JXTA: A Network Programming Environment

Li Gong • *Sun Microsystems Inc.*

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.

P2P is the latest buzzword sweeping through the computing industry. It brings a lot of hype with it, no doubt—but there is also a lot of substance in P2P. In this report, the term refers to a wide range of technologies that greatly increase the utilization of three valuable and fundamental Internet assets: information, bandwidth, and computing resources. All of these are vastly underutilized at this time, 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, research has shown that two exabytes or about 2×10^{18} bytes of information are produced every year, but only about 300 terabytes or about 3×10^{12} bytes are published. Moreover, Google claims that it searches about only 1.3×10^{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.

P2P technologies can 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, this style can also enhance the overall reliability and fault tolerance of computing systems.

Project JXTA Objectives

Project JXTA was originally conceived by Sun Microsystems and designed with the participation of a small number of experts from academic institutions and industry. The project defined a set of objectives based on what we perceived as shortcomings in many peer-to-peer systems, existing or under development.

Interoperability

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. 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 prebrowser Internet, where to have Internet access often meant a subscription with AOL, Prodigy, or CompuServe. 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 HTTP and the browser brought to the Internet.

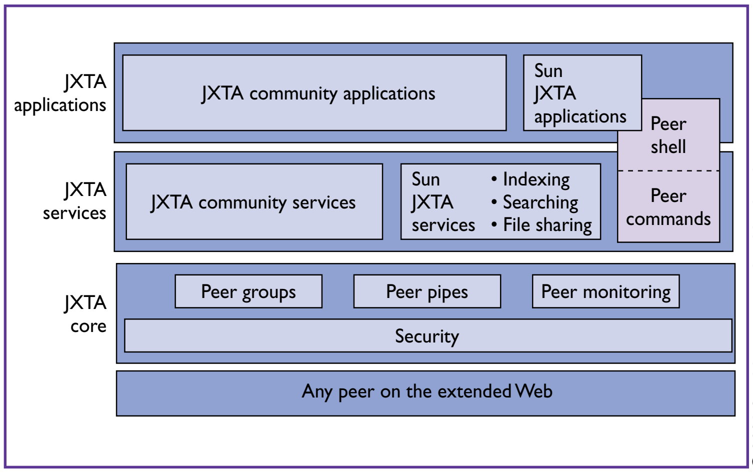

Figure 1. P2P software architecture. JXTA technology provides a layer on top of which services and applications are built.

Platform Independence

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 start-up companies, tend to choose Microsoft Windows as their target deployment platform. The reason cited 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 into) the system—often the consequence not of technical desire but just engineering realities of tight schedules and limited resources.

This approach is clearly shortsighted, 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 Technology

JXTA technology has just been open sourced and, as such, is still evolving. This report highlights the 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 Figure 1. A typical P2P software stack breaks down roughly into three layers. At the bottom, the core layer deals with peer establishment, communication management such as routing, and other low-level "plumbing." In the middle, a service layer handles 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 e-mail, auctions, 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, while still offering 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

At the highest abstraction level, JXTA technology is a set of protocols. Each protocol is defined by one or more messages exchanged among participants in the protocol; each message has a predefined format, and may include various data fields. In this regard, it is akin to TCP/IP. TCP/IP links Internet nodes together, while 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 six protocols are currently defined (for brief descriptions of each, see the sidebar, "JXTA Technology Protocols"):

- Peer Discovery Protocol
- Peer Eesolver Protocol
- Peer Information Protocol
- Peer Membership 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 used by a peer to perform discovery. The name might suggest that the protocol is used for discovering only peers, 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—identifiers, advertisements, peers, and others. JXTA technology concepts may appear very simplistic. This is deliberate. Obviously, we want to keep the specification simple and small. More importantly, in many areas there is no one correct way to do something, and 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 underspecify, focusing on mechanisms instead of policy, so that application developers can have the maximum freedom to innovate and offer competitive solutions. Because of the underspecification, 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, and so on). 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 guarantee uniqueness across an entire community that may consist of millions of peers. Because the UUID is an internal identifier, it is significant only after it is securely bound to other information such as a name and a network address. We 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, process, machine, or user.

Importantly, a peer does not need to understand all six defined JXTA protocols. It 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 unidirectional 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 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 JXTA specification does not dictate when, where, or why to create a peer group, or the type of 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 metaphysical. 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 Discovery Protocol.

