Project JXTA-C: Enabling a Web of Things

Bernard Traversat, Mohamed Abdelaziz, Dave Doolin,
Mike Duigou, Jean-Christophe Hugly, Eric Pouyoul

Project JXTA
Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
Bernard.Traversat@Sun.Com

Abstract

The Web, the collection of all devices connected to the Internet, is on the verge of experiencing a massive evolution from a Web of Computers to a Web of Things as new devices such as phones, beepers, sensors, wearable computers, telemetry sensors, and tracking agents connect to the Internet. The open-source Project JXTA was initiated a year ago to specify a standard set of protocols for ad hoc, pervasive, peer-to-peer computing as a foundation of the upcoming Web of Things. The following paper presents an up-to-date overview of the JXTA protocols and describes the latest implementation of the JXTA protocols in the "C" programming language. The paper discusses the overall architecture and trade off made to allow the JXTA-C implementation to run on a wide range of devices including supercomputers, servers, PCs, and memory-constrained embedded devices. The JXTA-C implementation delivers a small and efficient implementation of the JXTA protocols stack allowing the JXTA protocols to be embedded at the system level rather than the Java Virtual Machine (JVM) level for optimized performance and capabilities.

1. Introduction

The Web is on the verge of experiencing a massive evolution. From an Internet of nearly one hundred million computers, the Web is soon to become an Internet of nearly 100 *trillion* things [15]. Devices such as PCs, PDAs, phones, beepers, sensors, switches, wearable computers, telemetry sensors, and tracking agents are expected to connect to the Internet, flooding it with all kinds of information. Many of these devices are likely to create a spontaneous and unpredictable network, as devices will come and go at a much higher rate than today mainly static PC and server networks. This Web of things evolution is not likely to occur without problems.

The Web is close to reaching its fundamental growth and organizational limits, both in terms of the number of devices managed, and the number of contents published and searched. The most sophisticated search engines like Google are only able to index a fraction of the content available. Firewalls, NATs, and proximity network are creating complex partitioned network topologies left to be managed by application's developers. Simple denial-of-service attacks (e.g. overloading a URL address) have demonstrated the fragility and lack of resilience of the Web. The main computing model used on the Web is based on a client/server model where information and services are published at well-known static locations (URLs). Such location-dependent addressing schema is creating single points of failures and bandwidth bottlenecks to most popular Web sites. Centralized services like Distributed Name Service (DNS) are also creating network management nightmare already constraining the Internet's growth. Most home networks do not use DNS.

In the last couple of years, a more decentralized, *ad hoc*, peer-to-peer (P2P) computing paradigm has been employed by systems such as Freenet[10], Gnutella[11], and FreeHaven[12]. This paradigm presents the experience of a community environment, where each peer propagates and shares information in a fully decentralized, self-organizing, and open manner. Data and services are not owned by a particular member or peer, but are passed around, flowing freely towards the end subscribers (edge) without centralized control or management. Community members experience direct peer-to-peer interactions without mediating servers. The community as a whole ensures the protection and persistency of data through the network's unique ability to adapt, resist, and protect data by scattering multiple copies within the community. Reliability and resilience are created by making each member as interchangeable as possible, mimicking biologically redundant structures, such as bee or ant colonies.

Data are constantly migrating and replicating within the community, with data scattering schemes primarily based on access demands and popularity. When member

demands for some data increase, more copies are propagated and replicated within the community. When demand decreases, fewer copies are available as existing copies slowly disappear and are replaced by more popular data.

Peer-to-peer computing also fosters locality as data tend to transparently replicate towards the edge peers, where the data subscribers are located. Community members tend to interact more heavily with their neighbors for searching and accessing information, reducing overall network traffic and data latency, and leading to an overall better usage of available bandwidth. Daisy chaining and delegation capabilities enable members to forward requests to their neighbors for searching data beyond their own view. Within a community, each member can access the entire community's knowledge. There is no single, centralized search engine — every member participates in a search.

Project JXTA[1,2,3] is an open-source project (*www.jxta.org*) originally conceived by Sun Microsystems, Inc. and designed with the participation of a small but growing number of experts from academic institutions and industry. Project JXTA standardizes a thin and generic network protocol layer to build a wide variety of P2P applications.The following paper presents an up-to-date overview of the JXTA protocols and discusses the latest implementation of the JXTA protocols in the "C" programming language. Section 2. covers Project JXTA goals. Section 3. presents the JXTA virtual network abstractions (JXTA IDs, peergroup, advertisements, resolver and pipes). Section 4. discusses the overall design of the C implementation. Section 5. presents the implementation architecture, components, design trade off and implementation status. Section 6. discusses related works. Finally, Section 7. covers Project JXTA status and future directions.

