://jxta-spec.dev.java.net.

All of the standard JXTA protocols are asynchronous and are based on a query/response model. A JXTA peer uses one of the protocols to send a query to one or more peers in its peer group. It may receive zero, one, or more responses to its query. For example, a peer may use PDP to send a discovery query asking for all known peers in the default Net Peer Group. In this case, multiple peers will likely reply with discovery responses. In another example, a peer may send a discovery request asking for a specific pipe named “aardvark”. If this pipe isn’t found, then zero discovery responses will be sent in reply.

JXTA peers are not required to implement all six protocols; they only need implement the protocols they will use. JXSE supports all six JXTA protocols. The Java SE API is used to access operations supported by these protocols, such as discovering peers or joining a peer group.

5.13.1 Peer Discovery Protocol

The Peer Discovery Protocol (PDP) is used to discover any published peer resources. Resources are represented as advertisements. A resource can be a peer, peer group, pipe, service, or any other resource that has an advertisement.

PDP enables a peer to find advertisements on the network. The PDP is the default discovery protocol for all user defined peer groups and the default net peer group. Custom discovery services may choose to leverage the PDP. If a peer group does not define an alternate discovery service, the PDP is used to probe the network for advertisements.

There are multiple ways to discover distributed information. The current JXSE implementation uses a combination of IP multicast to the local subnet and the use of a rendezvous network, which is a technique based on a rendezvous maintained DHT (Distributed Hash Table). Rendezvous nodes provide the mechanism of directing requests into the network to dynamically discover information. A node may be configured with a predefined set of rendezvous nodes. A node may also choose to bootstrap itself by dynamically locating rendezvous nodes or network resources in its local network via multicast messages.

Nodes generate discovery query messages to discover advertisements within a peer group. This message is enclosed within a resolver query contains the peer group credential of the probing node and identifies the probing peer to the message recipient. Messages can be sent to any node within a peer group.

A query is not guaranteed to result in any responses, or result in responses matching the requested threshold.

5.13.2 Peer Information Protocol

Once a node is located, it's capabilities and status may be queried. The Peer Information Protocol (PIP) provides a set of messages to obtain peer status information. This information can be used for commercial or internal deployment of JXTA applications. For example, in commercial deployments the information can be used to determine the usage of a peer service and bill the service consumers for their use. In an internal IT deployment, the information can be used by the IT department to monitor a node’s behavior and reroute network traffic to improve overall performance. These hooks can be extended to enforce the IT department's control of the node in addition to providing status information.

The PIP ping message is sent to a peer to check if the peer is alive and to get information about the peer. The ping message specifies whether a full response (peer advertisement) or a simple acknowledgment (alive and uptime) should be returned.

The PeerInfo message is used to send a message in response to a ping message. It contains the credential of the sender, the source peer ID and target peer ID, uptime, and peer advertisement.

5.13.3 Peer Resolver Protocol

The Peer Resolver Protocol (PRP) enables peers to send generic query requests to other peers and identify matching responses. Query requests can be sent to a specific peer or can be propagated via the rendezvous services within the scope of a peer group. The PRP uses the Rendezvous Service to disseminate a query to multiple peers and uses unicast messages to send queries to specified peers.

The PRP is a foundation protocol supporting generic query requests. Both PIP and PDP are built using PRP and they provide specific query/requests: the PIP is used to query specific status information and PDP is used to discover peer resources. The PRP can be used for any generic query that may be needed for an application. For example, the PRP enables peers to define and exchange queries and to find or search service information (such as the state of the service, the state of a pipe endpoint, etc).

The resolver query message is used to send a resolver query request to a service running on another member of a peer group. The resolver query message contains the credential of the sender, a unique query ID, a specific service handler, and the query. Each service can register a handler in the peer group's resolver service to process resolver query requests and generate replies. The resolver response message is used to send a message in response to a resolver query message. The resolver response message contains the credential of the sender, a unique query ID, a specific service handler, and the response. Multiple resolver query messages may be sent. A peer may receive zero, one, or more responses to a query request.