There is a special group, called the World Peer Group, which includes all JXTA peers. This does not mean that peers inside this special group can always discover or communicate with each other; for example, 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 they are asynchronous. They are also unidirectional, 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

JXTA Technology Protocols

Project JXTA has defined six protocols so far. The developer community might define more over time.

Peer Discovery 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 might want to develop a premium discovery mechanism that might or might 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—enables a peer to send and receive generic queries to 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—allows a peer to learn about the capabilities and status of other peers. For example, a ping message can be sent to see if a peer is alive. A query can also be sent regarding a peer's properties where each property has a name and a value string.

Peer Membership Protocol—allows a peer to obtain group membership requirements, to apply for membership and receive a membership credential along with a full group advertisement, to update an existing membership or application credential, and to cancel a membership or an application credential. Authenticators and security credentials are used to provide the desired level of protection.

Pipe Binding 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—allows a peer to ask a peer router for available routes for sending a message to a destination peer. For example, when two communicating peers are not directly connected to each other, such as when they are not using the same network transport protocol or when they are separated by firewalls or NATs, peer routers respond to queries with available route information—that is, a list of gateways along the route. Any peer can decide to become a peer router by implementing the Peer Endpoint Protocol.

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* connects multiple peer endpoints together, from one output pipe to one or more input pipes. Accordingly, 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 with each section of the chain using an entirely different transport protocol. We designed pipes to be asynchronous, unidirectional, and unreliable, because this is the foundation of all forms of transport and carries with it the lowest overhead.

We expect the developer community to generate enhanced pipes with additional properties such as reliability, security, and quality of service. 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 bidirectional pipes.

Security Considerations

The security requirements of a P2P system are very similar to those of any other computer system. Requirements for confidentiality, integrity, and availability are dominant. They translate into requirements for specific functionalities that include authentication, access control, audit, encryption, secure communication, and nonrepudiation.

Such requirements are usually satisfied with a suitable security model or architecture, 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; a subject can read, write, or execute an object according to its permission as expressed by the permissions mode specified for the object. At lower levels within the system, however, the security model is expressed with integers, in terms of uid, gid, and the permission mode. 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, a security architecture could be envisioned in which peer IDs and group IDs are treated as low-level subjects (just like uid and gid), codats (meaning code and data) are treated as objects (just like files), and actions are specified 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 that 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 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 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. When UUIDs are bound to external names or entities to form security principals, authenticity of the binding can be ensured by placing security attributes in the data field, for example, digital signatures that testify to the trustworthiness of the binding. Once this binding is established, we will be able to authenticate the principal, control access based on the principal as well as the prevailing security policy, and perform other functions such as resource usage accounting.
- JXTA technology is neutral to cryptographic schemes or security algorithms.
- JXTA technology can sometimes satisfy security requirements at different levels of the system. For example, communications security can be provided through VPNs, secured pipes, or secured data over unsecure pipes. To allow maximum flexibility and avoid redundancy, JXTA technology typically does not force a particular imple-

mentation on developers. Instead, we envision that a number of enhanced platforms will emerge that provide the appropriate security solutions to their targeted deployment environment.

JXTA Technology Version 1.0

On 25 April 2001, the first prototype implementation was unveiled 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 Unix. The code should run on Windows 95, 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 more 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 of about 250 Kbytes. 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 second peer's permission, view its horizon, 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 their 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 Discovery 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 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 the Bay Area. In this case, the peer would like to multicast a message to a subset of the default World Peer Group. A subgroup can be formed especially for this purpose, but it seems more convenient and efficient to perform the multicast without forming 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 that indicates the scope for which a message is intended. Any peer receiving this message can propagate it based on the scope indicator. Using this approach, a sending peer should be bootstrapped with some well-defined 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 recoded to use the YML format.
- The use of XML does not imply that all peer nodes must be able to parse and create XML documents. For example, a cell phone with limited resources can be programmed to recognize and create certain canned XML messages, and still participate in a network of peers.
- To keep version 1.0 small, we used a lightweight XML parser that supports a subset of XML. We are working toward normalizing this subset according to an existing effort called MicroXML.

The JXTA Shell

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.

