JXTA Search: Distributed Search for Distributed Networks

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

Introduction

JXTA Search started life in June 2000 when Gene Kan and Yaroslav Faybishenko of Infrasearch created a demo search engine which connected multiple Web servers together using a peer-to-peer protocol. This demo consisted of a Web front-end to a distributed set of servers, each running a node hardcoded to respond to certain types of queries. The founders of Infrasearch conceived the idea to distribute queries to network peers best capable of answering them. For example, the Infrasearch search engine responded intelligently to queries for:

¥ Text, by hitting a *Moreover News* database
¥ Stock quotes, by sending the query to *Yahoo! Finance*
¥ Pictures, by using *OnlinePhoto Lab*
¥ Arithmetic, by sending the query to a calculator

Nearly a year later, the Infrasearch team is now part of Sun's Project JXTA (*http://jxta.org*), but has the same missions and objectives   to provide a common distributed query mechanism for devices from Web servers to small computers, effectively p2p-ing the Web via JXTA and Infrasearch s distributed search technology. In addition, the original Infrasearch demo has been extended to work not only on the Web, but also as part of the JXTA framework. Infrasearch now provides the default searching methodology for the JXTA framework in the form of *JXTA Search*.

The JXTA Search Network

Communication over the JXTA Search network is performed via an XML protocol called the *Query Routing Protocol* (QRP). The QRP defines mechanisms for sending and responding to queries in the JXTA Search network, as well as mechanisms for defining meta-data for nodes in the network. The JXTA Search network consists of the following participants:

¥ *JXTA Search Information Providers,* anything that responds to requests formatted in the JXTA Search QRP language. Information providers may be JXTA peers or Web servers, such as *cnn.com*.

¥ *JXTA Search Consumers,* anything that makes requests in the JXTA Search QRP language. Consumers may be JXTA peers or Web sites with HTTP client interfaces to the JXTA Search network.

¥ *JXTA Search Hub*, a mechanism that facilitates efficient query routing over the JXTA Search network by handling message routing between consumers and providers. Providers register a description of themselves with the hub and wait for matching requests. Consumers query the network through the hub and await responses. Hubs can also be providers or consumers   they can be chained together in a network.

Consumer applications send requests to the JXTA Search network via the nearest JXTA Search hub. The hub determines which of the known providers should receive the query based on provider meta-data. The hub sends the requests to providers, receives responses, and sends responses back to consumers.

In many applications, a program functions in a *peer-to-peer* manner, acting as both a provider and a consumer. Physically, the provider is either an individual computer or a load-balanced set of machines. Typically, a consumer is an individual computer, although it may be a Web service. The network is a cloud of machines. Providers and consumers contact the network through a specific hub machine that provides virtual access to the entire network. (Typically providers and consumers contact different hub machines.)

At first glance, JXTA Search is merely a *meta-search engine* for distributed networks. However, in contrast to conventional techniques for meta-search, such as HTML scraping, JXTA Search defines a common protocol for the exchange of query and response. This Query Routing Protocol enables participants in the network to exchange information in a seamless manner without having to understand the structure of their presentation layers.

Applications for this technology are seen in a wide variety of domains, from public accessible Web search, to private networks of trading partners, to interaction between distributed services and applications. An example of the application of this network to efficiently route queries in peer-to-peer frameworks is embodied in Sun s JXTA Search JXTA binding. The Gnutella community has found similar approaches desirable for scaling its network in the form of Clip2 s Reflector (*http://clip2.com*).

Another example of the application to deep Web search is demonstrated in Sun s JXTA Search Web client. The need for such technology in the public Web domain is evidenced by a recent study published in the Industry Standard (http://www.thestandard.com/article/0,1902,18134,00.html), which estimates over 550 billion content documents are contained in distributed information. In comparison, Google, the leading search engine, is capable of searching only 600 million pages out of an estimated 1.2 billion static pages. In addition, increased usage of application servers and Web-enabled business systems is resulting in a rapidly growing amount of content unsearchable by conventional means.

Wide and Deep Searches with JXTA Search

JXTA Search is intended for two complementary search types: *wide* and *deep*. The concept of the JXTA expanded Web supports both the wide search of distributed devices, such as PCs, handhelds, cell phones, and the deep search of rich content sources such as Web servers (Figure 1).