2. Project JXTA

Project JXTA envisions a world where each peer, independently of software and hardware platform, can benefit and profit from being connected to millions of other peers. Project JXTA 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), service definitions (such as RMI and WSDL), and network protocols (such as IP or non IP). The Project JXTA protocols have been designed with the minimal requirements to be deployable on any device with a network heartbeat, including sensors, consumer electronics, PDAs, appliances, network routers,

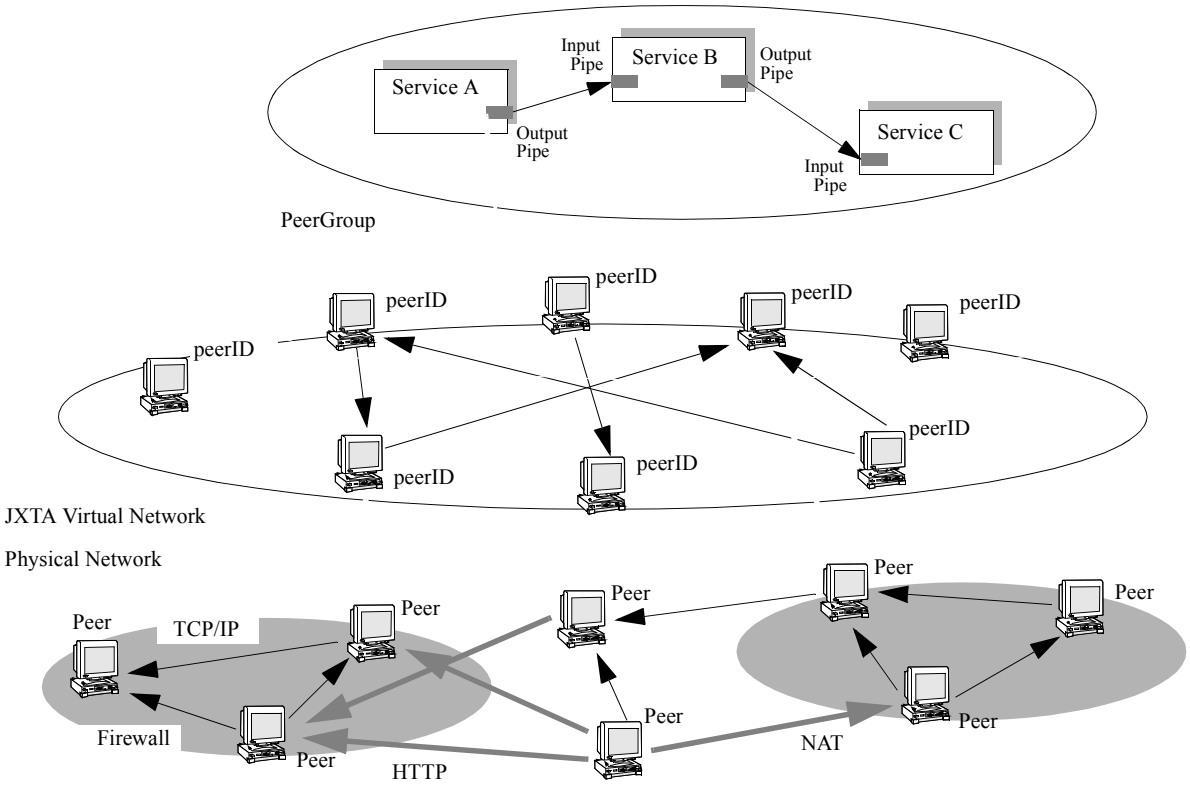

Figure 1. Project JXTA Virtual Network

desktop computers, data-center servers, and storage systems. Everything from cell phones to workstations and servers can access contents across multiple devices, regardless of location or network transport.

JXTA has become a leading P2P technology with a growing community of active developers and researchers engaged in more than 90 open-source JXTA-related projects. To date, there have been over 500,000 downloads of the source and binary code. ISVs are developing and shipping JXTA technology-based applications to a variety of markets including knowledge management and content delivery. Enterprises, government agencies, and educational institutions are making plans to adopt the technology as their primary P2P networking solution. The JXTA protocols have also been recently submitted to the IETF to drive the standardization of the protocols.

3. The Project JXTA Virtual Network

The Project JXTA protocols define a virtual network overlay on top of the existing physical network infrastructure upon which services and applications are built (Figure 1). The Project JXTA virtual network layer is designed to be thin and simple, but with powerful primitives for use by high-level services and applications. The main purpose of the JXTA virtual network is to hide all of the complexity of the underlying physical network topology (firewalls, NATs and proximity networks [6]), and thereby provide a uniform, addressable network for all peers in the network.

