Project JXTA: A Technology Overview

Li Gong
Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300

Introduction

JXTA technology is a network programming and computing platform that is designed to solve a number of problems in modern distributed computing, especially in the area broadly referred to as peer-to-peer computing, or peer-to-peer networking, or simply P2P.

Why Peer-To-Peer

P2P is the latest buzzword sweeping through the computing industry. Within the past few months, P2P has appeared on the covers of Red Herring and Wired, and was crowned by Fortune as one of the four technologies that will shape the Internet's future. There is a lot of hype, no doubt — but there is also a lot of substance in P2P.

The Internet has three valuable fundamental assets — information, bandwidth, and computing resources — all of which are vastly under utilized, partly due to the traditional client-server computing model.

- First, no single search engine or portal can locate and catalog the ever-increasing amount of information on the Web in a timely way. Moreover, a huge amount of information is transient and not subject to capture by techniques such as Web crawling. For example, according to research[1], the world produces two exabytes or about $2x10^{18}$ bytes of information every year, but only publishes about 300 terabytes or about $3x10^{12}$ bytes. In other words, for every megabyte of information produced, only one byte gets published. Moreover, Google claims that it searches about only $1.3x10^{8}$ web pages. Thus, finding useful information in real time is increasingly difficult.

- Second, although miles of new fiber have been installed, the new bandwidth gets little use if everyone goes to Yahoo for content and to eBay for auctions. Instead, hot spots just get hotter while cold pipes remain cold. This is partly why most people still feel the congestion over the Internet while a single fiber's bandwidth has increased by a factor of 10^{6} since 1975, doubling every 16 months.

- Finally, new processors and storage devices continue to break records in speed and capacity, supporting more powerful end devices throughout the network. However, computation continues to accumulate around data centers, which have to increase their workloads at a crippling pace, thus putting immense pressure on space and power consumption.

This paper does not attempt to define exactly what P2P is. Instead, the term peer-to-peer networking is applied to a wide range of technologies that greatly increase the utilization of information, bandwidth, and computing resources in the Internet. Frequently, these P2P technologies adopt a network-based computing style that neither excludes nor inherently depends on centralized control points. Apart from improving the performance of information discovery, content delivery, and information processing, such a style also can enhance the overall reliability and fault-tolerance of computing systems.

Why Project JXTA

Project JXTA has a set of objectives that are derived from what we perceive as shortcoming of many peer-to-peer systems in existence or under development.

1. For brevity, citations to data sources are omitted from this paper.

- *Interoperability.* JXTA technology is designed to enable interconnected peers to easily locate each other, communicate with each other, participate in community-based activities, and offer services to each other seamlessly across different P2P systems and different communities.

  Many peer-to-peer systems are built for delivering a single type of services. For example, Napster provides music file sharing, Gnutella provides generic file sharing, and AIM provides instant messaging. Given the diverse characteristics of these services and the lack of a common underlying P2P infrastructure, each P2P software vendor tends to create incompatible systems — none of them able to interoperate with one another. This means each vendor creates its own P2P user community, duplicating efforts in creating software and system primitives commonly used by all P2P systems. Moreover, for a peer to participate in multiple communities organized by different P2P implementations, the peer must support multiple implementations, each for a distinct P2P system or community, and serve as the aggregation point.

  This situation resembles the pre-browser Internet, where to have Internet access often meant a subscription with AOL, Prodigy, or CompuServ. The result was that a user was locked into one community, and service providers had to offer their services or content in ways that were specific to how each community operated. Project JXTA aims to bring to the P2P world what the browser brought to the Internet.

- *Platform independence*. JXTA technology is designed to be independent of programming languages (such as C or the Java™ programming language), system platforms (such as the Microsoft Windows and UNIX® operating systems), and networking platforms (such as TCP/IP or Bluetooth).

  Many P2P systems today offer their features or services through a set of APIs that are delivered on a particular operating system using a specific networking protocol. For example, one system might offer a set of C++ APIs, with the system initially running only on Windows, over TCP/IP, while another system offers a combination and C and Java APIs, running on a variety of UNIX systems, over TCP/IP but also requiring HTTP. A P2P developer is then forced to choose which set of APIs to program to, and consequently, which set of P2P customers to target. Because there is little hope that the two systems will interoperate, if the developer wants to offer the same service to both communities, they have to develop the same service twice for two P2P platforms or develop a bridge system between them. Both approaches are inefficient and impractical considering the dozens of P2P platforms in existence.

  JXTA technology is designed to be embraced by all developers, independent of preferred programming languages, development environments, or deployment platforms.