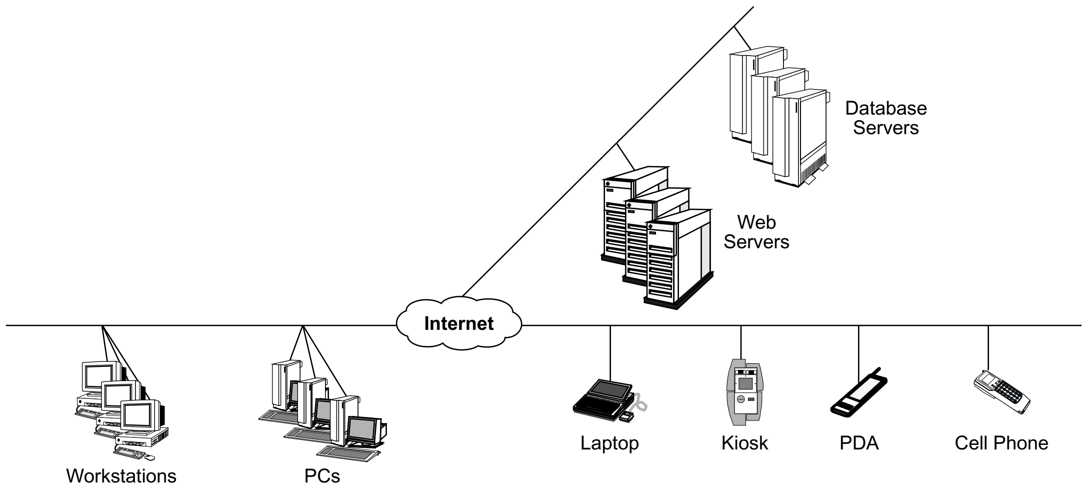

Figure 1: JXTA Search supports both wide and deep network searches.

¥ Wide search

A variety of methods exist for executing wide searches. One alternative is a fully distributed search in which each peer sends its queries to all known peers, an expensive and inefficient use of bandwidth and search speed resources. At the other extreme is a mechanism in which a server peer handles all queries for all peers in the network. In such a scheme, the server peer is not only a single point of failure in the network, but also a likely query bottleneck. In contrast, JXTA Search provides an efficient mechanism for distributing queries across a wide network of peers. The compromise employed by JXTA Search consists of a series of *hub peers,* each of which handles the queries for a group of peers. Each hub is intended to specialize in some way, such as geography, peer content similarity, or application. Hubs forward queries to other hubs if the queries cannot be satisfied, or if the search is to be expanded to the widest number of peers possible.

¥ Deep search

JXTA Search is also designed for deep searches in networks, such as JXTA and the Web. Conventional crawler-based search engines, like Google, are ideal for indexing static, slowly changing Web pages such as home pages or corporate information pages. In contrast, deep search engines find information embedded in large databases such as product databases (e.g. Amazon) or news article databases (e.g. CNN). Rather than crawling such databases and indexing and storing the data, JXTA Search determines which queries should be sent to them, directs these queries, and returns the results. The search results are intended to be more up-to-date and have wider coverage than a set of conventional crawler search engine results.

The Benefits of a Distributed Approach

A distributed approach such as JXTA Search is more suited to deep search than a crawler-based search engine in three important ways:

¥ *Speed of update*. Currently, the fastest crawlers take at least 3 months to crawl and index the Web. Given that current estimates of the deep Web size are at least 10 times larger than the crawlable Web, the time to crawl and index such resources is prohibitive given the frequency of update (roughly every 30 days). Although targeted or restricted crawling of headline or other meta-data is possible (such as that done by *moreover.com*), this is an inefficient method for most deep searches due to access issues.

¥ *Access*. Unlike static Web pages, deep Web resources typically do not have a *table of contents* or similar index. Consider a large online seller with millions of product descriptions in its databases that does not have a set of pages listing all these descriptions. Crawling such a resource requires the database to be queried repeatedly with every conceivable query term until all products are extracted. While this is possible, it is an inefficient approach that falls victim to the speed of update issue discussed previously. In addition, since many pages are generated dynamically given information about the consumer or context of the query (time, purchasing behavior, location, etc.), a crawler approach is likely to lead to data distortion. It is partly for this reason that some companies have been resistant to permitting other search engines to query their databases remotely. Finally, in some situations, content may be inaccessible due to access privileges (e.g. a subscription site), or for security reasons (e.g. a secure content site).

¥ *Efficiency.* According to Infrasearch investor and Netscape founder Mark Andreessen, it s counterintuitive to centralize search . Consider the role of a crawler accessing dynamic content from a site such as Cnet *news.com*. The content is created by editors and stored in a database as XML or other presentation neutral form. The *news.com* application server then renders the content as a Web page with the associated links using the current *news.com* templates. The end user sees a nicely presented page with the desired story. However, when a crawler hits the page it merely sees HTML. To extract the content of the story, the crawler must use information gleaned by a person about the structure of the HTML page to *scrape* the content and headline from the page. This content, or a processed version, is stored for indexing purposes in its own database, enabling the retrieval of the link and story when a matching query is submitted.

This process of moving data from a database to HTML and back again is inherently inefficient and prone to errors. In addition, the content provider has no control over the format of the article or the decision about which article to show in response to a query. In contrast, JXTA Search approach specifies a common query request and response protocol which gives both parties more flexibility and control over data exchange.

Architecture

JXTA Search is an open network framework based on the JXTA framework for distributed information routing that extends the JXTA framework. JXTA Search enables information providers to publish a description of queries they are willing to answer. Information consumers submit queries to the network, which routes each query to all interested providers.

The JXTA Search framework consists of:

¥ *The Query Routing Protocol (QRP)*. The QRP is an XML protocol for defining queries, responses and registrations. The QRP is designed to allow both structured, lightweight and efficient query message exchange.

¥ *Queryspaces*. Since providers may have widely differing types of content or resources in their datastores, the notion of queryspaces is allowed in the QRP to define the structure of a query and its associated registration. Queryspaces are similar in concept to XML namespaces. In their simplest interpretation, queryspaces define the structure in XML of a valid query against a provider. Queryspaces are explained in more detail later in this section.