Messages are transparently routed, potentially traversing firewalls or NATs, using different transport/transfer protocols (Bluetooth, IrDA, TCP/IP, HTTP) to reach the receiving peers. The Project JXTA network allows peers to communicate without needing to understand or manage complex and dynamically changing physical network topologies. The intent is for the JXTA network to be pervasive and easy to implement on any transport and platform. The Project JXTA protocols standardize the manner in which peers discover each other, self-organize into peergroups, advertise and discover network resources, communicate with each other, and monitor each other. The network transport layer is built of a uniform peer addressing scheme based on peer ID, relay peers that relay messages between peers, and a binary message format to efficiently transport either binary or XML payloads.

3.1. Uniform Addressing

The Project JXTA network addressing model is based on assigning each peer resource a unique *ID* (the reference implementation uses 128-bit UUIDs). For example, a peer is uniquely identified by its *peer ID*, even if it uses different network addresses. A laptop booting with DHCP, and therefore having many different IP addresses, will always be addressed via the same peer ID. Similarly, a peer supporting multiple network interfaces (Ethernet, wireless Ethernet, etc.) can be addressed via its unique peer ID. A peer endpoint encapsulates all of the available network interface addresses for a peer. Peer endpoints are very much like business cards, listing the many ways to contact a person (phone, fax, email address, etc.). When receiving a peer endpoint advertisement, another peer can select the most efficient way to communicate with that peer from the list of available peer endpoint addresses (TCP/IP over HTTP).

3.2. Message Relaying

The Project JXTA network is an *ad-hoc*, multi-hop, adaptive network. Connections may be transient. Message routing is nondeterministic. Routes may be unidirectional and may change rapidly. JXTA uses two primary mechanisms for routing and relaying messages. One mechanism permits JXTA messages to contain a routing information element as part of their payloads. Every time a message goes through a hop, the routing element of the message is updated with the reverse route information to the source. When a peer receives a message, it can use the message routing element as a hint for routing replies to the sender. The other mechanism uses *relay peers* for maintaining routing information. Any peer may elect itself to be a relay peer. Relay peers maintain routing tables for relaying messages to their destination. Relay peers also offer the ability to spool messages for unreachable or temporarily unavailable peers, and act as bridges between physical networks. Relay peers play the role of landmark in the network facilitating the routing of messages between peers. As part of peer advertisements, peers can advertise a set of preferred relays to help shortcut route discovery.

3.3. Peergroups

Peers in the Project JXTA network self-organize into *peergroups*. A peergroup represents a set of peers that have a common set of interests, and have agreed upon a common set of policies (membership, content exchange, etc.). Each peergroup is uniquely identified by a unique *peergroup ID*. JXTA does not dictate when, where, or why peergroups are created. JXTA only describes how a peergroup is created, published, and discovered. Users, service developers, and network administrators dynamically create peergroups to scope interaction between peers, and match their application demands. A peer can belong to multiple peergroups at the same time. Peergroups form logical regions whose boundaries limit access to non-members. A peergroup does not necessarily reflect the underlying physical network boundaries such as those imposed by routers and firewalls. Peergroups virtualize the notion of routers and firewalls, subdividing the network into secure regions without consideration for actual physical network boundaries. Peergroups are spontaneously formed and self-organized based upon the mutual interests of peers. No particular rules are imposed on the way peergroups are formed, but peers with the same interests will tend to join the same peergroups. Peergroups serve to subdivide the network into abstract regions, providing an implicit scoping mechanism

restricting the flooding of discovery and search requests within the entire network.

Peergroups typically publish a set of services called *peergroup services*. Project JXTA specifies a standard set of "core" peergroup services with their associated protocols (*discovery*, *resolver*, *pipe*, *peer info*, and *rendezvous*). If a core protocol is not adequate for a more demanding peergroup, it can be replaced with a custom protocol. Project JXTA core peergroup services provide the basic functionality to support a peergroup's existence (publishing and discovering resources and exchanging messages). Peergroup creators can customize their peergroup services to match their peergroup requirements (centralized versus decentralized, deterministic versus indeterministic).

3.4. Advertisements

All network resources in the JXTA network, such as peers, peergroups, pipes, and services, are represented by *advertisements*. Advertisements are language-neutral metadata structures represented as XML documents. Project JXTA standardizes the following core advertisements: *Peer advertisement*, *PeerGroup advertisement*, *Pipe advertisement, Module advertisement, PeerInfo advertisement, Content advertisement*, and *Peer Endpoint* advertisement. Peers cache, publish, and exchange advertisements to discover and find available resources. Peers discover resources by searching for their corresponding advertisements. All advertisements are published with a lifetime that specifies the availability of the associated resource in the network. An advertisement can be republished to extend the lifetime of its resource before the previous advertisements expire. Lifetimes provide a simple and robust mechanism for the network to repair itself in a fully decentralized environment.