- *Ubiquity.* JXTA technology is designed to be implementable on every device with a digital heartbeat, including sensors, consumer electronics, PDAs, appliances, network routers, desktop computers, data-center servers, and storage systems.

  Many P2P systems, especially those being offered by upstart companies, tend to choose (perhaps unsurprisingly) Microsoft Windows as their target deployment platform. The cited reason for this choice is to target the largest installed base and the fastest path to profit. The inevitable result is that many dependencies on Wintel-specific features are designed into (or just creep in) the system. This is often not the consequence of technical desire but of engineering reality with its tight schedules and limited resources.

  This approach is clearly short-sighted, as P2P does not stand for PC-To-PC. Even though the earliest demonstration of P2P capabilities are on Wintel machines — the middle of the computing hardware spectrum, it is very likely that the greatest proliferation of P2P technology will occur at the two ends of the spectrum — large systems in the enterprise and consumer-oriented small systems. In fact, betting on any particular segment of the hardware or software system is not future proof.

  Project JXTA envisions a world where each peer, independent of software and hardware platform, can benefit and profit from being connected to millions of other peers.

Project JXTA Technology

Project JXTA was originally conceived by Sun Microsystems, Inc. and designed with the participation of a small but growing number of experts from academic institutions and industry. JXTA technology is soon to be open-sourced and will be co-developed by many more contributors. As such, it is still evolving. This paper serves to highlight the set of technical concepts and attributes that are currently most significant. The up-to-date JXTA Technology Specification is available on the Web at *http://www.jxta.org*.

At the start of Project JXTA, we analyzed many P2P software architectures and found a common layering structure at the conceptual level depicted in the figure below.

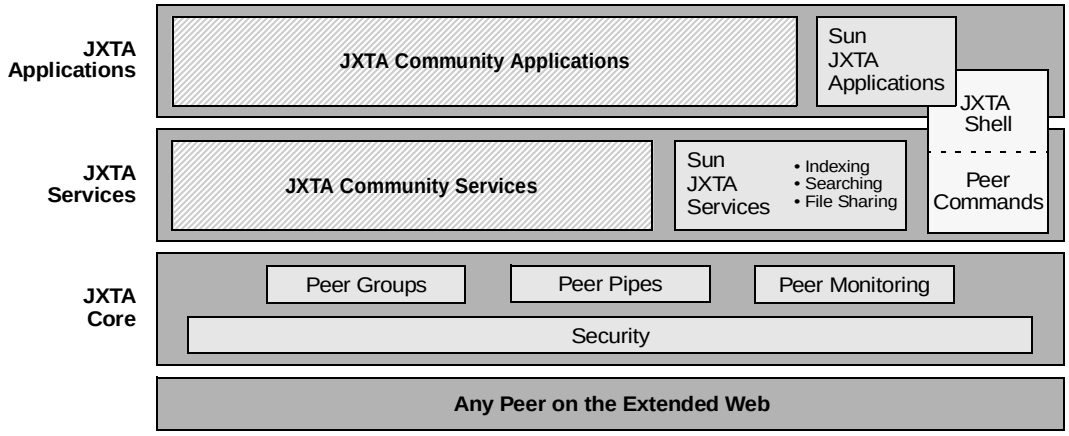

Figure 1: P2P Software Architecture

We can roughly break down a typical P2P software stack into three layers. At the bottom is the core layer that deals with peer establishment, communication management such as routing, and other low-level "plumbing". In the middle is a service layer that deals with higher-level concepts, such as indexing, searching, and file sharing. These services, which make heavy use of the plumbing features provided by the core, are useful by themselves but also are commonly included as components in an overall P2P system. At the top is the layer of applications, such as emailing, auctioning, and storage systems. Some features, such as security, manifest in all three layers and throughout a P2P system, albeit in different forms according to the location in the software architecture.

JXTA technology is designed to provide a layer on top of which services and applications are built. We designed this layer to be thin and small, yet providing interesting and powerful primitives for use by the services and applications. We envision this layer to stay thin and small as this is the best approach both to maintaining interoperability among competitive offerings from various P2P contributors and to providing maximum room for innovation (and profit) by these contributors.

JXTA Technology Concepts

JXTA Technology. At the highest abstraction level, JXTA technology is a set of protocols. Each protocol is defined by one or more messages exchanged among participants of the protocol. Each message has a pre-defined format, and may include various data fields.

In this regard, it is akin to TCP/IP. Whereas TCP/IP links Internet nodes together, JXTA technology connects peer nodes with each other. TCP/IP is platform-independent by virtue of being a set of protocols. So is JXTA. Moreover, JXTA technology is transport independent and can utilize TCP/IP as well as other transport standards.