¥ *Registration*. Providers register with the JXTA Search network, identifying the queries to which they wish to respond. The registration consists of an XML-based encoding of a logical statement characterized by a queryspace.

¥ *Query Formulation*. Users and applications present queries to the JXTA Search network as arbitrary XML adhering to specific queryspaces.

¥ *Query Resolution*. Queries are resolved by a *resolver* by matching query terms to registration terms. Providers whose registration terms match the query terms are returned by the resolver.

¥ *Query Routing*. Queries are routed to the appropriate provider by sending XML requests over HTTP. The *Router* sends all requests and awaits responses. It continually monitors providers to determine availability and reliability.

¥ *Provider Responses.* Providers respond to queries in arbitrary XML.

Note that the JXTA Search network does not perform any presentation of provider responses. JXTA Search collates all results from providers, performs ranking on the results with respect to the query, and returns them to the requesting client. The client performs the final presentation of the results. It is assumed that the client performs such presentation either as a Web page or as a client-side user interface. In principle, general applications query the network and act on the response as they see fit.

Design Goals

The JXTA Search protocol has the following design goals:

¥ *Simplicity.* A minimally-conforming client implementation can be built using existing libraries for manipulating XML and sending either JXTA or HTTP messages. A minimally-conforming server implementation can be built with either a JXTA reference implementation or a generic HTTP server.

¥ *Structure*. All queries to the JXTA Search network are made using XML messages conforming to a particular queryspace in which information providers register *templates* describing the structure of queries to which they are willing to respond.

¥ *Extensibility.* Arbitrary queryspaces can be used on the JXTA Search network there is no need for centralized queryspace management, simplifying ad-hoc collaboration.

¥ *Scalability.* The JXTA Search network is intended to support millions of publishers and consumers performing billions of transactions per day. Sophisticated implementations can take advantage of advanced connection management features provided by lower-level protocols.

JXTA Search was designed as an abstract query routing service for networks with arbitrary messaging and transport mechanisms. It currently binds to two kinds of network: the JXTA framework (XML over JXTA using either HTTP or TCP) and the Web (XML over HTTP). This document describes both bindings and the differences between them. In general, examples interchange freely between JXTA peers and HTTP servers or clients. In practice, some of the internal mechanics of the system differ, but the principles remain the same. The protocols used between the two bindings are identical, as are the resolution mechanisms. The major differences occur between the router and the client interfaces.

Figure 2 illustrates the JXTA Search network architecture. In this schematic of the JXTA Search/JXTA framework, each JXTA peer can run instances of the Provider, Consumer, and Registration services on top of its JXTA core. Each peer interacts with the JXTA Search Hub Service, itself running on top of the JXTA core.

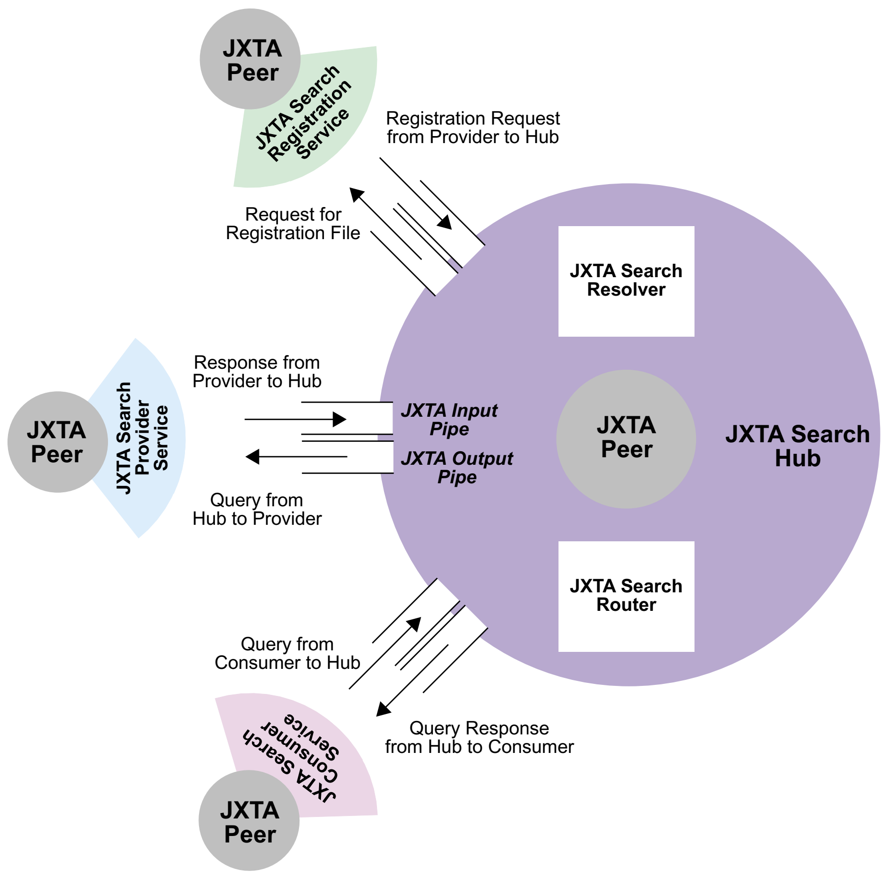
JXTA Peer
JXTA Search Registration Service
Registration Request from Provider to Hub
Request for Registration File
JXTA Search Resolver
Response from Provider to Hub
JXTA Peer
JXTA Search Provider Service
JXTA Input Pipe
JXTA Output Pipe
JXTA Peer
JXTA Search Hub
Query from Hub to Provider
JXTA Search Router
Query from Consumer to Hub
Query Response from Hub to Consumer
JXTA Search Consumer Service
JXTA Peer

Figure 2: The JXTA Search network architecture.

