

# A Framework for P2P Application Development

*James Walkerdine<sup>1</sup>, Danny Hughes<sup>1</sup>, Paul Rayson<sup>1</sup>, John Simms<sup>2</sup>, Kiel Gilleade<sup>2</sup>  
John Mariani<sup>1</sup>, Ian Sommerville<sup>3</sup>*

<sup>1</sup>{walkerdi, danny, paul, jam} @comp.lancs.ac.uk

<sup>2</sup>{j.simms, k.gilleade} @lancaster.ac.uk

Computing Department

InfoLab 21

Lancaster University

Lancaster

LA1 4WA

UK

<sup>3</sup>ifs@cs.st-andrews.ac.uk

School of Computer Science

Jack Cole Building

North Haugh

St Andrews

KY16 9SX

UK

## **Point of contact:**

James Walkerdine

Computing Department

InfoLab 21

Lancaster University

Lancaster

LA1 4WA

UK

**Email:** walkerdi@comp.lancs.ac.uk

**Telephone:** +44 (0) 1524 510352

**Fax:** +44 (0) 1524 510492

## **Abstract**

Although Peer-to-Peer (P2P) computing has become increasingly popular over recent years, there still exist only a very small number of application domains that have exploited it on a large scale. This can be attributed to a number of reasons including the rapid evolution of P2P technologies, coupled with their often-complex nature. This paper describes an implemented abstraction framework that seeks to aid developers in building P2P applications. A selection of example P2P applications that have been developed using this framework are also presented.

**Keywords:** Peer-to-Peer, Abstraction, Application development

## **1 Introduction**

There is no denying that P2P has become an increasingly popular technology over the last decade. As well as the development of well known P2P applications such as Napster [1], ICQ [2], MSN Messenger [3] and SETI@home [4], there has also been considerable research within the area. In particular, this has focused on aspects such as improving routing strategies, new overlay structures, and tackling quality of service issues [5, 6, 7].

Despite these significant developments the actual number of application domains that have exploited this technology has remained limited, typically focusing on groupware such as instant messaging, forums, shared workspaces and file sharing. Although P2P technology can obviously provide significant benefits to such domains, it does seem that the potential of P2P is not being met.

There are a number of technical and non-technical reasons why P2P technology has not been more widely adopted. These include the lack of practical business models, missing incentives to provide computational resources, security concerns, lack of design support for developers and the heterogeneity and complexity of the underlying technology. Some of these issues are beyond the scope of our work, however we believe we that by providing greater support to the developers of P2P applications the use of the technology can be encouraged.

The vast majority of P2P research has focused on the low level aspects of P2P technology (routing strategies, search algorithms, etc). As Foster et al. [8] highlight, little work has been carried out on the actual design process for P2P application development. These findings are also corroborated by a recent study that showed that less than 15% of recent research at a leading P2P forum (IPTPS) has focused on actual P2P applications [9]. The authors of the study raised the lack of applications as a concern and went on to say that "all the (core P2P) research done will receive neither feedback nor validation unless there's an active set of clients for the technology". To an extent this has been addressed within the EC funded P2P ARCHITECT [10] project that developed a methodology, reference architectures, notations and guidelines for P2P application development, with a particular focus on business environments. Even though these developments are significant, they still only represent a small amount of work within the overall P2P area.

Similarly, although there has been, and continues to be, P2P technological developments, they often evolve rapidly or require deep understanding of their workings before they can be successfully applied. In our own experiences with Sun's JXTA API [11], we found we needed to gain a clear understanding of the JXTA concepts and workings before we could success-fully build a P2P application - a task that involved several months of effort. Similar experiences have also been reported by others [12].

It is not surprising that given the complexity of existing P2P technology and the lack of support for the design process, developers can find the prospect of building P2P applications a daunting task. Ultimately this not only has a detrimental effect on the research area as a whole, with its potential exploitation being reduced, but also on its uptake within industrial domains with businesses being less likely to commit the money and effort in order to utilise it. The underlying P2P technologies also suffer, as higher-level application development plays a vital part in informing their further development and refinement.

This paper presents the P2P Application Framework, an abstraction model and implementation that hides the underlying complexity inherent in developing P2P applications by providing a set of generic and protocol independent application oriented services. These services represent common P2P functionality that developers would typically otherwise need to implement themselves. We believe that

the provision of an easy-to-use architecture will enable developers to concentrate more on aspects of application development other than the peer-to-peer characteristics. As a result it is hoped that this will encourage the development of more applications (and potentially in different application domains), and also make the use of P2P technologies more feasible within business environments.

The paper begins by providing an overview of the P2P Application Framework, describing its overall structure and key concepts. Our initial implementation of the framework is then presented, and an example is provided to illustrate the framework in use. The paper then ends with a showcase of the applications that have developed using the abstraction framework.

## **2 Related Work**

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 the simple network-oriented functionality provided by Gnutella [13] APIs such as Jtella[14], and Chord[6] APIs such as Accord[15], to the more complex and application-oriented functionality provided by JXTA[11] and Groove[16]. Each of these APIs require that developers possess detailed understanding of the underlying P2P technology. Due to the widely recognised [8] 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 [17], PROST [18] and the Open Overlays project [19].

The Common API for Structured P2P Systems provides a consistent abstraction for structured overlays such as Pastry [20], Past [21] and SplitStream [22]. The Common API for Structured P2P Systems provides three different abstractions, one for each major area of systems 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[23] 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.

### **3 The P2P Application Framework**

The P2P Application Framework is a mechanism to help developers in building (types of) P2P applications. It achieves this by providing layers of abstraction that further isolate the developer from the complexities of the underlying P2P technology. A consequence of this is that the developers who use the framework do not need to understand the specifics of how the P2P technology functions, and instead they can focus purely on building the applications (referred to as plug-ins) that will utilise it. Furthermore, the additional abstraction means that developed plug-ins are independent of the underlying P2P protocol/substrate that is used. So for example, by using different implementations of the framework the same Instant Messenger plug-in could operate over Pastry or JXTA without being changed. This can allow for situations such as where one company may use JXTA due its security and network configuration (NAT, etc) handling benefits, where as another may use Pastry due to its low overhead for peer communication.

Obviously the framework will not be suitable for all types of P2P application, in particular those that require greater control of the network or how communication between peers is performed (e.g. real-time media streaming or application-level multicast). However, even in these cases the framework can still have an important role to play, for example, for use in small-scale prototyping, interface and interaction design. As a general rule, the applications that would most benefit from the framework would be those that do not possess hard Quality of Service requirements nor require control of low-level networking functionality. Examples of applications which would benefit greatly from the P2P application framework include high-level groupware, resource sharing and distributed computation, though this list is by no means exhaustive.

This generic nature and flexibility of the framework makes it different from other related developments such as Groove, which are more rigid due to the fact that they enforce the use of their own underlying P2P technology, and place limitations on what P2P applications can be developed. Likewise the P2P Application Framework is also different from developments such as PROST that only focus on network development rather than application development. Consequently the framework should be viewed more as an abstraction of functionality to aid in application development, rather than as a middleware layer that supports the construction and manipulation of overlay networks.