The following protocols are currently defined:

- Peer Discovery Protocol
- Peer Resolver Protocol
- Peer Information Protocol
- Rendezvous Protocol
- Pipe Binding Protocol
- Endpoint Routing Protocol

The naming of these protocols may not be obvious to some readers. For example, the Peer Discovery Protocol is the protocol a peer uses to perform discovery. The name might have suggested that the protocol is used for discovering peers only, while in effect it can be used to discover peers, peer groups, and any other advertisements. The first word, peer, is the subject, and not necessarily the object, of discovery.

To underpin this set of protocols, JXTA technology defines a number of concepts including peer, peer group, advertisement, message, pipe, and more. We spend the rest of this section going over these.

JXTA technology concepts may appear very simplistic. This is deliberate. Obviously, we want to keep the specification simple and small. More importantly, there are many areas where there is not one correct way to do something or where what should be done depends on the nature and context of the overriding application. This phenomenon is especially acute in security, where every P2P application may choose a different authentication scheme, a different way to ensure communication security, a different encryption algorithm for data security, a different signature scheme for authenticity, and a different access control policy. For these areas, we tend to under-specify, focusing on mechanisms instead of policy, so that application developers can have the maximum freedom to innovate and offer competitive solutions. Because of the under-specification, it is important to distinguish between what is defined for JXTA technology and what the first implementation (Version 1.0) does. Often the implementation chooses to do something in a particular way, but this does not mean that JXTA is defined to behave in this fashion.

Identifiers. JXTA uses UUID, a 128-bit datum to refer to an entity (a peer, an advertisement, a service, etc.). It is easy to guarantee that each entity has a unique UUID within a local runtime environment, but because we do not assume any global state, there is no absolute way to provide a guarantee of uniqueness across an entire community that may consist of millions of peers. This is not much of a problem because a UUID is used as an internal identifier. It becomes significant only after it is securely bound to other information such as a name and a network address. We assume and expect that sophisticated naming and binding services will be developed for the JXTA platform.

Advertisements. An advertisement is an XML structured document that names, describes, and publishes the existence of a resource, such as a peer, a peer group, a pipe, or a service. JXTA technology defines a basic set of advertisements (see *JXTA Technology Specification* for details). More advertisement subtypes can be formed from these basic types using XML schemas.

Peers. A peer is any entity that can speak the protocols required of a peer. This is akin to the Internet, where an Internet node is any entity that can speak the suite of IP protocols. As such, a peer can manifest in the form of a processor, a process, a machine, or a user. Importantly, a peer does not need to understand all the six protocols given earlier. A peer can still perform at a reduced level if it does not support a protocol.

Messages. Messages are designed to be usable on top of asynchronous, unreliable, and uni-directional transport. Therefore, a message is designed as a datagram, containing an envelope and a stack of protocol headers with bodies. The envelope contains a header, a message digest, (optionally) the source endpoint, and the destination endpoint. An endpoint is a logical destination, given in the form of a URI, on any networking transport capable of sending and receiving datagram-style messages. Endpoints are typically mapped to physical addresses by a messaging layer. Such a message format is designed to support multiple transport standards.

Each protocol body contains a variable number of bytes, and one or more credentials used to identify the sender to the receiver. The exact format and content of the credentials are not specified. For example, a credential can be a signature that provides proof of message integrity and/or origin. As another example, a message body may be encrypted, with the credential providing further information on how to decrypt the content.

Peer Groups. A peer group is a virtual entity that speaks the set of peer group protocols. Typically, a peer group is a collection of cooperating peers providing a common set of services. The specification does not dictate when, where, or why to create a peer group, or the type of the group, or the membership of the group. It does not even define how to create a group. In fact, the relationship between a peer and a peer group can be somewhat meta-physical. JXTA does not care by what sequence of events a peer or a group comes into existence. Moreover, it does not limit how many groups a peer can belong to, or if nested groups can be formed. It does define how to discover peer groups using the Peer Discover Protocol.

There is a special group, called the *World Peer Group*, that includes all JXTA peers. This does not mean that peers inside this special group can always discover or communicate with each other — e.g., they may be separated by a network partition. Participation in the World Peer Group is by default.

Pipes. Pipes are communication channels for sending and receiving messages, and are asynchronous. They are also uni-directional, so there are input pipes and output pipes. Pipes are also virtual, in that a pipe's endpoint can be bound to one or more peer endpoints.