The JXTA Search network architecture consists of the following components:

¥ *JXTA Search Provider Service*, a service which accepts queries written in the QRP from the hub (or potentially JXTA Search consumers directly) and responds in the QRP to the requestor. The Provider Service does not perform any indexing or searching internally, but rather calls the appropriate interface on the Provider itself, such as JXTA CMS, database search, etc.

¥ *JXTA Search Consumer Service*, a service which sends queries written in the QRP to the hub (or potentially to JXTA Search providers directly) and awaits QRP responses. The Consumer Service performs formatting of the responses for presentation to the end user or application.

¥ *JXTA Search Registration Service*, a service which sends requests for registration to the hub and maintains the registration file for the provider.

¥ *JXTA Search Hub Service*, a service which performs routing of queries from consumers to providers. The Hub Service accepts queries, resolves those queries to the appropriate providers, manages the routing of the queries to the providers, and sends collated results back to the requesting consumer. The Hub Service consists of two sub components:

    ¥ *JXTA Search Router*, software which routes and manages query connections, collates results and returns results to consumers.

    ¥ *JXTA Search Resolver*, software which matches queries to providers using a full text search engine which indexes meta-data specified by the provider during registration.

Message Flow

Applications find information providers by sending queries XML messages with essentially arbitrary structure into the network via a specific access point. There are no restrictions on query tags. For a query to be sent to an information provider, two conditions must be met:

¥ *Matching queryspaces*. The information provider must have sent a registration message with a queryspace which matches the queryspace of the query.

¥ *Matching predicates*. The path predicate specified in the registration message must select a non-empty set of nodes inside the query.

Once an information provider receives a matching query, it composes a response and sends it back to the network. The network collates all responses and returns them to the querying application. Note that the network does not actually evaluate competing relevance rankings; that task is left to the application.

The fact that the network collates responses and sends them back to the user helps achieve simplicity as clients are not required to listen for asynchronous responses. Collation is also important for security reasons. For example, collation helps prevent distributed denial-of-service attacks based on spoofed queries. However, it is anticipated that network messages will be used often to establish peer-to-peer connections.

The response message is arbitrary XML, enabling the network to provide implicit support for any XML protocol to be carried as payload in the JXTA Search protocol. For example, SOAP messages can be carried between providers and consumers using JXTA Search to match particular method tags or identifiers. By utilizing simple, powerful enveloping protocols and matching engines, JXTA Search enables developers to build network applications that facilitate the discovery and interaction of providers and consumers without the need for location or resource information.

The protocol does not require queries or responses to identify machine addresses. Certain queryspaces may agree to share addresses explicitly (e.g. peer-to-peer file sharing), while other queryspaces may choose to share addresses implicitly (e.g. with embedded XHTML). The structure of both the query and the response is specified, explicitly or implicitly, by the chosen queryspace.

Distributed Search

Although a single hub is scalable enough to potentially handle hundreds of thousands of providers, in practice it is desirable to run separate hubs for separate types of applications or specialist domains. For example, a Web site might run a hub as a vertical aggregator of content pertaining to Java programming. Its providers would be other sites with content focused on Java. However, it could also send queries to a hub running on a more general technology news site whose providers would be sites such as *CNet* or *Slashdot*. Similarly, in a peer-to-peer network, hubs are expected to become an efficient way to group peers with similar content, geography or queryspaces.

While the protocol does not define methods for chaining hubs together, it does contain mechanisms for supporting multiple hubs. These mechanisms include support for multiple responses wrapped in a response message, and the specification of the query lifetime for which the result is valid. It is conceivable to develop more sophisticated schemes for maintaining multiple hubs, such as the automatic generation of hub registrations from provider indexes. However, these approaches are not considered in this document.