To assist the developer the framework provides a set of generically accessible services. The initial services we provide stem from common functionality that we have identified within many existing P2P applications, for example message communication or resource searching. Over time we envision that additional services will be added to support different application domains, and bringing with it the possibility of also allowing the framework to be re-configured by the application. Within the framework, unlike typical middleware that tends to operate at a low level, these services operate at a higher level and are provided specifically for use by GUI styled applications.

The design of the P2P Application Framework is such that it can be applied over different types of network topology. Our initial implementation instance (discussed later) has been built on top of a semi-centralised structure to support our work carried out during the P2P ARCHITECT project [10]. There is no reason, however, why a fully decentralised implementation could not be developed. In either case, the application plug-ins that have been built on top would not need to be changed.

Because the P2P Application Framework is a light weight API based abstraction, the overhead from using it (in terms of additional processing occurring behind the scenes) is minimised. The difference is that functionality that would normally have to be built into the application is instead moved down into the framework. This results in negligible performance loss, but does result in the developer having less control of the underlying network as a consequence of certain design decisions having already been made for them.

Figure 1 shows the structure of an individual peer within the P2P Application Framework model. All peers within the network would possess such a structure. A breakdown of this model will now be provided.

**P2P Protocol/Substrate** - this layer within the model represents the underlying P2P technology that is being used within the system. This could be, for example, JXTA, Pastry or Gnutella.

**Protocol/Substrate specific Interface Layer** - a key characteristic of the P2P Application Framework is that it is sufficiently generic and abstract that it can be utilised on top of different types of P2P protocol/system. In order for the framework to communicate with the underlying technology there needs to be a special interface layer that bridges the gap between the two. For example, if it were desired for the framework to be built on top of Pastry then an interface layer that is tailored for Pastry would be required to allow the Application Development Layer to make use of the protocol.

The difficulty in building interface layers will be dependent on the individual protocol/substrate, however existing work has already shown such layers can be created for a number of protocol types [24] without significantly compromising their functionality.

In many cases functionality provided by the protocol/substrate can be directly mapped to the services provided within the Application Development Layer (for example, message communication). In the instances where this is not the case then the additional functionality needs to be built into the interface layer itself, utilising whatever functionality that already exists (for example, developing awareness support by implementing a mechanism built around existing message communication functionality).

As will be discussed later, our initial implementation instance of the framework has been based on JXTA and so in this case the interface layer supports the communication between the application development layer and JXTA.

**Application Development Layer (AD Layer)** - this layer sits on top of the Interface Layer and provides the foundation for application development. The layer provides a number of services which are useful to a broad range of P2P applications. Rather than attempt to provide an exhaustive set of services, the services provided by the framework have been derived from the functionality that is common to many existing P2P applications. Of course additional functionality can be added at a later date.

- *Message Communication* - the layer handles all communication aspects within the P2P system. Developers do not need to worry about how to construct messages specific to the underlying P2P technology or how peers within the network are addressed. Instead messages



are simply comprised of attribute name: value pairs (e.g., 'Message: Hi!'), and are sent to targets based on their User ID. If a target's User ID is not known, it can be identified by using relevant discovery mechanisms within the Protocol/Substrate Specific Interface Layer. For example this could involve interrogating an index peer, or by doing a propagated search.

- *Search* - the layer provides generic search facilities that allow plug-ins and users to search the network for resources and other users. Again, the mechanism used to achieve this is dependent on the underlying P2P technology.
- *Awareness* - the layer provides awareness facilities that allow peers and their respective plug-ins to stay up to date on the status of users and resources they have an interest in [25].
- *Monitoring* - the layer provides support for monitoring capabilities which allows users/plugin to access information about the P2P system. This could, for example, be information about peer communication, peer resources, peer availability or peer location.
- *File Transfer* - the layer provides generic facilities for file transfer between peers. Obviously the capabilities of the file transfer support will be dependent on the underlying P2P technology that is used. This means that, depending on the technology, NAT and firewalls may be an issue (for example, if using Gnutella).
- *Favourites List* - the layer is able to maintain a favourites list for each user. This can be used to capture favourite/interested users and also potentially some types of re-sources (for example, shared disk space). Users can use the contents of the favourites list as shortcuts or as reminders.
- *Front-end* - the layer provides a general front-end in which the user is able to manage and activate developed plug-ins, access the layers search service, and manage their user details (including favourites list).

**User** - as with some existing P2P systems such as Instant Messengers (ICQ, MSN Messenger, etc), within the framework it is the user rather than the peer that is the unique identity within the system. When a new user first makes use of the framework they are assigned a unique ID by the Protocol/Substrate specific Interface Layer. This ID maps directly onto the addressing scheme used within the underlying technology.

**Plug-ins** - a key concept within the framework is that of plug-ins. A plug-in can essentially be thought of as a P2P application (such as an instant messenger, file sharing tool, etc) that is built to make use of the AD Layer. Plug-ins draw upon the services provided by the layer in order to carry out their operation. Because a significant amount of the required functionality is already catered for, it allows for plug-ins to be developed with less effort, requiring less code than their standalone equivalents.

A flexible and independent relationship exists between the plug-ins and AD Layer as a result of a standardised two-stranded communication protocol. This is comprised of a:

- *Common Plug-in Interface* - All plug-ins that wish to be a part of the framework must conform to a specified interface. This ensures that suitable access points exist through which the AD Layer can interrogate the plug-in. Such information could range from the plug-in's name through to what resources it contributes to the network. These access points also provide the means by which the layer passes along messages relevant to the plug-in that it has received. Implementing the interface would involve the plug-in providing a set of publicly accessible methods.
- *Application Development Layer API* - In order to provide a means for the plug-ins to access the services that are provided, the AD Layer possesses an API. This API provides plug-ins (and their developers) with a simple way to access the AD Layer's functionality. This would typically include access to aforementioned services and possibly utility methods to help plug-in developers.

Because the relationship between the plug-ins and AD Layer is quite generic and loosely coupled, a fair degree of adaptability is provided for (as will be discussed later).

Plug-ins possess a unique ID that is consistent across the whole system. These IDs are assigned by the network during the plug-in's creation (for example, by a central authority or from a secure hash (i.e. a nodes IP address)), and help to determine where messages are from and where they should be routed. For example, a message could be sent to a plug-in (Plug-in ID:2) on a users machine (User ID: 5).

**Resources** - plug-ins are able to contribute resources to the network (for example, a file sharing plug-in may contribute MP3 files). The term 'resource' is used in a very broad sense in that a resource could be, in theory, anything that a plug-in can contribute. Furthermore a contribution does not necessarily need to be a physical entity (for example, a file), but could, for example, also represent the willingness for that peer/user to take part in a certain activity (i.e. a P2P based game).

**Resource Types** - to help identify resources within the network, they can be assigned a Resource Type. A Resource Type essentially represents a broad classification of the resource. For example, an MP3 resource could be assigned an 'Audio' Resource Type. In a sense they are similar to Mime Types, although they are not just restricted to file based resources and are more open ended to cope with the broader range of resources.