Networked Command-Line Interface

Like the Unix shell, the JXTA Shell provides interactive access to the JXTA platform via a simple command-line interface. With the Unix shell, users can learn a lot about how Unix works 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 execute in a networked environment. When a user command generates a sequence of message exchanges between a set of peers, some computation might occur on remote peer nodes and the answer is returned to the user.

You can use the JXTA Shell to play with the JXTA platform core building blocks such as peers, peer groups, pipes, and codats (content units that can hold both code and data). You 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. To the extent that makes sense, we have deliberately chosen to name JXTA Shell commands after the Unix shell commands, such as "ls" and "cat", in the hope that this makes the JXTA Shell more user friendly to Unix shell users.

JXTA Shell Commands

In our Java-based implementation, most shell commands are not built in *per* se. Rather, they are just Java language programs that 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.

Pipe Operator

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. The JXTA shell is more powerful in a number of ways. For example, 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 the JXTA Shell, because pipes are more permanent than the entirely transient ones in Unix systems, a user can dynamically disconnect and reconnect pipes between commands.

The JXTA Shell also supports piping in both directions, not just one. A special operator "<>" is used for creating crossing pipes between two commands. For example, with the following command "cmd1 <> cmd2", the output pipe of the first command is connected to the standard input pipe of the second command, and at the same time the output pipe of the second command is connected to the standard input pipe of the first command. Of course, this operator has to be used carefully to avoid infinite data loops.

Security

As mentioned earlier, at many places JXTA is independent of specific security approaches. Nonetheless, Version1.0 takes a first step toward providing a comprehensive set of security primitives to support the security solutions used by JXTA services and applications. More specifically, JXTA Version 1.0 attempts to provide the following security primitives:

- A simple crypto library supporting hash functions (such as MD5), symmetric encryption algorithms (such as RC4), and asymmetric crypto algorithms (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 nonmembers cannot access such data.
- A transport security mechanism that is modeled after SSL/TLS, with the exception that the unidirectional pipe does not allow it to perform a handshake, a crypto-strength negotiation, or a two-way authentication on a single pipe.
- The demonstration services called InstantP2P and CMS (content management service), which also make use of additional security features provided by the underlying Java platform.

NAT and Firewalls

The widespread use of network address translation (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. 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 to be picked up at a designated relay peer outside the firewall. The peer inside the firewall can initially reach outside the fire-

wall, 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 track a (local or remote) peer's status, control its behavior, and respond to actions on its part. It is very useful when a peer network wants to offer premium services with properties such as reliability, scalability, and guaranteed response times. 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 providers offering flat-rate services can benefit from being able to collect data and analyze usage patterns.

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 might choose to authenticate any command it receives. It might also decide not to answer queries from suspect sources.

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

Conclusion

JXTA provides a network-programming platform specifically designed to be the foundation for peer-to-peer systems. As a set of protocols, the technology stays away from APIs and remains independent of programming languages. This means that heterogeneous devices with completely different software stacks can interoperate through JXTA protocols. JXTA technology is also independent of transport protocols. It can be implemented on top of TCP/IP, HTTP, Bluetooth, HomePNA, and many other protocols.

We have developed a JXTA Shell, similar to the Unix shell, for writing scripts (see the sidebar, "The JXTA Shell"). Like the Unix shell, the JXTA Shell helps users learn a lot about the inner workings of JXTA during the process of writing scripts.

The open sourcing of JXTA technology through http://www.jxta.org, on 25 April 2001, marked a significant turning point for Project JXTA. Many developers have already started working to advance JXTA and related technologies. The following are among the areas where we expect to see immediate activities:

- 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 cell phones can become JXTA peers;
- A testbed scaling JXTA to thousands and millions of nodes;
- 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 the 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 topics, there are many issues that need substantial research and development work. We look forward to a productive year ahead with JXTA technology.

Acknowledgments

I am grateful to Mike Clary and Bill Joy for their management support and technical guidance, and to members of the JXTA engineering team and the JXTA Technology Advisory Council for their early contributions to Project JXTA. This report is adapted from "Project JXTA: A Technology Overview," 25 April 2001, available online at http://www.jxta.org.

Li Gong is director of engineering and head of the JXTA engineering team at Sun Microsystems. Previously, he led Sun's effort in home networking technologies and products. He received a BS and MS from Tsinghua University, Beijing, China, and a PhD from the University of Cambridge, England. He is a member of the editorial board for *IEEE Internet Computing*.

Readers can contact the author at li.gong@sun.com.