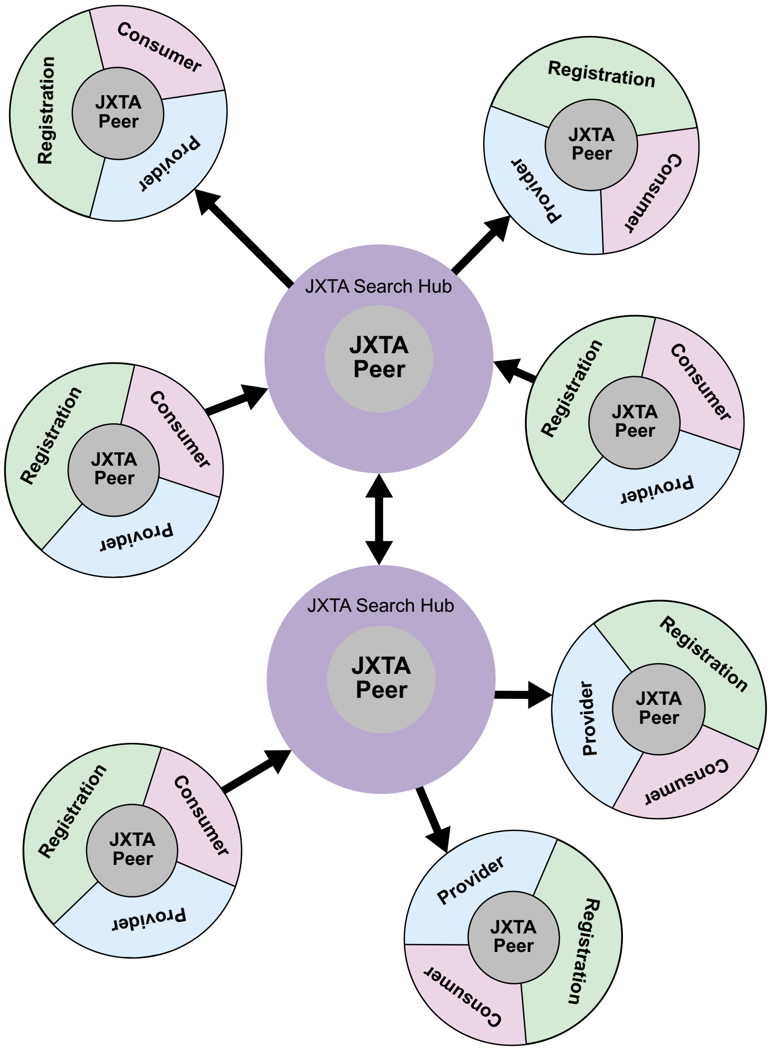

Figure 3: In a distributed JXTA Search, each peer within the network interacts with the hub using its appropriate service.

Queryspaces

A *queryspace* is a unique identifier for an abstract space over which a query travels. Queryspaces are a fundamental component of the JXTA Search framework. Like XML namespaces, queryspaces are identified by unique URIs. Like XML namespace URIs, queryspace URIs do not necessarily reference actual content they are simply identifiers used by providers and consumers to find each other.

The JXTA Search protocol makes no assumptions about the syntax or semantics of queryspaces. JXTA Search does not currently process queryspaces, nor does it attempt to validate queries and responses. Today, queryspaces are simply used for coordination between consumers and providers. Queryspaces should generally incorporate the following information:

¥ *Structure*. Queryspaces should allow providers and consumers to agree on the structure of messages. As a result, a queryspace should specify structural constraints in a standard form, such as a DTD or an XML-Schema.

¥ *Semantics*. Providers and consumers must also agree on the meaning of exchanged messages. While structural information can be machine-readable, semantic information usually is intended for developers writing client and server software.

¥ *Ranking*. Queryspaces should explain how clients should sort the results received. Typically, ranking is application-dependent, and some applications may not require any ranking to be performed.

JXTA Search does not specify methods for exchanging queryspace information the protocol simply ensures that providers receive only queries that match their queryspaces. However, JXTA Search encourages efficiency by enabling providers to filter received queries by including a predicate with each registered queryspace. The predicate is applied to each candidate query in the given queryspace. Only queries that match the query are sent to the provider. Internally, JXTA Search uses the predicates to optimize routing.

Each query should contain at least one query section which can contain arbitrary XML that conforms to the specified queryspace. Otherwise, the query is not likely to match any information provider predicates, resulting in no received responses. (As previously mentioned, JXTA Search does not attempt to validate the query.) If multiple query sections are specified, the information provider can select the query to which it is to respond.

The Query Routing Protocol

The QRP consists of the following components: a *query request*, a *query response*, and a *registration*.

The details of the QRP are detailed in the JXTA Search protocol document located at *http://search.jxta.org/protocol.html*. This document simply provides examples of each message type.

The following two examples are employed throughout the remainder of this document:

¥ A query using a simple text query space for articles on JXTA, Java and other terms
¥ A query using a product queryspace for books matching certain authors, titles, etc.

An explanation of how these queries, their responses, and associated registrations by providers are implemented in the JXTA Search framework is discussed.

Query Messages

The simple text query for the term JXTA in the *http://search.jxta.org/text* queryspace is as follows:

```
<?xml version='1.0'?>
<request xmlns= http://search.jxta.org
   xmlns:t= http://search.jxta.org/text
   query-uuid= 1C8DAC3036A811D584AEC2C23
   query-space= http://search.jxta.org/text >
   <t:query>
      <t:text>
         sunw
      </t:text>
   </t:query>
</request>
```

Query messages are structured as follows:

¥ The default namespace is *http://search.jxta.org*. JXTA Search currently does not support other default namespaces.
¥ The query message is contained within the envelope `<request>...</request>`.
¥ The query unique ID is specified in the `uuid` attribute of the `<request>` tag.
¥ The query space is specified in the `query-space` attribute of the `request` tag
¥ The query data can be arbitrary XML within a namespace that matches *http://search.jxta.org/text*, which includes the tag `<query>` to specify the start of the actual query data and the tag `<text>` to specify free text, or within any other namespace specified by the `query-space` definition.