By semantically classifying resources in this way it becomes easier for the AD Layer and other plug-ins to identify and utilise the resources that exist within the network. This allows the possibility of a resource that is being made available by one plug-in to be then utilised by another plug-in. For example (figure 2), the user of Peer A makes use of the *file sharing plug-in* and contributes an 'Audio' Resource Type resource to the network. The user of Peer B is doing a search for audio files. He discovers Peer A's audio file and plays it with his *audio player plug-in*.

In order to keep aware of what plug-in:resource type relationships exist, the AD Layer interrogates each plug-in to discover what Resource Types it is interested in. For example, the audio player plug-in would inform the framework that it is interested in 'Audio' Resource Types. Again this provides the framework with a degree of flexibility.

A key factor with the relationships that exist between the plug-ins, its resources and the AD Layer is that of adaptability. Because there exists a generic and loosely coupled connection between these three entities it means that plug-ins and resources can be added and removed from the system at will, with the AD Layer being able to adjust itself accordingly. For example, a user may have located a resource on the network of 'Audio' Resource Type. However, they do not possess a suitable plug-in to handle

that type and the AD Layer indicates this to them. After installing the audio player plug-in, the AD Layer will now highlight that this can handle the particular resource.

Our implementation instance for the framework highlights other ways in which this adaptability can be utilised, and this will be discussed in the following section.

## **4 Implementing the P2P Application Framework**

An initial implementation instance of the P2P Application Framework has been developed and fully working releases have been made publicly available. The framework has been used within a number of separate projects, resulting in the development of a range of application plug-ins. A summary of these, along with experiences in using the framework are provided later in this paper.

Development work was spread over a year, and built on our experiences within the P2P ARCHITECT project. This project focused on supporting P2P developments for use within industry, where servers would play a crucial role. To accommodate this, our initial implementation instance was based on a semi-centralised topology (where there exists a single index peer, as used within systems such as Napster) and used JXTA as the underlying protocol. Our future work will involve adding support for more decentralised topologies.

The framework itself was written in Java and makes use of reflection to provide on the fly interrogation of the available plug-ins (an example is provided in section 5). Again this reinforces an adaptable nature by ensuring that no plug-ins or resources are hardwired into the framework itself. Developed plug-ins are stored within a directory structure that is searched by the framework on initialisation. Any found plug-ins are loaded and instantiated.

The index peer makes use of a MySQL database in order to capture details about the state of the network and as a result is able to support the services provided by the AD Layer. This includes Napster styled resource/user searching, awareness/monitoring mechanisms and the allocation of User ID's. Currently the implementation is able to monitor various aspects of a peer including available resources,

CPU performance, and networking information. Such information can, in turn, be made available to developed plug-ins.

The current stable release of the framework possesses all the services that have been previously described. So far eight plug-ins have been developed that utilise various aspects of the framework, with more still in development.

#### *Plug-in Interface and the AD Layer API*

As previously discussed communication between plug-ins and the AD layer is based around two channels. All plug-ins that are built to make use of our framework must implement a set interface that allows the AD layer to interrogate them. In turn the AD layer possesses an API which the plug-ins can use to the various services. Figure 4 summarises some of the key methods of the interface and API. A full list of the methods/API can be found on Lancaster's P2P website (<http://polo.lancs.ac.uk/p2p>).

Section 4 provides examples of how the developer can use such methods.

#### *P2P Application Framework Front-end*

It was desired to develop a front-end for the framework that was similar in appearance to many existing instant messenger applications, primarily focusing around a buddy list (a favourites list), but with buttons to access the framework's functionality and installed plug-ins. Figure 5 provides a screenshot of the framework implementation's front-end. Of course, the GUI for the framework could have been implemented differently and, if needed, could be hidden altogether from the user.

In addition to the features common to instant messenger applications (for example, user awareness within the favourites list), the screenshot also illustrates the plug-ins that have been loaded into the framework, as well as showing one way in which the framework implementation can adapt to the plug-ins. In this instance the options that appear in the pop-up menu reflect what is specified by the plug-ins (i.e., whether or not the individual plug-ins desire to add a menu entry). As new plug-ins are added or old ones removed the options within the pop-up menu adapt accordingly.

### *Searching with the framework*

Our implementation allows both plug-ins and users to search the network for other users and available resources. When a search is performed a query is sent to the index peer, which interrogates its associated database. Figure 6 provides a screenshot of the implementations search facilities. In this example the user has performed a general resource search and so a list of all currently available resources is returned. As can be seen, the Resource Type for each resource is displayed, along with an indication of the plug-in that is providing the resource (if it is known by the user's framework installation). By selecting a resource the user can bring up a menu that displays all the plug-ins that can access this resource (based on which plug-ins have registered an interest in that Resource Type). Selecting a plug-in would then invoke it with the selected resource. In this example, by selecting the File Sharing plug-in the user can download the file to their peer.

As well as resource searching, our implementation also supports user searching. This not only includes searching via name, nickname and ID, but also via expertise. This is as a result of allowing users to describe their expertise during initial user registration. The idea of adding such a search feature stemmed from our P2P Helpdesk prototype application that was developed early during the P2P ARCHITECT project.

The implementation's API also provides a search method that allows plug-ins to access the frameworks search service in a manner that is transparent to the user.

### *Implementing Framework instances that utilise different P2P protocols*

Although this instance of the framework has been implemented on JXTA using a semi-centralised topology, the framework is portable across a range of architectures and protocols. The properties and functionality provided by the underlying P2P protocol will clearly affect the performance of application-level functionality provided by the framework, however this functionality remains consistent, despite potentially heterogeneous underlying networks.

Considering a DHT-based implementation of the framework, a combination of technologies can be used to provide the equivalent functionality. For example, the message-passing service from the X

layer, can be implemented directly using a DHT such as Pastry [20]. Furthermore, the search service from the AD layer could be implemented using a combination of a resilient DHT-based file-store such as PAST [21], which is capable of providing a highly available and secure distributed file-store and a distributed indexing service, such as that used in OverCite [33], this would effectively allow for the provision of the same capabilities currently provided by our JXTA based implementation. Here the effect upon the applications performance would be mixed. The framework's lookup service would potentially be more resilient and scalable (as this architecture has no single point of failure), though search functionality may be less reliable due to the index being distributed across standard peers (which may disconnect or fail).

Considering an unstructured implementation of the framework, for example over Gnutella [13], the search service can be easily re-implemented as a wrapper for Gnutella's broadcast resource discovery scheme, which can also be used to provide lookup of nodes. This service also effectively allows the lookup of nodes (for example by assigning each node a unique identifier and using specially formatted queries to locate them). Here the effect upon the application would be quite significant. In a Gnutella network, the search horizon [34] effect means not all nodes can reach each other with search messages, leading to unreliability for those applications which depend upon a consistent contact list, such as instant messaging and group-working.

Any re-implementation will affect the application developer in terms of the performance of the underlying P2P as in the examples above, according to the dependability properties of the specific P2P network in question [35].

## **5 Using the P2P Application Framework**

In order to demonstrate the use of the framework from the developers' perspective, we will now provide a simple breakdown of our Instant Messenger plug-in. This example will focus on highlighting the aspects in which the plug-in and AD Layer interact.

*Case Study: Instant Messenger Plug-in*

The Instant Messenger plug-in was the first plug-in to be developed for use within the P2P Application Framework. It was intended that it would provide simple text based instant messaging support, though over time this has been extended. The current version of the plug-in allows images and sounds to be also sent within a message (as illustrated in figure 7).