A pipe is usually dynamically bound to a peer at runtime via the Pipe Binding Protocol. This also implies that a pipe can be moved around and bound to different peers at different times. This is useful, for example, when a collection of peers together provide a high level of fault tolerance, where a crashed peer may be replaced by a new peer at a different location, with the latter taking over the existing pipe to keep the communication going.

A point-to-point pipe connects exactly two peer endpoints together. The pipe is an output pipe to the sender and input pipe to the receiver, with traffic going in one direction only — from the sender to the receiver. A propagate pipe con-

nects multiple peer endpoints together, from one output pipe to one or more input pipes. The result is that any message sent into the output pipe is sent to all input pipes.

JXTA does not define how the internals of a pipe works. Any number of unicast and multicast protocols and algorithms, and their combinations, can be used. In fact, one pipe can be chained together where each section of the chain uses an entirely different transport protocol.

We designed pipes to be asynchronous, uni-directional, and unreliable, because this is the foundation of all forms of transport and carries with it the lowest overhead. We expect that very soon enhanced pipes with additional properties such as reliability, security, quality of service, will be created by the developer community. Of course, when JXTA runs on top of transports that already have such properties, it is not hard for an implementation to optimize and utilize them. For example, when two peers communicate with each other and both have TCP/IP support, then an implementation can easily create bi-directional pipes.

JXTA Technology Protocols

Initially Project JXTA has defined the following six protocols. More protocols may be developed by the developer community.

Peer Discovery Protocol. This protocol enables a peer to find advertisements on other peers, and can be used to find any of the peer, peer group, or advertisements.

This protocol is the default discovery protocol for all peer groups, including the World Peer Group. It is conceivable that someone may want to develop a premium discovery mechanism that may or may not choose to leverage this default protocol, but the inclusion of this default protocol means that all JXTA peers can understand each other at the very basic level.

Peer discovery can be done with or without specifying a name for either the peer to be located or the group to which peers belong. When no name is specified, all advertisements are returned.

Peer Resolver Protocol. This protocol enables a peer to send and receive generic queries to find or search for peers, peer groups, pipes, and other information. Typically, this protocol is implemented only by those peers that have access to data repositories and offer advanced search capabilities.

Peer Information Protocol. This protocol allows a peer to learn about other peers' capabilities and status. For example, one can send a *ping* message to see if a peer is alive. One can also query a peer's properties where each property has a name and a value string.

Rendezvous Protocol. This protocol allows a peer to propagate a message within the scope of a peergroup.

Pipe Binding Protocol. This protocol allows a peer to bind a pipe advertisement to a pipe endpoint, thus indicating where messages actually go over the pipe. In some sense, a pipe can be viewed as an abstract, named message queue that supports a number of abstract operations such as create, open, close, delete, send, and receive. Bind occurs during the open operation, whereas unbind occurs during the close operation.

Endpoint Routing Protocol. This protocol allows a peer to ask a peer router for available routes for sending a message to a destination peer. Often, two communicating peers may not be directly connected to each other. Example of this might include two peers that are not using the same network transport protocol, or peers separated by firewalls or NAT. Peer routers respond to queries with available route information, which is a list of gateways along the route. Any peer can decide to become a peer router by implementing the Endpoint Routing Protocol.

Security Considerations

The security requirements of a P2P system are very similar to any other computer system. The three dominant requirements are confidentiality, integrity, and availability. These translate into specific functionality requirements that include authentication, access control, audit, encryption, secure communication, and non-repudiation.

Such requirements are usually satisfied with a suitable security model or architecture, which is commonly expressed in terms of subjects, objects, and actions that subjects can perform on objects. For example, the UNIX operating system has a simple security model. Users are subjects. Files are objects. Whether a subject can read, write, or execute an object depends on whether the subject has permission as expressed by the permissions mode specified for the object. However, at lower levels within the system, the security model is expressed with integers, in terms of uid, gid, and the permission mode. Here, the low-level system mechanisms do not (need to) understand the concept of a user and do not (need to) be involved in how a user is authenticated and what uid and gid they are assigned.

Given that JXTA is defined around the concepts of peers and peer groups, one can envision a security architecture in which peer IDs and group IDs are treated as low-level subjects (just like uid and gid), codats[1] are treated as objects (just like files), and actions are those operations on peers, peer groups, and codats. However, the reality is more complicated. For example, given that codats can have arbitrary forms and properties, it is unclear what sets of actions should be defined for them. It is quite likely that codats will carry along with them definitions of how they should be accessed. Such codats are analogous to objects, which define for themselves access methods others can invoke.