3.5. "Just-in-time" Resource Binding

JXTA provides a universal "just-in-time" resource binding mechanism, the *resolver,* to perform all resolution operations found in traditional distributed systems, such as resolving a peer name into an IP address (DNS), binding an IP socket to a port, locating a service via a Directory service (LDAP), or searching for content in a distributed filesystem (NFS). The dynamic and ad hoc nature of P2P systems make "just-in-time" resource binding resolution an essential component of JXTA. In JXTA, all resolution operations are unified under the simple discovery of one or more advertisements. All binding operations are implemented as the discovery or search of one or more XML documents. Project JXTA does not specify how the search of advertisements is performed, so it is providing a default resolution policies for core advertisements. Each peergroup can tailor its resolver implementation to use a decentralized, centralized, or hybrid search approach to match the peergroup requirements.

The Project JXTA network provides a bootstrapping resolver service based on *rendezvous* peers. Rendezvous peers are peers that have agreed to cache a large number of advertisements in support of a peergroup. Rendezvous is conceptually like a virtual town square, where peers gather for exchanging and trading information. A peergroup can have as many rendezvous peers as needed. Each peergroup has its own set of rendezvous peers, and any peer can potentially become a rendezvous peer. The rendezvous peer infrastructure provides a simple, but powerful, self-adapting mechanism for discovering advertisements while providing hooks that allow high-level search services to participate in the advertisement search process. Rendezvous enables the most popular advertisements to be potentially found in a minimum 1 hop search, regardless of the size of the peergroup, as popular advertisements tend to aggregate on rendezvous function of their popularity. Rendezvous maintains a cache of advertisement based on a LRU scheme. Rendezvous peers maintain a loosely-couple view of other rendezvous to propagate requests. Advertisement queries are only forwarded between rendezvous. Edge peers are not allowed to forward request to limit flooding. It is important to point out that since each peergroup has its own set of rendezvous, query propagations only occur within the scope of a peergroup, not the entire network. Edge peers index their advertisements on their preferred set of rendezvous. Rendezvous uses their known rendezvous view to *walk* the rendezvous. JXTA provides a pluggable walker framework so each service can configure different walker policies. JXTA defines a *limited-range walker* (find and stop) and a *propagation walker* (walk through the entire view) policies to propagate queries within the rendezvous view. Rendezvous provide a more efficient mechanism for searching popular advertisements than homogeneous hash-key distributed indexed networks[4,5], which always require log(N) hops even as advertisement is more popular than another one. The rendezvous organization also improves locality between groups of peers that are exchanging the same kinds of advertisements, thus leading to higher locality and faster search.

3.6. Pipes

Pipes are virtual communication channels used to send and receive messages between services and applications. Pipes offer a virtual abstraction over the peer endpoints, providing the illusion of virtual in and out mailboxes that are not physically bound to a specific peer location. Pipes can connect one or more peer endpoints. At each endpoint, software exists that can send, receive, or manage message queues or streams. Message queues are optional. The receiving end of the pipe is referred to as the *input pipe*; the sending end, the *output pipe* (see Figure 1). Pipes are published and discovered using pipe advertisements, and are uniquely identified by a Pipe ID. Pipe ends are dynamically bound to a peer endpoint at runtime by the resolver. Using the pipe abstraction, applications and services can

failover transparently from one physical peer endpoint to another in order to mask a service or peer failure, or to access a newly published instance of a service. The pipe binding process consists of searching and connecting the two or more ends of a pipe. When a message is sent into a pipe, the message is sent by the local output pipe to the destination input pipe currently listening to the pipe. Bi-directional, reliable, and secure pipe services have been implemented on top of the core pipe service.

4. Project JXTA-C

Project JXTA-C (*jxta-c.jxta.org*) is an open-source community project aimed at delivering a fully compliant implementation of the Project JXTA protocols, as specified by the JXTA Protocols Specification (Version 1.3.4 [16]), in ISO C and POSIX. Project JXTA-C demonstrates the platform independence of the JXTA protocols by fully interoperating with the existing Java™ 2 Platform, Standard Edition (J2SE™) Reference Implementation of the JXTA protocol[17]. Project JXTA-C is the latest complete implementation of the JXTA protocols. Project JXTA-C peers have the same level of interoperability and functionality as J2SE edge peers. JXTA-C peers fully interoperate with existing J2SE relay and rendezvous peers for connecting to the JXTA virtual network. JXTA-C peers can discover both C and Java peers, join the NetPeerGroup, and discover and create new peergroups that can be joined by either C or J2SE peers (as long as a Java or C implementation of the peergroup services is available). JXTA-C peers can publish, establish pipe connections, and send and receive messages between either C or Java peers.