The plug-in interacts with the framework in a number of ways:

- Plug-in activation
- Sending of a message
- Receiving of a message
- User status changes

#### *Activating the plug-in*

During the development, it was decided that it should only be possible to initialise an instant messaging conversation with users who were currently on your favourites list. Consequently when the plug-in is first initialised and the AD Layer interrogates it, the plug-in informs the layer to add a 'Send Message' entry to the pop-up menu (as illustrated in figure 5). When a user selects this menu entry the framework informs the plug-in which displays a messaging window similar to that shown in figure 7. As this plug-in is not interested in any resources around the network it does not register any Resource Type/Plug-in relationships with the framework. Examples of how the AD Layer is able to access the plug-in, and vice-versa, are provided in the following sections.

#### *Sending a message*

When a user wishes to send a message the plug-in takes the message contents (text, image, sound) and wraps it up into a structured object (in this case a hash map). This also contains additional details about the sender and target of the message (for example, the plug-ins involved). A method within the AD Layer API is then called passing the object and User ID of the target as parameters.

*AD.sendMessage(targetuserID, messageObject);*



The framework then sends the message to the target.

#### *Receiving a message*

When the AD Layer of a peer receives a message it first determines the destination plug-in by looking at the relevant name:value pair within the message object. In this case the layer routes forward any messages containing the ID for the Instant Messenger plug-in. To achieve this, the layer uses reflection to call the method '*messageArrived*' (which is part of the common plug-in interface) within the Instant Messenger plug-in. This is illustrated in Figure 8.

When the Instant Messenger plug-in receives a message it deconstructs it and acts accordingly based on its content. Typically this would involve taking the contents of the message and displaying it within the relevant messaging window.

#### *Reacting to user status changes*

Obviously a user's state can change and other users need to be kept informed of this. When the AD Layer of a peer receives a user status change message it alerts all installed plug-ins. Again reflection is used to call the '*changeOfPeerStatus*' method within each plug-in.

When this method is called within the Instant Messenger plug-in it updates the status within the relevant messaging window (shown in top right hand corner).

As can be seen, although the Instant Messenger plug-in and AD Layer only communicate with each other for a few operations, these operations represent a significant part of the plug-in's overall functionality. Because the framework takes care of this functionality the developer can instead focus on the higher-level application functionality of the plug-in (such as GUI or providing additional features). A similar interaction structure is used by the other plug-ins that have been developed, allowing them to draw upon the frameworks functionality where required.

## **6 Evaluating the Framework**

The benefits of the P2P Application Framework have been evaluated using a combination of quantitative and qualitative approaches. Firstly the suitability of the framework for supporting diverse P2P applications is illustrated through case-study implementations. Secondly, by comparison with JXTA-based applications, we illustrate the code-length reductions that may be achieved using this abstraction.

## **6.1 Plug-in Developments**

A pre-release of the implemented framework was released to developers in early 2004. This allowed for testing, refinement of the AD Layer API and development to begin on an initial set of plug-ins.

The first full release was made publicly available in the summer of 2004, accompanied with API documentation, user guide and examples. To date a range of applications have been developed using the framework including, an Instant Messenger, Distributed Video Encoder [26], distributed virtual world and a novel P2P based Digital Library [27]. Not only has the framework been used locally (including by BSc and MSc students), there has also been interest expressed from external institutions. Table 1 lists the plug-ins that have been developed and also the Resource Types that have so far been defined.

The following sections provide an overview of some of the plug-ins that have been successfully developed and deployed by using the P2P Application Framework.

### ***6.1.1 Distributed Video Encoder***

Current video encoding technologies tend to focus on single machine solutions; the Distributed Video Encoder (DVE) plug-in [26] sought to exploit available resources (spare CPU cycles) within a P2P network to distribute the load. A key benefit of this is that faster video encoding can be achieved than single-machine solutions.

The DVE plug-in was developed using the P2P Application Framework and makes use of its search, file sharing and resource-awareness services to help identify available peers, and then to distribute and re-collected video files. The Java Native Interface (JNI) is used together with the Microsoft Windows

Media Encoder SDK to implement the video encoding functionality. Figure 9 summarises the architecture of the DVE.

Evaluation of the DVE plug-in found large performance increases over single machine encoding. A more detailed description of the DVE can be found in [26]

### ***6.1.2 Digital Library***

The Digital Library (DL) plug-in [27] was developed as part of the P2P-4-DL project that aimed to investigate and build a DL system that would operate over a P2P structure. With this DL, rather than storing digital objects centrally they remained the responsibility of the individual peers that provided them. This allowed the system to utilise network resources more efficiently as well as providing users with a greater sense of control over the digital objects they shared. The DL prototype also drew upon Natural Language Processing (NLP) techniques in an attempt to increase the usability of the system – keywords for documents could be automatically extracted from the text.

The DL plug-in made use of the P2P Application Framework to help with the activities of searching for documents, transferring of documents between peers, and awareness of when these documents were available (on-line). The DL was also able to recommend papers to the user and this involved using the framework to perform searches that were transparent to the user. Figure 10 provides a screenshot of the DL in use. A more detailed description of the DL can be found in [27].

### ***6.1.3 Audio streaming***

The Audio Streaming plug-in was developed as part of a student's final year project and involved building an application that would allow peers to stream audio to one another. By making use of the plug-in, users were able to:

- Register their audio streams as being available on the P2P network
- Search for audio streams that are currently available on the network
- Access a peer's play list
- Provide feedback to the streaming peer, on what is being played

The plug-in worked in tangent with Shoutcast [28] and WinAmp [29] for creating a stream, and the JavaZoom MP3SPI [30] for playing it. The P2P Application Framework provided support for the discovering and recommending of audio streams, and also for the initial handshake used to establish a connection between the two peers. The actual streaming was done independently of the framework, as it could not be guaranteed that quality of service could be maintained. As part of the project, the audio streaming plug-in was deployed and used over a number of peers

#### **6.1.4 Net World**

The most recent plug-in development has involved building a distributed virtual world based on a P2P network. The goal was to develop a virtual game world, Net World [31], in which the unpredictable nature of P2P networks could be exploited to design and evolve the world. In turn this would provide users with a more novel gaming experience (similar approaches have been used for mobile device games [32]).

For our Net World implementation it was decided that the virtual world should closely correlate with the peer-to-peer architecture. The game world is therefore composed of an extensible number of ‘zones’ each hosted by a peer on the underlying network. Connections between zones represent network links between peers and these are derived from user *favourites lists*. If a user or resource is on a favourites list, then a connection to the peer where they are based is created. In this way a user may only access zones hosted by peer that they are aware of (either via a user or a resource). Access is possible via a number of corridors each representing a network link. Each zone contains a simple puzzle, configured by the user of the host peer. The objective of the game is to traverse the ever-changing network graph while solving the customised puzzles.

Net World was developed as a plug-in allowing it to make use of the P2P Application Framework’s monitoring mechanisms. A number of properties, universal to all nodes, could be measured and these could then be used to influence the virtual world in a number of ways.