Developing a more concrete and precise security architecture is an ongoing project. As we gain more experience with developing services and applications on top of JXTA, we will understand better what particular architecture is the most suitable.

In considering a security architecture, it is important to note that security requirements for JXTA are further affected by some unique characteristics:

- JXTA technology is a platform focused on mechanisms and not policy. For example, UUIDs are used throughout, but they by themselves have no external meaning. Without additional naming and binding services, UUIDs are just numbers that do not correspond to anything like a user or a principal. Therefore, unlike some other computer systems, JXTA does not define a high-level security model such as information flow, Bell-LaPadula, or Chinese Wall.

  When UUIDs are bound to external names or entities to form security principals, authenticity of the binding can be ensured by placing in the data field security attributes, for example, digital signatures that testify to the trustworthiness of the binding. Once this binding is established, we will be able to perform authentication of the principal, access control based on the principal as well as the prevailing security policy, and other functions such as resource usage accounting.

- JXTA technology is neutral to cryptographic schemes or security algorithms. As such, it does not mandate any specific security solution. In such cases, at best, we have provided a framework where different security solutions can be plugged in. At minimum, we have provided enough hooks and place holders so that different security solutions can be implemented.

  For example, every message has a designated credential field that can be used to place security-related information. However, exactly how to interpret such information is beyond the scope of the specification, and is left to services and applications.

- JXTA technology can sometimes satisfy security requirements at different levels of the system. To allow maximum flexibility and avoid redundancy, JXTA technology typically does not force a particular implementation on developers. Instead, we envision that a number of enhanced platforms will emerge that provide the appropriate security solutions to their targeted deployment environment.

To illustrate the last point, let's examine two requirements, communications security and anonymity.

Communications security. We have defined that peers communicate through pipes. For the sake of discussion, suppose both confidentiality and integrity in the communications channel are desired. One solution is to use VPNs to move all network traffic. Another solution is to create a secure version of the pipe, similar to a protected tunnel, such that any message transmitted over this pipe is automatically secured. A third solution is to use regular communications mechanisms but let the developer protect specific data payloads with encryption techniques and digital signatures. JXTA technology can accommodate any of these solutions.

Anonymity. Anonymity does not mean the absence of identity. Indeed, sometimes a certain degree of identification is unavoidable. For example, a cellphone number or a SIM card identification number cannot be kept anonymous, because it is needed by the phone company to authorize and set up calls. As another example, the IP number of a PC cannot be hidden from its nearest gateway or router if the PC wants to send and receive network traffic.

Moreover, anonymity can be built on top of identity, but not verse versa. And often there are multiple ways to ensure anonymity. In the examples above, it is much more difficult to link a pre-paid SIM card sold over the retail counter for cash to the actual cellphone user. Likewise, a co-operative gateway or router can help hide the PC's true IP address from the outside world by using message relays or NAT.

1. I coined the term codat to means code and data.

Suppose a JXTA technology-based naming service can bind a peer to a human user. The user's anonymity can be ensured through the naming service, or the authentication service, or a proxy service, or any combination of these. JXTA is independent of the solution chosen by a particular application.

JXTA Technology Version 1.0

As of April 2001, the first prototype implementation is available on *http://www.jxta.org*. It is implemented on JDK™ release 1.1.4, which we decided is the most common Java platform available on machines running Microsoft Windows and the UNIX operating system. The code should run on Windows95, 98, 2000, ME, and NT out of the box. It also runs on the Solaris™ Operating Environment and Linux with the appropriate level of Java runtime environment support.

Version 1.0 is a starting point for the developer community than a finished product. We expect to see lots of tuning in the coming months. Without any effort to optimize the code size, the core classes packed into a jar file have about 250 KB. We expect that much smaller implementations will soon emerge.

This section discusses some of the key issues encountered during this phase of the project.

Discovery Mechanisms

JXTA does not mandate exactly how discovery is done. It can be completely decentralized, completely centralized, or a hybrid of the two. In JXTA Version 1.0, we support the following discovery mechanisms:

- LAN-based discovery. This is done via a local broadcast over the subset.
- Discovery through invitation. If a peer receives an invitation (either in-band or out-of-band), the peer information contained in the invitation can be used to discover a (perhaps remote) peer.
- Cascaded discovery. If a peer discovers a second peer, the first peer can, with the permission of the second peer, view the horizon of the second peer, discovering new peers, groups, and services.
- Discovery via rendezvous points. A rendezvous point is a special peer that keeps information about the peers it knows about. A peer that can communicate via a rendezvous peer, perhaps via a pipe, can learn of the existence of other peers.