4.1. No Protocol Proxy for Small Devices

Project JXTA-C delivers direct peer-to-peer connectivity to the small device world, embedding the JXTA protocols without requiring protocol proxy services. A proxy solution will create a bottleneck and single point of failure. As the native TCP/IP stack is embedded at the system level, not at the JVM level, Project JXTA-C guarantees optimize performance, functionality, and smallest memory footprint. The JXTA-C implementation can have direct access to the underlying OS resources (thread, memory and network transport). For instance, virtually all Java Virtual Machines are relying on native implementations of the TCP/IP stack. We expect the same to be true for the JXTA protocols.

Project JXTA-C supports peer-to-peer interactions by allowing peers that have direct physical connections to communicate with each other without involving a protocol-based proxy service. Peers that do not have a direct physical connection, due to firewall or NAT partitioning, use JXTA relay peers to relay messages as in the J2SE implementation. The Project JXTA protocols have been designed to be implemented easily on unidirectional links and asymmetric transports. In particular, many forms of wireless networking do not provide equal capability for devices to send and receive. Project JXTA permits any uni-directional link to be used when necessary, improving overall network connectivity in the system. The intent is for the Project JXTA protocols to be pervasive and easy to implement on any transport. Implementations on reliable and bi-directional transports such as TCP/IP or HTTP should lead to efficient bi-directional communications.

A JXTA-C peer only needs to implement the protocols that it requires. For example, a device may have all the advertisements it uses stored in memory, so the peer does not need to implement the Peer Discovery Protocol [2]. A peer may use a pre-configured set of relays for routing all its messages. Hence it does not need to implement the Peer Endpoint protocol [2], but just sends messages to its relays for forwarding. A peer may not need to obtain, or wish to provide, monitoring information to other peers, and therefore does not need to implement the Peer Information Protocol [2]. The implementation needs to be modular enough so only the components required can be configured.

4.2. JXTA Services and Applications in Java

It is important to point out that while the protocols are implemented in C, we do expect most JXTA applications to be implemented in Java to take full advantage of Java code mobility and sandbox security framework. A JNI interface layer could be implemented for accessing JXTA-C from a Java application. Only single function devices with extremely limited capabilities where code download is not required, and devices not running Java are expected to use non-java services and applications.

4.3. Project JXTA-C Architecture

The Project JXTA-C architecture defines three layers: 1) the porting layer (OS/Runtime specific), 2) the protocols layer (OS/Runtime independent), and 3) the application layer (Figure 2).

4.3.1 Porting Layer

In order to facilitate the porting of the JXTA-C implementation to other OS runtime environments (Microsoft Windows, UNIX®, and RTOS), all OS-specific dependencies are isolated to a platform-specific layer. The porting layer encapsulates all OS-specific runtime dependencies such as memory allocation, thread management, and networking particularities. The porting layer defines a porting API, isolating the generic protocol implementation from OS-specific dependencies. Porting the JXTA-C implementation to a new runtime environment only requires re-implementing the porting layer. The use of OS dependencies in the entire implementation has been kept to its minimum. There is no use of high-level constructs that are not likely to be supported on small RTOS. The JXTA-C architecture greatly simplifies the porting of the C implementation to new runtime environments. We anticipate that

the Project JXTA-C implementation will be ported to a wide range of devices from servers, PCs, cellular phones, PDAs, and sensors, to embedded devices. Project JXTA-C is expected to lead to high-performance implementations of the protocols for running JXTA relay and rendezvous peers by taking full advantage of the native runtime environment. The implementation minimizes the number of threads.

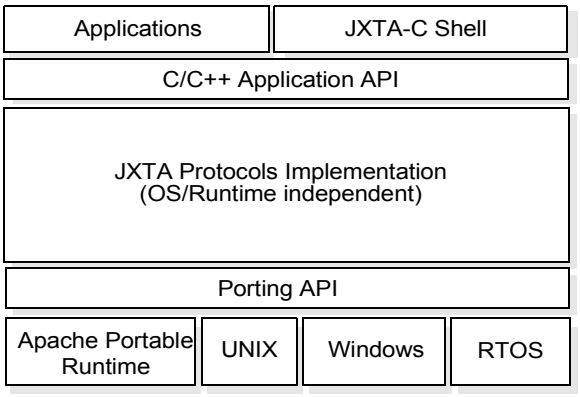

Figure 2. JXTA-C Implementation Overview

4.3.2 Protocols layer

The protocols implementation layer implements the core JXTA protocols. This layer is built on top of the porting layer and is completely isolated from OS/Runtime specifics. To optimize modularity, each protocol is implemented as a separated component or service. This is key point to enable developers to only configure the components they need. The modularity of the implementation enables to scale down features for supporting very small devices.