A peer’s *Node properties* are used to influence zone design:

- *Available memory* defines the size of the zone

- *CPU performance* defines the time-limit for the contained puzzle

A peer's *Link properties* are used to influence the design of connecting corridors.

- *Hop-count* defines the length of connecting corridors.
- *Throughput* defines how fast a user can move down a corridor
- *Loss and delay* define the look and feel of the corridors, such that low quality links generate grim looking corridors

Figure 11 provides a screenshot of our Net World implementation and highlights how some of these properties influence the virtual world.

Net World is an ongoing project with new features, such as allowing multiple users within the same world, still being developed. User trials are also to be undertaken to evaluate the effect that modelling the P2P properties can have upon a user's experience. A more detailed description of NetWorld can be found in [31].

#### **6.1.5 Other developments**

As well as the more sophisticated plug-ins that have been presented, initially a set of simple plug-ins were developed to test and help refine the P2P Application Framework. As well as the Instant Messenger that has been already discussed, these included a file-sharing application and a simple P2P based game. The former made use of the frameworks resource registration and file transfer functionality, whilst the latter made use of the frameworks messaging functionality in order to ensure concurrency between the two gaming peers.

## **6.2 Code Comparisons**

In general developers found the framework to considerably reduce the complexity of building P2P applications. Although this is difficult to quantify, we explored the extent to which the framework simplifies P2P application development through a series of code comparisons. Figure 12 shows two sections of code that are used for sending a message to another peer. In figure 12a the message is sent

using JXTA's programming API, in figure 12b the message is sent using the framework (which may use JXTA underneath). As can be seen it is significantly easier to achieve the same objective using the framework, not only in the amount of code required but also in reducing the need to understand all aspects of the underlying protocol. Table 2 provides code length comparisons for some of the plug-ins that have been developed with the framework, against the length when they have been built purely with JXTA. The varying percentage reductions reflect the fact that the different applications will possess different amounts of non-P2P code, depending on their purpose (the Instant Messenger and NetWorld required more user interface code, where as the DVE is largely a computational application).

To take this into account, the final three columns of the table illustrate the code sizes for only the P2P code within the applications. Again the Instant Messenger and NetWorld applications show smaller reductions, and this reflects the fact that their peer-to-peer communications are more complex and a greater processing of messages is required. Either way, the table illustrates how the P2P Application Framework reduces the amount of development work required.

As previously mentioned, because the framework only acts as an abstraction layer as opposed to a more sophisticated middleware there is negligible performance loss from using it in comparison to the standalone JXTA applications. Of course the overall performance is directly linked to the underlying P2P technology that is used, so although the framework might not have an impact on performance, switching to a different implementation instance (for example, one that is more decentralised) will. For example, discovery or search functionality might not be as efficient as that provided by our semi-centralised/JXTA implementation. However, examining the impact of the different underlying technologies is beyond the scope of this work and initial work has already been done elsewhere [35].

It is also our intention to carry out more detailed evaluations and testing of the framework. Such evaluations would most likely be qualitative in nature, with user feedback also helping to further refine the framework. We also intend to investigate the feasibility of performing quantitative evaluations, although evaluation based upon development time is difficult to perform objectively.

## 7 Conclusions and Future Work

This paper has presented the P2P Application Framework, a generic and flexible mechanism to assist developers in the building of P2P applications. Essentially the framework provides an abstraction that wraps around the (often complex) underlying technology and provides developers with a set of generic application centric services. A consequence of the framework is that developers can focus more on building P2P applications and, in turn, this will hopefully encourage the development of new types of P2P application (for example, P2P based Digital Libraries or P2P based games). Expanding the areas where P2P technology could be applied would in turn also further inform the development of the underlying P2P technologies. Furthermore, the additional abstraction allows for applications that have been developed using the framework to be utilised over different underlying P2P protocols/substrates.

Our initial implementation of the framework has been built for use over a semi-centralised network topology, with JXTA being the underlying protocol. This implementation has been made publicly available and used as the basis of a number of P2P application developments. A summary of these have been presented here. Initial experiences have shown that the framework can provide significant benefits to the P2P application development process.

The P2P Application Framework, itself, is a work in progress and will be further developed and refined. A key focus will be to extend the implementation and develop additional interface layers so that the framework can support different network topologies and underlying P2P protocols. Pastry will be used as the basis for a second interface layer, as it provides a significant difference in topology and protocol. We will be drawing upon techniques developed within the Open Overlays project [19] to help achieve this.

Additional plug-ins are also planned and it is hoped that as the framework matures a broad range of P2P applications will be produced. Future possible plug-ins include P2P based distributed database systems and dependability tools that can draw upon monitoring information. Feedback from plug-in development will also potentially result in the provision of new framework services.

The P2P Application Framework is not open source, but more information and executable downloads can be found at our departmental P2P website - <http://polo.lancs.ac.uk/p2p>

## References

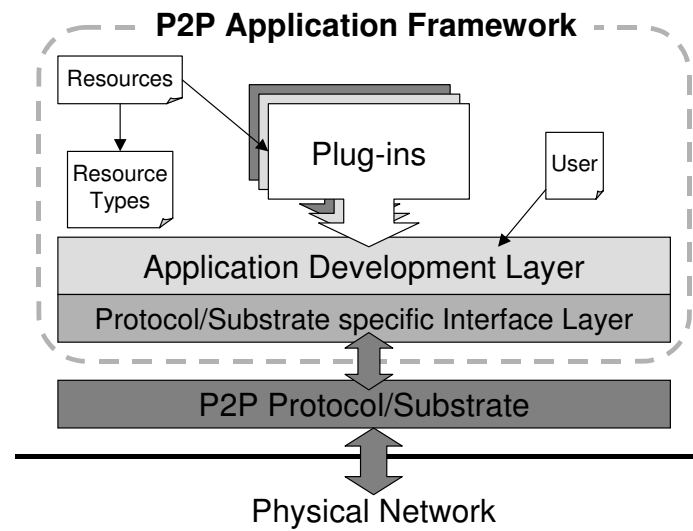
1. Napster. MP3 file sharing application. More information at the URL <http://www.napster.com>
2. ICQ. Instant Messenger Application. More information at the URL <http://www.icq.com>
3. MSN Messenger, Microsoft Corporation. Instant Messenger Application. More information at <http://specials.msn.com/ms/default.asp>
4. The SETI@home project. Distributed computation application. SETI. More information can be found at the URL <http://setiathome.ssl.berkeley.edu/>
5. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S., A Scalable Content Addressable Network. In Proceedings of the ACM SIGCOMM Symposium on Communication, Architecture, and Protocols, pages 161--172, San Diego, CA, U.S.A., August 2001
6. 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.
7. Hughes, D., Warren, I., Coulson, G., AGnuS: the altruistic Gnutella server. In the proceedings of IEEE P2P 2003, Linkoping, Sweden, September, 2003
8. 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
9. Nagaraja, K., Rollins, S., Khambatti, M., Looking beyond the Legacy of Napster and Gnutella. In IEEE Distributed Systems Online, vol. 7, no. 3, 2006
10. 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](http://www.atc.gr/p2p_architect/index.htm)
11. JXTA v2.0 Protocols Specification, Sun Microsystems, The Internet Society, 2001-2003. More information can be found at the URL <http://www.jxta.org>
12. Halepovic, E., Deters, R., Building a P2P Forum System with JXTA. In the proceedings of P2P 2002, Linkoping, Sweden, 2002.