Peers may also participate in the Shared Resource Distributed Index (SRDI). SRDI provides a generic mechanism enabling JXTA services to utilize a distributed index of shared resources with other peers that are grouped as a set of more capable peers, such as rendezvous peers. These indexes can be used to forward queries in the direction where the query is most likely to be answered and repropagates the messages to peers interested in those messages. The PRP sends a resolver SRDI message to the named handler on one or more peers in the peer group. The resolver SRDI message is sent to a specific handler and it contains a string that will be interpreted by the targeted handler.

5.13.4 Pipe Binding Protocol

The Pipe Binding Protocol (PBP) is used by peer group members to bind a pipe advertisement to a pipe endpoint. The pipe virtual link (or pathway) can be layered upon any number of physical network transport links, such as TCP/IP. Each end of the pipe works to maintain the virtual link and to re-establish it, if necessary, by binding or finding the pipe’s currently bound endpoints.

A pipe can be viewed as an abstract named message queue, which supports create, open/resolve (bind), close (unbind), delete, send, and receive operations. Actual pipe implementations may differ, but all compliant implementations use PBP to bind the pipe to an endpoint. During the abstract create operation, a local peer binds a pipe endpoint to a pipe transport.

The PBP query message is sent by a peer pipe endpoint to find a pipe endpoint bound to the same pipe advertisement. The query message may ask for information not obtained from the cache. This is used to obtain the most up-to-date information from a peer. The query message can also contain an optional peer ID which, if present, indicates that only the specified peer should respond to the query.

The PBP answer message is sent back to the requesting peer by each peer bound to the pipe. The message contains the Pipe ID, the peer where a corresponding InputPipe has been created, and a boolean value indicating whether the InputPipe exists on the specified peer.

5.13.5 Endpoint Routing Protocol

The Endpoint Routing Protocol (ERP) enables JXTA peers to send messages to remote peers without having a direct connection to the destination peer. The message will be passed through intermediary peers to reach it's final destination. ERP defines a query and response protocol which it uses to discover peer routing information and a message “envelope” that is attached to JXTA Messages describing the route a message should travel from one peer (the source) to another (the destination).
To send a message to another peer, the source peer first looks in its local cache to determine if it has a route to the destination peer. If it does not already have a route to the destination peer, it sends a route resolver query request asking for route information to the destination peer. Any peer receiving this route query will check to see if knows a route to the requested peer. If the peer does know of a valid route, it will respond to the route query with the route information for the desired destination peer. Any peer can query for route information and any peer within a peer group may offer route information and/or route messages destined for other peers. Relay peers typically cache route information.

Route query requests are sent by a peer to request route information for a destination peer. Route responses include the peer ID of the responder, the peer ID of the route's destination, and a semi-ordered sequence of peer IDs. The sequence of peer ID hops may provide a complete or partial route to the destination but at minimum contains one peer ID. In some cases the sequence may contain several alternative routes to the destination. A route can safely express alternatives because of the way in which routes are used.

The semi-ordered sequence of peer IDs provided in a Route Response provide information as to the path that a message may be forwarded in order to reach a destination peer. Each peer along the stated path may enhance and/or optimize the route the message takes based upon it's own knowledge. For example, if a peer receives a message containing a ten hop route to a destination peer but it is itself directly connected to the destination peer then it makes sense to forward the message directly rather than sending it along the long route. Similarly, a peer may shorten a route by using a shorter route it knows between any two hops in the route which is included with a message.

The Message routing procedure used by the Endpoint Routing Protocol is roughly as follows;

1. If I am originating the message then check if I have a route to the destination. If I don't have a route to the destination, query for a route and wait for a route to be found. Eventually give up if no route is found.
2. If I am not originating the message and do not have a route for the destination nor any peer listed in the route attached to the message then send a failure message to the peer from which I received the message. JXTA peers do not currently generate route queries for messages they do not originate as this is too easily used to create distributed denial of service attacks (DdoS).
3. Remove my peer ID from the message route and all peer IDs which preceded mine in the message route.- Excluding all Peer Ids already in the message route, prepend the route I know to the destination peer to the message route.
4. Starting at the last peer ID listed in the message route, check if I have a direct connection to any of the listed peer IDs. If so, remove all of the peer IDs before that peer in the message route and forward the message to directly connected peer.