Rendezvous points are especially helpful to an isolated peer by quickly seeding it with lots of information. It is conceivable that some web sites or its equivalent will be devoted to providing information of well-known rendezvous points.

Propagation Scopes

JXTA does not mandate how messages are propagated. For example, when a peer sends out a peer discovery message, the Peer Discover Protocol does not dictate if the message should be confined to the local area network only, or if it must be propagated to every corner of the world.

The current implementation of JXTA uses the concept of a peer group as an implicit scope of all messages originated from within the group. In theory, any scope can be realized with the formation of a corresponding peer group. For example, a peer in San Francisco looking to buy a used car is normally not interested in cars available outside of the Bay Area. In this case, the peer would like to multicast a message to a subset of the current worldwide peer group, and a subgroup can be formed specially for this purpose. On the other hand, it seems more convenient and efficient if such a multicast can be done without the formation of a new peer group.

We can envision a number of approaches to solving this problem. For example, all messages can carry a special scope field, which indicates the scope for which a message is intended. Any peer who receives this message can propagate the message based on the scope indicator. Using this approach, it is desirable that a sending peer is bootstrapped with some well-defined scopes and also can discover additional scopes. Further work is needed in this area.

XML

In theory, JXTA can be independent of any format used to encode advertisement documents and messages. In practice, it uses XML as the encoding format, mainly for its convenience in parsing and for its extensibility. Three points worth noting about the use of XML.

- If the world decides to abandon XML tomorrow and uses YML instead, JXTA can be simply re-defined and re-coded to use the YML format.
- The use of XML does not imply that all peer nodes must be able to parse and to create XML documents. For

example, a cellphone with limited resources can be programmed to recognize and to create certain canned XML messages and can still participate in a network of peers.
- To keep Version 1.0 small, we do not use a XML parser that can parse any XML document. Instead, we use a lightweight parser that supports a subset of XML. We are working towards normalizing this subset according to an existing effort called MicroXML.

Security

As we mentioned earlier, at many places JXTA is independent of specific security approaches. Nonetheless, Version 1.0 is a first step towards providing a comprehensive set of security primitives to support the security solutions used by JXTA services and applications. To be more specific, JXTA Version 1.0 attempts to provide the following security primitives:

- A simple crypto library supporting hash functions (e.g., MD5), symmetric encryption algorithms (e.g., RC4), and asymmetric crypto algorithms (e.g., Diffie-Hellman and RSA).
- An authentication framework that is modeled after PAM (Pluggable Authentication Module, first defined for the UNIX platform and later adopted by the Java security architecture).
- A simple password-based login scheme that, like other authentication modules, can be plugged into the PAM framework.
- A simple access control mechanism based on peer groups, where a member of a group is automatically granted access to all data offered by another member for sharing, whereas non-members cannot access such data.
- A transport security mechanism that is modeled after SSL/TLS, with the exception that it is impossible to perform a handshake, a crypto strength negotiation, or a two-way authentication on a single pipe, as a pipe is unidirectional.
- The demonstration services called InstantP2P and CMS (content management service) also make use of additional security features provided by the underlying Java platform.

NAT and Firewall

The wide-spread use of NAT and firewalls severely affects the smooth operation of many P2P systems. It also affects the usability of JXTA. In particular, a peer outside a firewall or a NAT gateway cannot discover peers inside the firewall or the NAT gateway. In the absence of getting system administrators to let JXTA traffic through (say by opening a special incoming port at the firewall or gateway), there are two rather obvious ideas to deal with this problem.

- Ask peers inside firewalls to initiate connections to peers outside the firewall.
- Set up peer nodes that operate like mailbox offices where traffic to a peer inside the firewall is queued up to be picked up at a designated relay peer outside the firewall. The peer inside the firewall can initially reach outside the firewall, select a relay peer, and widely advertise this fact. Later, it can periodically contact the relay peer to retrieve messages.

These are far from ideal solutions and this is an active research area with lots of ongoing work.

Peer Monitoring and Metering

Peer monitoring means the capability to closely keep track of a (local or remote) peer's status, control the behavior of a peer, and to respond to actions on the part of a peer. This capability is very useful when a peer network wants to offer premium services with a number of desirable properties such as reliability, scalability, and guaranteed response time. For example, a failure in the peer system must be detected as soon as possible so that corrective actions can be taken. It is sometimes better to shut down an erratic peer and transfer its responsibilities to another peer.

Peer metering means the capability to accurately account for a peer's activities, in particular its usage of valuable resources. Such a capability is essential if the network economy is to go beyond flat-rate services. Even for providers offering flat rate services, it is to their advantage to be able to collect data and analyze usage patterns in order to be convinced that a flat rate structure is sustainable and profitable.