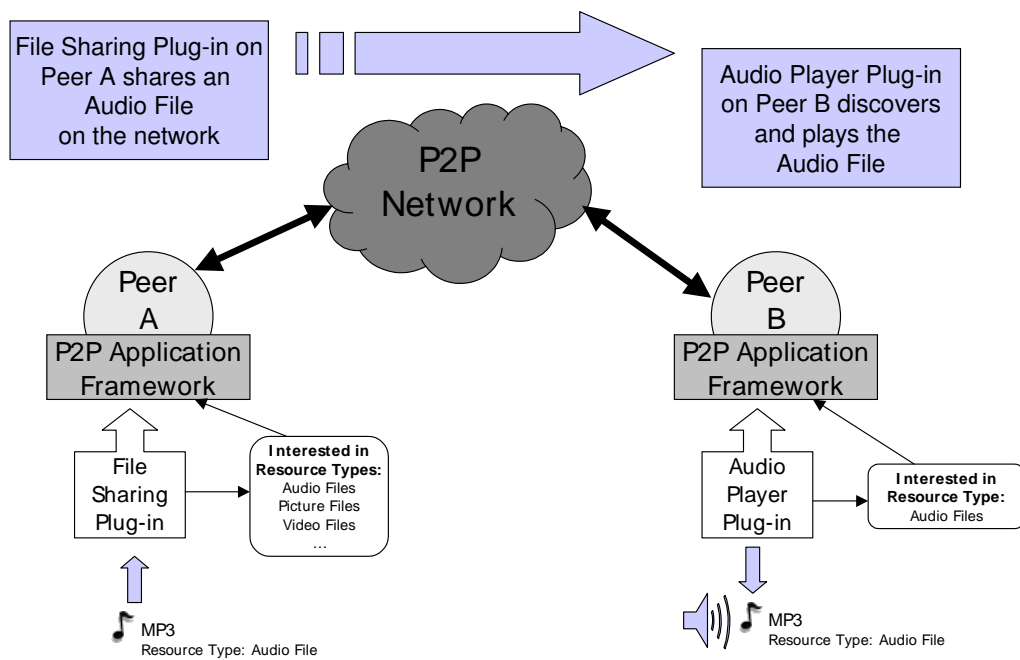
13. The Gnutella protocol specification v0.4. Clip2 Distributed Search Services. Available from [http://www9.limewire.com/developer/gnutella protocol 0.4.pdf](http://www9.limewire.com/developer/gnutella%20protocol%200.4.pdf)
14. Jtella. More information and the latest version of Jtella can be found at <http://polo.lancs.ac.uk/p2p/JTella/jtella.htm>
15. Accord. More information about the Accord project can be find at <https://accord.dev.java.net/>
16. Groove Peer Computing Platform, Groove Networks Inc., 2000. More information can be found at the URL <http://www.groove.net>
17. Dabek, F. et al, Towards a common API for structured P2P overlays. In the proceedings of IPTPS'03, Berkeley, CA, February, 2003
18. 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.
19. 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>
20. 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.
21. 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.
22. 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.
23. Easy Entry Library (EZEL) for JXTA. <http://ezel.jxta.org/>.
24. Coulson, G., Grace, P., Blair, G., Mathy, L., Duce, D., Cooper, C., Yeung, W. K., Cai, W., Towards a component-based middleware framework for configurable and reconfigurable grid computing, proceedings of ETNGRID-2004, Italy, June 2004.
25. Walkerdine, J., Melville, I., Sommerville, I., Designing for Presence within P2P Systems. Technical Report COMP-003-2004, Computing Department, Lancaster University, 2004
26. Hughes, D. and Walkerdine, J. (2005), Distributed Video Encoding Over A Peer-to-Peer Network. In the proceedings of PREP 2005, Lancaster, UK, 30th March - 1st April, 2005

27. Walkerdine, J., Rayson, P., P2P-4-DL: Digital Library over Peer-to-Peer. In the proceedings of IEEE P2P 2004, Zurich, Switzerland, 2004.
28. Shoutcast audio streaming system, Nullsoft. More information can be found at <http://www.shoutcast.com>
29. WinAmp audio-visual media player, Nullsoft. More information can be found at <http://www.winamp.com>
30. MP3SPI, Java Service Provider Interface that provides MP3 support for the Java platform, JavaZoom. More information can be found at <http://www.javazoom.net/mp3spi/mp3spi.html>
31. Hughes, D., Gilleade, K., Walkerdine, J., Mariani, J., Exploiting P2P in the Creation of Game Worlds. In the proceedings of ACM GDTW 2005, Liverpool, UK, 8th-9th November, 2005
32. Cheok, A, Fong, S., Goh, K., Yang, X., Liu, W., Farbiz, F., Human Pacman: A Mobile Entertainment System with Ubiquitous Computing and Tangible Interaction, Proceedings of the Fifth International Symposium on Human Computer Interaction with Mobile Devices and Services
33. Stribling, J., Li, J., Councill, I. G., Kaashoek, M. F., Morris, R., Overcite: A Distributed, Cooperative CiteSeer, Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI'06),
34. Adar, E., Huberman, B., Free Riding on Gnutella, [http://www.firstmonday.dk/issues/issue5\\_10/adar/index.html](http://www.firstmonday.dk/issues/issue5_10/adar/index.html) First Monday, Oct. 2000.
35. Walkerdine, J., Melville, L., Sommerville, I., Dependability Properties of P2P Architectures, Proceedings of the 2nd IEEE Int'l Conf. Peer-to-Peer Computing (P2P 02), IEEE CS Press, 2002, pp. 173 174.

## Figures



**Figure 1** - Peer model of the P2P Application Framework



**Figure 2** – Using resources with different types of plug-in

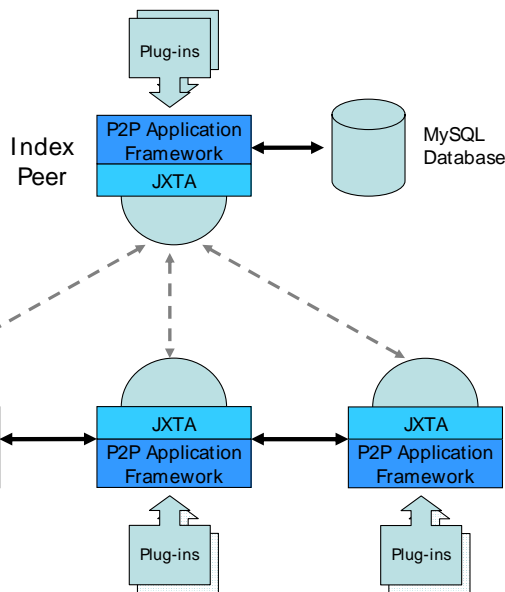


Figure 3 – Structure of our implementation of the framework

### Selected Interface Methods

**void activatePlugin(Vector resources)**

- activates a plug-in and passes a set of resources for it to act upon

**String getPluginName()**

- returns the name of the plug-in

**int getPluginID()**

- returns the ID of the plug-in

**void messageArrived(HashMap message)**

- informs the plug-in that it has received a message. The message is passed as a parameter