A more complex example is shown below:

```
<?xml version='1.0'?>
<request id= 1C8DAC3036A811D584BEC2C63
    query-space= http://bigbookseller.com/js
    xmlns= http://search.jxta.org
    xmlns:b= http://bigbookseller.com/JxtaSearch >
   <b:query>
       <b:author>
```

```
            <quote>Bill Joy</quote>
          </b:author>
          <b:title>
            Java
          </b:title>
      </b:query>
   </request>
```

In this example, the query space is defined as *http://bigbookseller.com/JxtaSearch*. The namespace `b` therefore matches the URI for this query space, a requirement for JXTA Search messages using queryspaces. The query specifies that the `author` within the name space `b` should be `Bill` or `Joy`, or that the title should contain `Java`.

Response Messages

A response to the first query answered by a JXTA Peer running a stock quote service appears as follows:

```
<?xml version='1.0'?>
<responses xmlns= http://search.jxta.org
          xmlns:t= http://search.jxta.org/tickerqs
          query-uuid= 1C8DAC3036A811D584AEC2C23 >
  <response>
    <data>
     <t:ticker>
        SUNW
     </t:ticker>
     <t:price>
        20.59
     </t:price>
    </data>
  </response>
</responses>
```

The response message is structured as follows:

¥ The default name space is*http://search.jxta.org*.
¥ The response message is enveloped within the <`responses`>...</`responses`> tags, with each specific response enveloped in <`response`>...</`response`> tags.
¥ The body of the response is contained within the <`data`>...</`data`> tags. It can be arbitrary well-formed XML. In this case it conforms to the XML namespace *http://search.jxta.org/tickerqs* (a fictional XML schema for ticker symbols).

Providers can be any peer within the network at the moment, either JXTA peers running the JXTA Search service plug-in, or HTTP peers adapted to the QRP.

A response to the second message appears as follows:

```
<?xml version='1.0'?>
<responses xmlns= http://search.jxta.org
   xmlns:b= http://bigbookseller.com/JxtaSearch
   query-uuid= 1C8DAC3036A811D584BCC2C23 >
  <response>
    <data>
     <b:author>
      Bill Joy, Guy Steel, James Gosling, Gilad Bracha
     </b:author>
     <b:URL>
      http://www.bigbookseller.com/0201310082
     </b:URL>
     <b:title>
```

```
      The Java Language Specification, Second Edition (The Java Series)
     </b:title>
     <b:price>
       $39.95
     </b:price>
     <b:abstract>
       A definitive technical reference for the Java programming
       language, written by the inventors of the technology.
     </b:abstract>
   </data>
  </response>
</responses>
```

Registration Messages

Information providers must register with the JXTA Search network. To register, a provider contacts an access point with a registration message. A registration message is an XML document with three components:

- ¥ The JXTA Search network queryspace URL, identifying the URL at which, when queries are posted to it, the provider s predicates are checked for matches.
- ¥ A set of predicates.
- ¥ The provider’s query server endpoint   either a JXTA pipe ID or a URL. Queries which match one of the provider’s predicates are posted to this endpoint.

Registrations are XML documents containing meta-data about the information the provider wishes to expose.The registration must contain the following tags:

```
<register>...</register>
<predicate>... </predicate>
```

The predicate defines the structure and content of the queries in which the provider is interested. The next section describes the rules for forming predicates and the implementation of the resolver.

Consider the following registration file for a provider:

```
<?xml version='1.0'?>
<register xmlns= http://search.jxta.org >
   <title>JXTA Stock Quote Provider</title>
   <link>http://search.jxta.org</link>
   <description>Given a ticker symbol, returns a 15-minute delayed quote
   </description>
   <query-server>jxta://
59616261646162614A757874614D5047CF403C5700D44AE68F9FB626DD3F18E500000000
000000000000000000000000000000000000000000000000401</query-server>

   <query-space uri="http://search.jxta.org/text">
      <predicate>
         <query>
            <text>sunw aol orcl</text>
         </query>
      </predicate>
   </query-space>
</register>
```

Registrations are structured as follows:

¥ The registration body is enveloped within <register> ... </register> tags.

¥ The query server, the URL or Pipe ID of the provider to which queries should be sent, is specified within <query-server>...</query-server>.

¥ The*predicate*, the logical statement which queries must match in order to be routed to this provider, is enveloped within <predicate>...</predicate> tags.

¥ The predicate contains the queries to be matched by this provider, each enveloped within <query> ... </query> tags. Each predicate can contain multiple <query> envelopes.

¥ The query body can contain arbitrary XML as long as it matches the namespace that matches the specified query-space for this provider.

¥ In this example the provider specified by the query-server*jxta://...*. (the pipe ID of the JXTA node) has registered for the query <text>sunw aol orcl</text>,. Queries containing the text terms *sunw*, , *aol* or *orcl* are routed to the JXTA peer..