JXTA currently approaches monitoring and metering through the Peer Information Protocol, where a peer can query another peer for data such as up time and amount of data handled.

Obviously, security is central to peer monitoring and metering. A peer may choose to authenticate any command it receives. It may also decide to not answer queries from suspect sources.

A new project on monitoring and metering is set up on the Project JXTA website and we expect to see lots of activities in this area in the very near future.

JXTA Technology-Based Services and Applications

It is beyond the scope of this paper to explore in any depth what services and applications the JXTA platform may inspire, or technically how these services and applications might be built. This section is an illustration of what JXTA-based services and applications can be, using a few examples the Project JXTA team has built for demonstration purposes.

Services

A service denotes a set of functions that a provider offers. A peer can offer a service by itself or in cooperation with other peers. A service provider peer publicizes the service by publishing a service advertisement. Other peers can then discover this service and make use of it. Each service has a unique ID and name that consists of a canonical name string and a series of descriptive keywords that uniquely identifies the service.

Sometimes, a service is well-defined and widely available such that a peer can just use it. Other times, special code may be needed in order to actually access a service. For example, the way to interface with the service provider may be encoded in a piece of software. In this case, it is most convenient if a peer can locate an implementation that is suitable for the peer's specific runtime environment. Of course, if multiple implementations of the same service are available, then peers hosted on systems with Java runtime environments can use Java programming language implementations while native peers to use native code implementations. Service implementations can be pre-installed into a peer node or loaded from the network. The process of finding, downloading, and installing a service from the network is similar to performing a search on the Internet for a web page, retrieving the content of the page, and then installing the required plug-in to work with the page. Once a service is installed and activated, pipes may be used to communicate with the service.

We refer to a peer service that executes only on a single peer as a peer service. We call a service that is composed of a collection of cooperating instances of the service running on multiple peers a peer group service. A peer group service can employ fault tolerance algorithms to provide the service at a higher level of availability than that a peer service can offer.

Although the concept of a service is orthogonal to that of a peer and a peer group, a group formed using the JXTA platform may require a minimum set of services needed to support the operation of the group.

The first JXTA implementation has built in a set of default peer group services such as peer discovery, as well as a set of configurable services such as routing. This does not mean that any peer must have these services. For example, we can imagine a cellphone peer that is pre-configured with enough information to contact a fixed server provided by the telecom operator. This may be enough to bootstrap the cellphone peer without requiring it to independently carry with it additional services.

The JXTA Shell: An Example Application

The JXTA Shell is an important application built on top of the JXTA platform, both as a powerful demonstration of JXTA and as a useful development environment. The JXTA Shell provides interactive access to the JXTA platform via a simple command line interface, just like in UNIX where interactive access to the UNIX platform is through a shell command line interpreter that enables users to manipulate and manage UNIX files and processes. With the UNIX shell, one can learn a lot about how UNIX internal works and also accomplish a lot by writing shell scripts. The same is true for the JXTA Shell. However, while most of the UNIX shell commands are designed to execute on the local machine, the JXTA Shell is designed to be executed in a networked environment. What happens under the cover is that a user command is likely to generate a sequence of message exchanges between a set of peers, with some computation occurring on remote peer nodes, and with the answer being returned to the user.

Using the JXTA Shell, one can play with the JXTA platform core building blocks such as peers, peer groups, pipes, and codats. Codats are units of contents that can hold both code and data. A user can publish, search, and execute codats, discover peers or peer groups, create pipes to connect two peers, and send and receive messages.

The interpreter in the JXTA Shell operates in a simple loop: it accepts a command, interprets the command, executes the command, and then waits for another command. The shell displays a "*JXTA>*" prompt, to notify users that it is ready to accept a new command. Just as there are multiple widely used versions of the UNIX shell (e.g., the original Bourne Shell, the C Shell, BASH, the Korn Shell, TCSH), it will not be surprising if the same situation develops for the JXTA Shell.

It is beyond the scope of this paper to either describe how the JXTA Shell is built or give the complete list of supported commands, except perhaps for a few notes.

First, to the extent that makes sense, we have deliberately chosen to name JXTA Shell commands after the UNIX shell commands, such as “ls”, “cat”, in the hope that this makes the JXTA Shell more user friendly to UNIX shell users.

Secondly, in our Java technology-based implementation, most shell commands are not built in *per se*. Rather, they are just Java language programs and are dynamically loaded and started by the shell framework when the corresponding commands are typed in. Therefore, adding a new shell command is as easy as writing a program in the Java programming language.