**Vector getAvailableResources()**

- requests details of all the resources the plug-in is currently making available

**void changeOfPeerStatus(String id, String name, String status)**

- informs the plug-in that a peer/user has changed its on-line status

**int[] getResourcesInterested()**

- request details of the Resource Types the plug-in is interested in

### Selected API Methods

**void resourceSearch(int pluginid, int resourcetype, String nickname, String keywords)**

- requesting the framework to perform a resource search (this can also include users)

**long sendMessage(String targetid, HashMap messageObject)**

- requesting the framework to send a message to a target

**void resourcesUpdate(Vector resources, int changeType)**

- inform the framework that the status of the specified resources has changed

**Vector getFavouritesList()**

- request the user/peers favourites list

**Vector getUsableResourceTypes()**

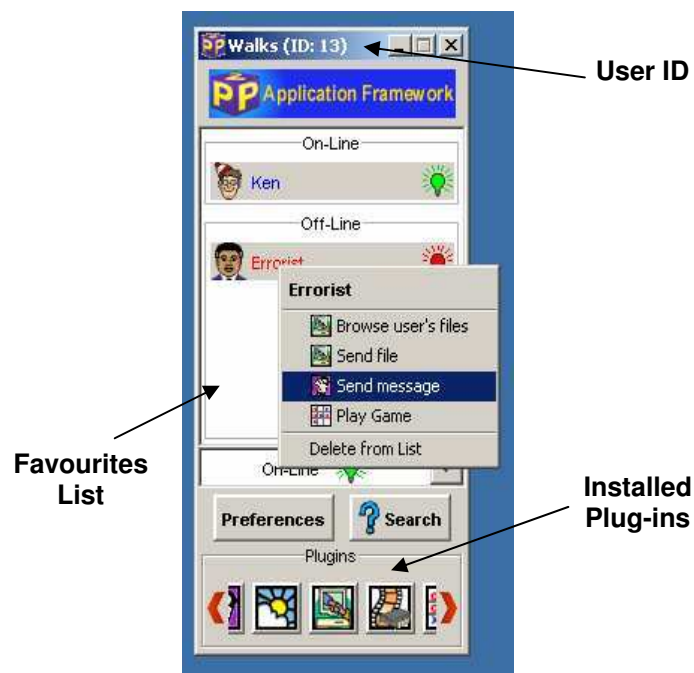
- request a list of the currently registered Resource Types within the framework

**Vector getMonitoringInfo(String targeted, int from, int to)**

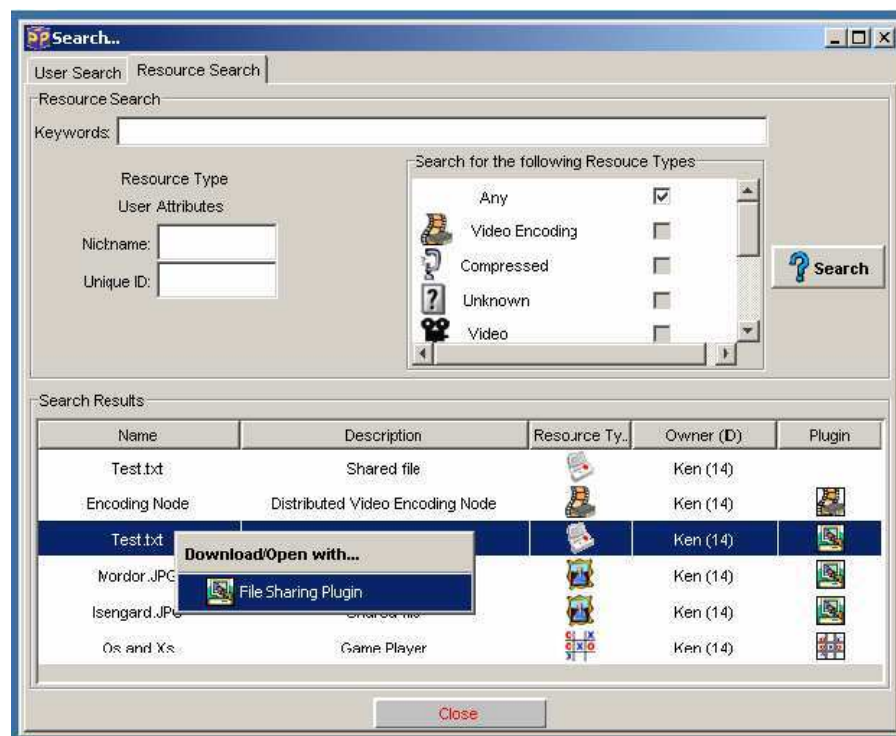
- request monitoring information about a given user/peer

Figure 4 – Selection of interface methods that plug-ins must implement, and

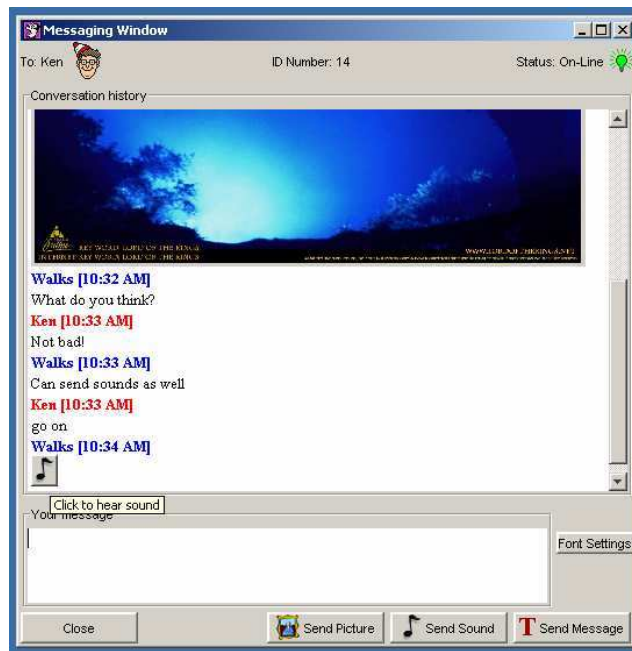
API methods that are implemented within the AD layer



**Figure 5** - Front-end of the framework implementation



**Error!Figure 6** – Searching with the framework



**Figure 7** - The Instant Messenger Plug-in

```
// Obtain Plug-in Class
Class IMpluginClass = pluginInstance.getClass();

// Specify class of parameters
Class param[] = {java.util.HashMap.class};

// Obtain MessageArrived method from class
Method m = IMpluginClass.getMethod("messageArrived", param);

// Invoke method within plug-in, passing the message as parameter
m.invoke(pluginInstance, (Object) HashMapMessage);
```