An example of a live registration on the Web, O Reilly s Meerkat service has recently been adapted to the JXTA Search service. The following is an example of their registration (they are registering for all queries in the *search.jxta.org/text* space):

```
<?xml version='1.0'?>
<register xmlns= http://search.jxta.org >
<title>O'Reilly Network</title>
<link>http://meerkat.oreillynet.com/</link>
<description>
     The Source for Open and Emerging Technologies.
</description>
<image>
     <url>http://meerkat.oreillynet.com/icons/meerkat-powered.jpg</url>
     <width>88</width>
     <height>31</height>
</image>
<query-server>http://www.oreillynet.com/meerkat/jxtasearch/</query-
server>
<query-space uri="http://search.jxta.org/text">
     <predicate>
       <query>
          <text>
       <  /text>
       </query>
     </predicate>
</query-space>
</register>
```

Query Resolution

Query resolution is the process of determining to which set of providers a given query should be routed. Clearly, sending all queries to all providers is inefficient. As a result, JXTA Search attempts greater efficiency by:

¥ Defining a framework for providers to register the type of queries they are interested in receiving
¥ Providing an efficient query resolution and routing service

The minimal condition for matching a query to a provider is that the query must have the same query-space as the provider registration. The set of providers is selected by the resolver in the following order:

¥ Providers which have all clauses of at least one predicate satisfied. In order to match a predicate, a query is first tokenized into a set of patterns. Providers are ranked based on the matched pattern scores.

¥ Providers which do not have a matching predicate, but are similar in their responses and have the same query-space as providers who have a matching predicate.

¥ If the number of providers returned is still less than the maximum, a provider is selected at random from the same query-space as the query. This enables the exploration of the provider content in case the provider registration file is incomplete or is not updated frequently.

Providers may specify the type of queries they wish to receive in their registration file. This file encodes the type an structure of queries, queryspaces, and response formats the provider is interested in receiving. This file can be thought of as an advertisement of the provider s meta-data and its structure. The registration file specifies:

¥ The query server to which queries should be directed. In the Web domain, the query server is a CGI script capable of processing the QRP request messages and responding with a QRP response.

¥ The queryspace of the queries accepted by the provider.

¥ The structure and content of the queries the provider is interested in receiving, specified in predicate form.

Consider a possible registration from our previous *bigbookseller.com* example:

```
<?xml version='1.0'?>
<register xmlns="http://search.jxta.org"
   xmlns:b= http://bigbookseller.com/jxtasearch >
   <query-server>
   http://bigbookseller.com/exec/jxtasearch.pl
   </query-server>
   <query-space uri= http://bigbookseller.com/jxtasearch >
      <predicate>
         <query>
            <b:author>
               <quote>Bill Joy</quote><quote>Neal Stephenson</quote>
            </b:author>
            <b:title>
               Java JXTA XML Cryptography
            </b:title>
         </query>
      </predicate>
   </query-space>
</register>
```

This registers a provider with the text queryspace specified by *http://bigbookseller.com/jxtasearch*. This registration registers the provider for the following queries:

¥ Any query containing*Bill Joy* or *Neal Stephenson* in the <author> field, or
¥ Any query containing*Java*, *JXTA*, *XML* or *Cryptography* in the <title> field

Queries matching these conditions are directed to the query server running at *http://bigbookseller.com/exec/jxtasearch.pl*. In practice, predicates are much larger and typically have a more complex structure.

Query Node Patterns (QNPs)

Query Node Patterns are used to match predicates against queries. Each query is expanded into a series of QNPs.

Query Node Patterns (QNPs) are XML fragments which form the basic building blocks of query predicates in registrations. In this implementation, QNPs restrict the structure of registration predicates. Each QNP matches a node of an XML query. A match occurs when a QNP matches some subset of a query's structure, or, more formally, when they can be constructed by a series of the following transformations: 1) deleting a node in the query; or 2) replacing the query with a subnode of itself.

For example, consider the following query:

```
<request>
   <object type=file>
      <format>jpeg</format>
      <desc>foo bar</desc>
   </object>
</request>
```

This query is matched by the QNPs shown in Table 1.

QNP	Matches Queries with...
<object></object>	An OBJECT node
<object type=file></object>	An OBJECT node with parameter TYPE=file
<format>jpeg</format>	A FORMAT node with jpeg in its text
<object><format></format></object>	An OBJECT node containing a FORMAT node
<object><desc>foo bar</desc></object>	An OBJECT node containing a desc node with *foo* or *bar*

Table 1: QNP matching

QNPs contain the following restrictions:

¥ *Text tag tokenization.* In QNP matching, tag text or character data is tokenized at whitespace breaks and considered a set of tokens. Matching is case-insensitive. The `<quote>`*phrase*`</quote>` format may be used to specify a phrase.

¥ *Single Path Restriction*. In a registration, a QNP can only contain one path through the query XML[1]. As a result, the valid query from the previous example is an invalid QNP registration:

```
<object>
   <format>jpeg</format>
   <desc>foo</desc>
</object>
```

The invalid QNP registration must instead be specified as a predicate containing the conjunction of two separate QNPs, as follows:

```
<and>
   <object>
      <format>jpeg</format>
   </object>
   <object>
      <desc>foo</desc>
   </object>
</and>
```

1. This restriction may be removed at a later date.

Tag text is an exception to the single path restriction. If a QNP node contains multiple text tokens, the tokens form an implicit disjunction[1]. For example, the following is a valid registration:

```
<object>
   <format>jpeg gif png</format>
</object>
```

Query Predicates

A query predicate is a boolean expression composed of QNPs. Predicates must be in conjunctive normal form a conjunction of disjunctions[2].