The “pipe” operator (“|”) for chaining commands, together with the notion of *stdin*, *stdout*, and *stderr*, are fundamental to UNIX shell programming. The JXTA Shell provides a similar “pipe” capability to redirect a command output pipe into another command input pipe. Every JXTA Shell command is given a standard input, output and error pipes that a user can connect, disconnect and reconnect to other Shell commands.

Dynamic pipe operations. In the UNIX operating system, the C Shell command '*cat myfile | grep “jxta”*' has to complete or be killed with a Ctrl-C. The user cannot modify the pipe re-direction when the command is in flight. In JXTA, because pipes are more permanent than the entirely transient ones in UNIX systems, a user can dynamically disconnect and reconnect pipes.

Summary and Future Directions

We have developed a network programming platform specifically designed to be the foundation for peer-to-peer systems. Its objectives are

- Interoperability — any P2P system built with JXTA can talk to each other,
- Platform independence — JXTA can be implemented with any programming language and run on any software and hardware platform, and
- Ubiquity — JXTA can be deployed on any device with a digital heartbeat.

JXTA is defined as a set protocols, which use XML-encoded messages. As such, it stays away from APIs and remains independent of programming languages, so that it can be implemented in C/C++, Java, Perl, or other languages. This means heterogeneous devices with completely different software stacks can interoperate through JXTA protocols.

JXTA technology is designed to be also independent of transport protocols. It can be implemented on top of TCP/IP, HTTP, Bluetooth, HomePNA, and many other protocols. This means that a system built on top of JXTA functions in the same fashion when the system is expanded to a new networking environment or to a new class of devices, as long as there is a correct transport protocol handler for the new networking protocol.

The benefits of Project JXTA can be illustrated with a few application or usage scenarios. For example, assume there is a P2P community offering a search capability for its members, where one member can post a query and other members can hear and respond to the query. We can imagine that one member happens to be a Napster user and cleverly implemented a feature so that whenever a query contains a question for a MP3 file, this member will look up the Napster directory and then respond to the query with information returned by the Napster system. Here, a member without any knowledge of Napster can benefit because another member implemented a bridge to connect their P2P system to Napster. This type of bridging is very useful, but when the number of services is large, pair-wise bridging becomes more difficult and undesirable. JXTA aims to be the platform bridge that connect various P2P systems together.

In another example, suppose one engineering group requires a sizable storage capability, but with redundancy to protect data from sudden loss. A common solution is to purchase a storage system with a large capacity and mirrored disks. Another engineering group down the hall later decides to purchase the same system. Both groups end up with a lot of extra capacity, and have to pay higher prices for the mirroring feature. With JXTA technology, the groups could create a system that would require that they only buy a simple storage system without mirroring. In this system, disks could discover each other automatically and form a storage peer group, mirroring data using their spare capacity.

As a third example, many devices (such as cell phones, pagers, wireless email devices, PDAs, and PCs) carry directory and calendar information. Currently, synchronization among them is tedious and difficult. Often, the PC becomes the central synchronization point, where every other device has to connect to the PC using a unique device driver for each device. With JXTA technology, all of these devices could be made to interact with each other, without extra networking interfaces except those needed by the device themselves. JXTA would be the common layer of communication and data exchange.

The open sourcing of JXTA technology through *http://www.jxta.org* marks a significant turning point for Project JXTA. We expect that many more people will find it interesting and rewarding to contribute to the development of

JXTA and related technologies. More specifically, we hope to see immediate activities covering, but are not limited to, the following areas (not in any particular order):

- A native C/C++ implementation for systems without Java runtime environment support;
- A KVM-based implementation so that all KVM capable devices such as PDAs and cellphones can become JXTA peers;
- A test-bed scaling JXTA to thousands and millions of nodes, to discover design flaws;
- Testing and modeling technologies developed for P2P systems in general and for JXTA in particular;
- Naming and binding services;
- Mechanisms for supporting propagation scopes;
- Security services, including authentication, access control, and secure pipes;
- Solutions to overcome limitations of firewalls and NAT gateways;
- Rich definitions for peer monitoring and metering;
- An enhanced JXTA Shell, with new commands and new implementations.

Apart from these listed topics, there are many issues that need substantial research and development work. We look forward to a productive year ahead with JXTA technology.

SUN MICROSYSTEMS, INC., 4150 NETWORK CIRCLE, SANTA CLARA, CA 95054 USA
PHONE: 650-960-1300 or 650-960-1300 INTERNET: *www.sun.com*

October 29, 2002