**Figure 8** - The AD Layer routing a message to the plug-in

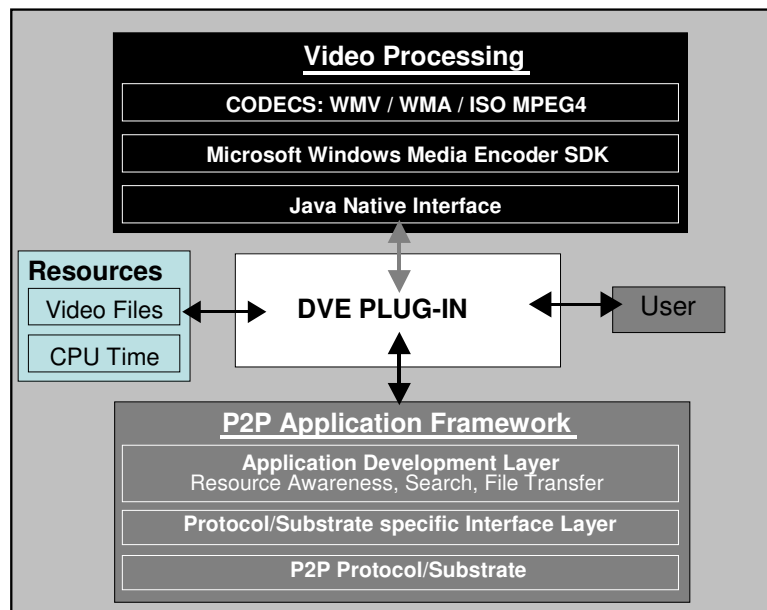


Figure 9 – The DVE Architecture [26]

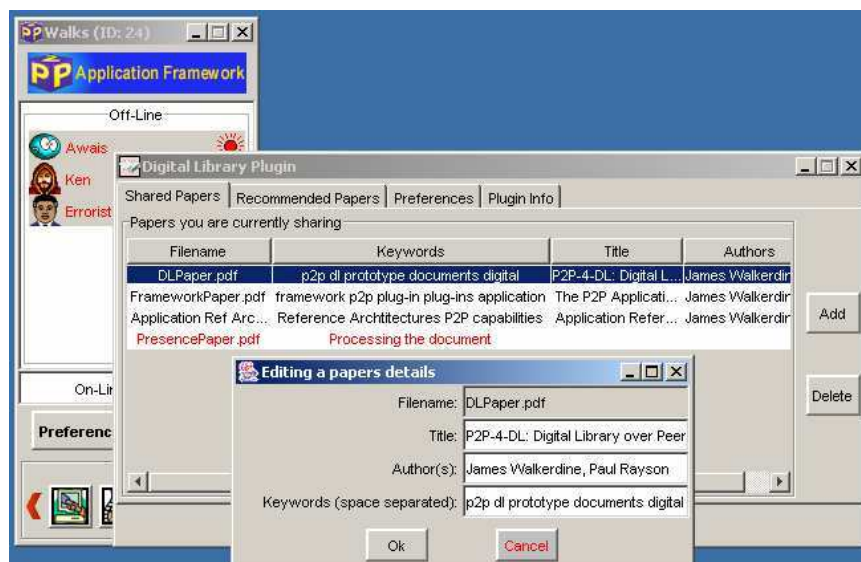
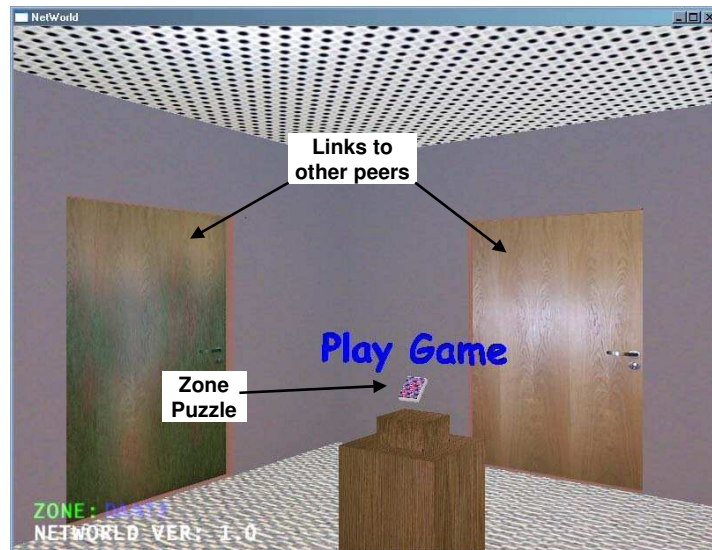


Figure 10 – The Digital Library Plug-in



**Figure 11** - Exploring a zone within Net World

```
// create, and Start the default jxta NetPeerGroup
netPeerGroup = PeerGroupFactory.newNetPeerGroup();

// get the pipe service
pipeSvc = netPeerGroup.getPipeService();

// create output pipe
OutputPipe CurrentOutPipe = null;

for (int i=0; i<Attempts; i++)
{
    try
    {
        CurrentOutPipe = pipeSvc.createOutputPipe(target, timeOut);
        break;
    }

    catch (java.io.IOException e)
    {;}
}

if (CurrentOutPipe == null)
{
    return false;
}

// send message
CurrentOutPipe.send(message);
```

(a)

```
// Connect to network
framework.connectToNetwork();

// send message
framework.sendMessage(target, message);
```




















(b)

**Figure 12** - Code comparison between JXTA based (a) and P2P Application Framework (b) based

code



## Tables

Icon	Plug-in	Description	Icon	Resource Type	Description
	<i>Instant Messenger</i>	A typical instant messenger application		<i>Video Encoding</i>	Donating CPU resources for video encoding
	<i>File Sharing</i>	Napster styled file sharing application		<i>Audio</i>	Audio Files
	<i>Network Mood</i>	Simple visualisation of overall user 'mood' within the network		<i>Compressed</i>	Compressed Files
	<i>O's and X's Game</i>	Simple P2P based Noughts and Crosses game (or Tic-Tac-Toe)		<i>Document</i>	Document Files
	<i>Distributed Video Encoder</i>	P2P based video encoding application		<i>Program</i>	Program Files
	<i>Digital Library</i>	P2P based Digital Library with natural language processing facilities		<i>Picture</i>	Image Files
	<i>Audio Streamer</i>	P2P based audio streaming application		<i>Video</i>	Video Files
	<i>Net World</i>	Distributed virtual game world		<i>O's and X's</i>	Availability for a Noughts and Crosses game
				<i>Audio Stream</i>	Audio Stream
				<i>A Net World</i>	Contributing a zone to the Net World game
				<i>Unknown</i>	Unrecognised Resource

**Table 1** – Developed Plug-ins and Resource Types used within the framework

Application	Total lines of code for JXTA version	Total lines of code for P2P Application Framework version	Total Code Percentage Reduction	Lines of P2P code for JXTA version	Lines of P2P code for P2P Application Framework version	P2P Code Percentage Reduction
<i>Distributed Video Encoder</i>	1191	649	45.5%	801	259	67.6%
<i>P2P Digital Library</i>	2195	1627	25.9%	857	289	66.3%
<i>NetWorld</i>	4947	4379	11.5%	1803	1235	31.5%
<i>Instant Messenger</i>	1557	1222	21.5%	780	445	42.9%
<i>File Sharing</i>	1767	1129	36.1%	885	317	64.1%

**Table 2** - Code comparisons for a selection of the P2P applications that have been developed