4.3.3 Application layer

The application layer defines a C/C++ application-level API to be used for developing JXTA-C applications in C or C++. Include files are provided with a JXTA-C dynamic library as part of the Project JXTA-C standard developer kit. Header files have been generated to be imported in both C and C++ applications.

Project JXTA-C also provides a demo Shell interpreter (JXTA-C Shell) with a set of commands to be run from the standard Shell or Windows command line. JXTA-C Shell commands, such as *whoami, talk, peers, groups, and join*, use the same syntax and have the same semantic as the J2SE JXTA Shell (*shell.jxta.org*) commands. The JXTA-C Shell interpreter is provided as a demo environment to showcase JXTA-C functionality.

4.4. Apache Portable Runtime (APR) Environment

The Apache Portable Runtime (APR) [7] was selected as the first port environment for Project JXTA-C for a number of reasons. APR defines a porting layer that we used as the initial porting layer for Project JXTA-C. This leverages the work already performed by the Apache open-source community to isolate OS dependencies. The APR environment is available on most UNIX (Solaris™ Operating Environment, Linux, AIX, etc.) and Windows environments. Using APR makes JXTA-C run on both windows and UNIX. The APR environment is well known to open-source community developers, enabling Project JXTA-C to reach the maximum number of open-source developers. Finally, the Apache license is compatible with the JXTA license. Selecting APR enables the initial JXTA-C implementation to be run on the maximum number of deployed environments with minimal porting effort. Native ports to the Solaris Operating Environment or Windows is expected to be done for maximum performance.

4.5. Single-Address Space Approach

In order to address very small embedded devices where only a single application may be running at a time, a single-address space approach is used. Each JXTA-C application is linked with the Project JXTA-C runtime library creating a single-address space application. No shared object library or application server daemon is used. This design provides the most efficient implementation limiting the number of thread context switches and allowing sharing of memory between the library and the application. The low resource usage overhead is also essential for running a peer as relay or rendezvous, so we can get optimize message routing performance. We expect this design to be extended to create a service daemon that can be shared by multiple applications running on a same peer.

5. Project JXTA-C Implementation Overview

The following section presents an overview of the main components of the C implementation and their relationships. Each component defines its own set of APIs to interact with other components. We attempted to create a modular implementation that enables easy replacement or extension of components. The implementation uses a share object model allowing efficient sharing of data structures between components for optimizing performance and reducing object copies. For example, the HTTP transport, router and endpoint services are able to shared the same copy of a message, not requiring multiple copies to be made for minimizing memory usage and reducing message latency. The C implementation design provides a blueprint for developers to implement the JXTA protocols.

5.1. HTTP Transport

The HTTP transport implements the HTTP transport binding as specified in the JXTA protocols specification.

This component enables a peer to request a lease, and to connect to a relay peer for sending and receiving forwarded messages. Project JXTA-C initially implemented the HTTP transport for supporting the traversal of firewalls and NATs. The TCP/IP transport is expected to increase performance and will be implemented shortly.

5.1.1 Message

The message service implements the JXTA wire binary message format used when messages are sent in the JXTA virtual network. JXTA uses a binary format for ensuring efficient transfer of both binary and XML payloads. All JXTA protocol messages are represented as XML payloads. Messages are formed as an ordered sequence of elements. Each element has a unique name, length and MIME-type. It is important to point that the JXTA wire format representation is agnostic of data representation. Any kinds of data can be sent. Each service when processing a message can add or remove its own message elements. Message elements permit to isolate the different parts of a message allowing greater protection. For instance, the Router service is only allowed to manipulate the Router element message, but does not touch any other elements in the message.

5.1.2 Endpoint Service

The endpoint service implements the JXTA endpoint abstraction, which is used for encapsulating multiple endpoint transports into a single virtual peer endpoint. Endpoints provide the virtual network abstraction that is used by peers to communicate independently of the underlying network location (firewalls or NAT) and physical transport. The endpoint service maintains the mapping between a peer ID and its available peer endpoint transports. The endpoint service multiplexes messages over the available endpoint transports, optimizing communication over the most efficient transport.