5. If no direct connection exists to any of the peer Ids listed in the message route then forward the message to the first peer listed in the message route.

5.13.6 Rendezvous Protocol

In JXTA, a rendezvous peer provides simple peers in private networks with the capability to broadcast messages to other members of a peer group outside the private network.This functionality is independent of the underlying network transport, allowing message propagation over transports that don't support multicast or broadcast capabilities. The Rendezvous Protocol (RVP) is used for propagation of messages within a peer group.

Before a peer can use a rendezvous peer to propagate messages, it must connect to the rendezvous peer and obtain a lease.A lease specifies the amount of time that the peer requesting a connection to the rendezvous peer is allowed to use the rendezvous peer before it must renew the connection lease.To handle the interactions required to provide this functionality, the RVP defines three message formats:

- **Lease Request Message**—A message format used by a peer to request a connection lease to the rendezvous peer
- **Lease Granted Message**—A message format used by the rendezvous peer to approve a peer's Lease Request Message and provide the length of the lease
- **Lease Cancel Message**—A message format used by a peer to disconnect from the rendezvous peer

Unlike previous protocols, these messages are not specifically defined in terms of XML; instead, they are defined in terms of message elements. As in XML, message elements consist of a name and the contents of the element, and they
can be nested.

The RVP provides mechanisms which enable propagation of messages to be performed in a controlled and efficient way. The RVP is divided into three parts;

- The protocol used by the Rendezvous Peers to organize themselves, also known as the PeerView protocol.
- The protocol used by client peers to register interest in receiving propagation messages, a simple lease protocol.
- The protocol used for propagating messages to the peers which have expressed an interest in the destination address. The message propagation protocol is the only protocol which all participants must implement

5.14 OSGI (IAN)

The OSGi Service Platform facilitates the componentization of software modules and applications and assures interoperability of applications and services over a variety of networked devices. Building systems from in-house and off-the-shelf OSGi modules increases development productivity and makes them much easier to modify and evolve.
The java implementation of the jxta framework (JXSE 2.7) and and it's mobile version (JXME), support the OSGi framework by providing the bundles needed to export the necessary services for the various system components.
The integration of the Apache Felix (implementation of the OSGi framework) and the JXSE

(implementation of the jxta framework) is very easy and it provides us with the ability to start, stop and update the jxta framework and the system components at runtime without stopping the whole system.
Apache Felix provides us with a number useful bundles out of the box, such as an http server, a command line tool to be used for managing the bundles installed, logging, security etc.

6 Conclusions

This document describes the work has been done in Task 3.4 of the project Peerassit entitled "Peer-to-peer overlay network selection". In this task, we have analyzed the existing platforms for building P2P networks. All candidates platforms have been evaluated in order to select the most appropriate one as a base technology for P2P. Platforms evaluation has been conducted considering the entire set of PeerAssist requirements including functional and non functional. From these requirements special attention has been paid to those that are mainly related to networking functions such as connectivity, communication grouping, interoperability, openness, serviceability, security, trust, scalability, efficiency, etc. The most appropriate technology for PeerAssist is JXTA.

JXTA provides a common set of open protocols backed with open source reference implementations for developing P2P applications. The JXTA protocols standardize the manner in which peers: (I) discover each other; (ii) self-organize into peer groups; (iii) advertise and discover network resources, (iv) communicate with each other; and (v) monitor each other. The JXTA protocols are designed to be independent of programming languages and transport protocols Using JXTA technology, developers can write networked, interoperable applications that can:

- Find other peers on the network with dynamic discovery across firewalls and NATs
- Easily share resources with anyone across the network
- Create a group of peers that provide a service
- Monitor peer activities remotely
- Securely communicate with other peers on the network

JXTA has been implemented in a small scale testbed for peerassist. The basic functionality of JXTA has been tested and possible improvements and enhancements has been drawn.