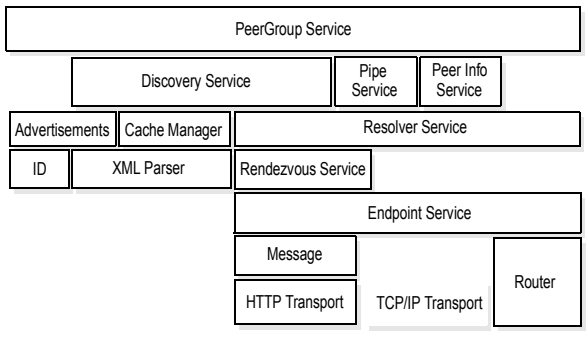

Figure 3. JXTA-C Implementation Architecture

5.1.3 Router

The router implements the routing service used by a peer endpoint for discovering and maintaining route information to other peers. Some peers may not have a direct connection, so messages need to be relayed through one or more relay peers to their final destination. The router implements the Endpoint Routing Protocol (ERP), allowing a peer to query and discover route information from its known set of relays. The router maintains a in-memory routing table of discovered routes. Each message sent and received contains a routing element used for maintaining route information. When receiving a message, the router payload is examined by the router for adding or updating its route information. The router provides route information to the Endpoint service for delivering messages.

5.1.4 Rendezvous Service

The rendezvous service is used for propagating messages within the scope of a peergroup. This service implements the RendezVous Protocol (RVP). The rendezvous service allows client peers to obtain a lease for propagating messages. The rendezvous service uses the endpoint service for propagating messages.

5.1.5 Resolver Service

The resolver service is used for sending generic query/response (asynchronous RPC) requests within the scope of a peergroup. The Resolver component implements the Peer Resolver Protocol (PRP). The resolver uses the endpoint and rendezvous service for unicasting and propagating requests within the scope of a peergroup.

5.1.6 Discovery Service

The discovery service is used for discovering and publishing any type of advertisement (peer, peergroup, pipe, module, etc.) in a peergroup. The discovery service implements the Peer Discovery Protocol (PDP). The peer discovery service uses the resolver service for sending and receiving discovery requests. The discovery service uses the cache manager for persistent storage of advertisements.

5.1.7 Pipe Service

The Pipe service is used for creating and binding pipe ends (input and output pipes) to peer endpoints within the scope of a peergroup. Two types of pipes are implemented, Unicast (one-to-one), and Propagate (one-to-N). The pipe service uses a pipe resolver service for dynamically binding a pipe end to a peer endpoint. The pipe resolver service implements the Pipe Binding Protocol (PBP).

5.1.8 Cache Manager

The cache manager is used for persistent storage and caching of advertisements. This component enables the efficient storage and retrieval of advertisements. Advertisement objects are serialized and deserialized using the

cache manager interface. The cache manager provides persistent storage of advertisements across restarts. The cache manager implements an efficient index mechanism for retrieving advertisements and optimizes retrieval time for the core JXTA services (Resolver, Discovery, and Pipe). The cache manager controls the decay of advertisements in the cache. Each advertisement is published in the cache with an associated time-to-live expiration date. Advertisements are deleted from the cache when they have expired.

5.1.9 PeerGroup Service

The PeerGroup service implements the bootstrapping of a peer in the NetPeerGroup and the peergroup navigation. The service exposes all the core NetPeerGroup services (Discovery, Resolver, Pipe, Rendezvous, etc.) to applications that have joined the NetPeerGroup. The Project JXTA-C NetPeerGroup implementation is fully compatible with the J2SE implementation. The PeerGroup service enables a peer to create, advertise, and join a new peergroup. Peergroups export a set of PeerGroup services that are represented as modules. The PeerGroup service manages the underlying infrastructure for advertising and loading modules. The PeerGroup service manages the PlatformConfig configuration file (peer name, transport info).

5.1.10 XML Parser

The XML parser provides generic parsing of XML documents. The parser enables the serialization and deserialization of JXTA-C objects into output and input XML character streams. A lightweight XML parser, *Expat* [9], is used. The JXTA protocols use a minimal subset of XML. Small devices may not need a parser as all protocol messages may be hardcoded.

5.1.11 Advertisements

The advertisement module implements the core advertisements used in the JXTA Protocols: *Peer*, *PeerGroup*, *Endpoint*, *Rendezvous*, and *Module* and *Content* advertisements. The advertisement module allows a C structure to be serialized and deserialized into an XML document representing an advertisement.

5.1.12 ID

This component implements the JXTA IDs (128 bits UUID) as specified in the JXTA protocol specification.

5.2. Status

The JXTA-C implementation has been completed in less than 3 months and fully available at *jxta-c.jxta.org*. The implementation is compact (less than 100 files) and provide a full implementation of all six JXTA protocols. The full implementation runs in less than 150 KB. This compares with the 25 MB heap size of the Java Reference implementation. The implementation is fully compatible with the Java reference implementation. As previously mentioned, peers only need to implement the protocols they need. Not all services or components may be necessary. Some ports of the Project JXTA-C implementation may remove some services, like router, discovery or peer info, on limited functionality devices.

6. Related Works

A number of vertical P2P applications Freenet[10], Gnutella[11], Freehaven[13], Jabber[14], Groove[18] have been developed. Most of these applications provides a single function environment for either file sharing, instant messaging or collaborative works. Project JXTA provides a generic platform for implementing any kinds of P2P applications. Groove attempts to provide a set of generic APIs, but it only runs on Windows and it is a proprietary system. It is anticipated that Microsoft .Net[19] product will support some P2P instant messaging enhancements. But, these will also be proprietary and limited to a SOAP over HTTP transport. The JXTA protocols are transport agnostics and can be implemented over TCP/IP or non-IP networks. JXTA also introduces the concept of virtual peergroup domains that is not available on any of the above systems to scope and secure peer interactions.

7. Conclusion

Project JXTA was initiated to define a common set of protocols for building *ad hoc* peer-to-peer applications. Project JXTA has become a leading industry P2P platform with a growing community of active developers and researchers. To date, there have been over 500,000 downloads of the technology. ISVs are developing and shipping JXTA technology-based applications to a variety of markets including knowledge management and content delivery. Enterprises, government agencies, and educational institutions are making plans to adopt the technology as their primary P2P networking solution. The JXTA protocols have been recently submitted to the IETF to drive the standardization of the JXTA protocols.

The JXTA virtual network standardizes the manner in which peers discover each other, self-organize into peergroups, advertise and discover network resources, communicate, and monitor one another. Project JXTA-C is the latest implementation of the JXTA protocols in the "C" programming language. Project JXTA-C delivers a small and efficient implementation of the JXTA protocols stack. It delivers peer-to-peer computing to the world of small devices without requiring any proxy services and can be easily ported to single-threaded, small memory, embedded devices. We expect the Project JXTA-C implementation to lead to high-performance optimized implementations of the protocols for running relay and rendezvous peers on high-end servers. Project JXTA-C enables embedding the JXTA protocols at the system level rather than at the JVM level for optimized performance and functionality.

8. Acknowledgments

We would like to thank Bill Joy, Mike Clary, Ingrid Van Den Hoogen, Bill Yeager, Juan Soto, Matt Reid, Sheriff Botros, Gene Kan and Yaroslav Faybishenko from Sun, Dorothea Wiarda as the JXTA-C Shell community contributor, and the entire Project JXTA community for helping us refine the Project JXTA protocols.

9. References

[1] Project JXTA, *www.jxta.org*.

[2] B. Traversat *et. al*., *The Project JXTA Virtual Network*, *http://www.jxta.org/docs/JXTAprotocols.pdf*.

[3] S. Oaks, B. Traversat, L. Gong, *JXTA in a Nutshell*, O'Reilly Press, 0-596-00236-X, Sept. 2002.

[4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A Scalable Content Addressable Network*, ACM SIGCOM, 2001.

[5] F. Dabek, E. Brunskill, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica, and H. Balakrishnan, *Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service*, 2001.

[6] LongWork Network, *http://www.echelon.com/products/Core/protocol/Default.html*.

[7] Apache Portable Runtime, *http://apr.apache.org*.

[8] Bill Joy, *The 6 Webs*, *http://content.techweb.com/wire/story/TWB20000111S0009*.

[9] Expat XML Parser, *http://expat.sourceforge.net*.

[10] Freenet, *www.freenet.sourceforge.net*.

[11] Gnutella, *www.gnutella.wego.com*.

[12] FreeHaven, *www.freehaven.net*.

[13] Jabber, *www.jabber.org.*

[14] T. Dierks and C. Allen, *The TLS Protocol*, IETF RFC2246. January, 1999.

[15] Greg Papadopoulos, *The Next Big Thing*, *http://www.sun.com/analyst2002/presentations/Papadopoulos_WWAC_020702.pdf.*

[16] Project JXTA Protocols Specification, *http://spec.jxta.org/v1.0/docbook/JXTAProtocols.pdf*.

[17] Project JXTA J2SE Reference Implementation, *platform.jxta.org*.

[18] Groove, *www.grove.net.*

[19] .Net, *www.microsoft.com.*