7 References

[1] B. Ford *et al.*, "Persistent Personal Names for Globally Connected Mobile Devices". in *OSDI*, November 2006.

[2] Adolfo Rodriguez, Charles Killian, Sooraj Bhat, Dejan Kostic´, MACEDON: methodology for automatically creating, evaluating, and designing overlay networks, in: Proc. NSDI'04, 2004, pp. 267–280.

[3] "SourceForge: Project Info – The Peer-to-Peer Trusted Library." Version 0.2. April 5, 2001. URL: http://sourceforge.net/projects/ptptl (3 July 2001).

[4] "The Peer-to-Peer Trusted Library (Release 0.2) README." Version 0.2. April 5, 2001. URL: http://sourceforge.net/docman/display_doc.php?docid=3851&group_id=19950 (3 July 2001)

[5] Taylor, Ian J. From P2P to Web Services and Grids - Peers in a Client/Server World. Springer, 2005

[6] TheMACEDONproject,Available from:<http://macedon.ucsd.edu/>.

[7] The Mace project, Available from: <http://mace.ucsd.edu>

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

[9] Hughes, D., Warren, I., Coulson, G., AGnuS: the altruistic Gnutella server. In the proceedings of IEEE P2P 2003, Linkoping, Sweden, September, 2003

[10] Foster, I., Iamnitchi, A., On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In the proceedings of IPTPS'03, Berkeley, CA, USA, February 2003

[11] Nagaraja, K., Rollins, S., Khambatti, M., Looking beyond the Legacy of Napster and Gnutella. In IEEE Distributed Systems Online, vol. 7, no. 3, 2006

[12] The EC funded P2P ARCHITECT project (IST-2001-32708). More information can be found at the URL http://www.atc.gr/p2p_architect/index.htm

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

[14] Halepovic, E., Deters, R., Building a P2P Forum System with JXTA. In the proceedings of P2P 2002, Linkoping, Sweden, 2002.

[15] The Gnutella protocol specification v0.4. Clip2 Distributed Search Services. Available from http://www9.limewire.com/developer/gnutella protocol 0.4.pdf

[16] Jtella. More information and the latest version of Jtella can be found at http://polo.lancs.ac.uk/p2p/JTella/jtella.htm

[17] Accord. More information about the Accord project can be find at https://accord.dev.java.net/

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

[19] Dabek, F. et al, Towards a common API for structured P2P overlays. In the proceedings of IPTPS'03, Berkeley, CA, February, 2003

[20] Portmann, M., Ardon, S., Senac, P., Seneviratne, A., PROST: A Programmable

Structured Peer-to-Peer Overlay Network. In the proceedings of IEEE P2P 2004, Zurich, Switzerland, 2004.

[21] Open Overlays Project. E-Science Project (EPSRC BR/S68514/01 & GR/S68521/01). More information can be found at http://www.comp.lancs.ac.uk/computing/research/mpg/projects/openoverlays/index.htm

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

[23] A. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility", 18th ACM SOSP'01, Lake Louise, Alberta, Canada, October 2001.

[24] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment", SOSP'03,Lake Bolton, New York, October, 2003.

[25] Easy Entry Library (EZEL) for JXTA. http://ezel.jxta.org/.

[26] Mandar Kelaskar, Vincent Matossian, Preeti Mehra, Dennis Paul, and Manish Parashar. A study of discovery mechanisms for peer-to-peer applications. In *CCGRID*, pages 444–445, 2002.

[27] In suk Kim, Yong hyeog Kang, and Young Ik Eom. An efficient contents discovery mechanism in pure p2p environments. In *GCC (1)*, pages 420–427, 2003.

[28] Dimitrios Tsoumakos and Nick Roussopoulos. A comparison of peer-to-peer search methods. In *WebDB*, pages 61–66, 2003.

[29] Beverly Yang and Hector Garcia-Molina. Improving search in peer-to-peer networks. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 5. IEEE Computer Society, 2002.

[30] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46+, 2001.

[31] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network, 2000.

[32] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[33] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.