Consider the following query predicate:

```
<predicate>
   <and>
      <object type=file>
      <object><format>jpeg</format></object>
      <or>
         <desc>bar</desc>
      </or>
   </and>
</predicate>
```

Note that the first two conjuncts are implicit disjunctions. When an `<or>...</or>` tag contains only a single QNP the `<or>...</or>` may be eliminated. Similarly, if the top-level contains only one element, the `<and>...</and>` may also be eliminated. Therefore, at its simplest, a predicate can be of the form `<predicate>foo bar </predicate>` which matches any query containing the word *foo* or the word *bar*.

Building the Index from the Provider Registration

The resolver implementation is similar to a full text search engine. Inspired by Doug Cutting s work on Lucene (*http://lucene.com*), the resolver indexes all tags and text in the registration files. A reverse index is created which maps query terms to providers. For efficiency, the resolver creates separate indices for each query space.

A resolver consists of a set of indices, one for each query-space. When a provider sends a registration file, the resolver parses it into a set of predicates. Each predicate has a set of clauses, and each clause has a set of disjunctions[3]. Each predicate is assigned a global unique predicate ID, and each clause a local clause ID. For each pattern in the registration, a posting is created which contains the predicate ID and the clause ID. The predicate ID and clause ID are used to trace the pattern to the clause in the registration where the pattern occurs. The pattern/posting pair is stored in the corresponding query space index. The posting also contains a score which is updated based on feedback received from the user. (Provider scoring is discussed later in this document.)

Following is an example of a simple XML fragment of two predicates from a registration and the corresponding index entries:

```
<predicate>
   <and>
      <object type=pics>
      <object><format>jpeg</format></object>
      <or>
         <desc>foo</desc>
         <desc>bar</desc>
      </or>
   </and>
</predicate>
```

1. This could be done automatically. However, in some cases the size of the predicates may be increased significantly. It is currently not done automatically in an effort to encourage providers to construct smaller and faster predicates. With this restriction, the efficiency of a predicate is proportional to its size.
2. This restriction may be removed in the future.
3. Currently, predicates are restricted to be in conjunctive normal form.

```
<predicate>
<predicate>
   <and>
      <object type=recipes>
      <object><format>recipes</format></object>
      <or>
         <title><quote>Rhubarb Pies</quote></title>
         <title><quote>Scrambled Eggs</quote></title>
      </or>
   </and>
</predicate>
```

The index contains eight entries. At least three of these entries must match a query for the query to be routed to the provider. Following are the corresponding index entries:

```
object&type=pics    ◊ (predicate0, clause0)
object>format>jpeg  ◊ (predicate0, clause1)
desc>foo            ◊ (predicate0, clause2)
desc>bar            ◊ (predicate0, clause2)
object&type=recipes ◊ (predicate1, clause0)
object>format>recipes◊ (predicate1, clause1)
title>Rhubarb Pies  ◊ (predicate1, clause2)
title>Scrambled Eggs◊ (predicate1, clause2)
```

Scoring

As mentioned previously, a score is associated with each pattern/posting pair in the resolver index. Scoring is used to determine the popularity of providers for a particular type of query. Scoring is not an essential part of the resolver, but if present, helps select the most popular providers relevant to the query.

Scoring works as follows. If a user sends feedback in response to a query response, the pattern/posting pairs that matched the query are retrieved from the corresponding query-space index, and their scores updated. (Scores are increased for positive feedback or decreased for negative feedback). In the current JXTA Search implementation, a simple score update formula is used, however, more complex algorithms can be conceived. The following describes the formula used, where (`0 < alpha < 1`) determines the rate of change of the score.

```
Score(t+1) = (alpha) * Score(t) + (1 – alpha) * Feedback
```

Provider Similarity

In the event there are very few providers who match a query, JXTA Search may select providers who did not match the query, but who have registered the same query-space and who are similar to a provider who matched the query.

Concepts similar to collaborative filtering are used in determining provider similarity. In essence, providers who tend to match the same queries are considered more similar. The current JXTA Search implementation maintains a similarity matrix in the resolver. The entries in this matrix determine the degree of similarity between two providers.

Query Routing

The router performs the following functions:

¥ Receives the queries from the end-user application or consumer
¥ Routes the queries to the appropriate providers
¥ Merges the results of the queries and presents them to the end-user application

When the router receives a request from the network, it asks the resolver for a list of nodes on the network that are registered as wanting to receive queries similar to this one. Once the resolver returns a set of network node endpoints and network node IDs, the router routes the query to the set of providers.

The router uses an exponential back-off algorithm to prevent spamming slow or temporarily downed hosts. If a provider exceeds a set timeout, the resolver subsystem is alerted that the provider may be inactive and that further resolutions should not include this provider.

JXTA Router

For JXTA networks, the router works as follows:

¥ Opens an output pipe to the provider end point
¥ Sends the message to the provider end point using the pipe
¥ Accepts responses from providers on a dedicated input pipe

The JXTA router has several components:

¥ A component that can receive requests from JXTA peers
¥ A component that can route queries to JXTA peers
¥ A component that can receive responses from JXTA peers

The component that receives requests from JXTA peers simply listens to an input pipe for query requests. When the query request arrives, the resolver determines the set of peers to which to route the query. If the peer is a JXTA-based peer, the JXTA router sends the request over an output pipe to that peer's input pipe. The JXTA router has one input pipe dedicated to receiving query responses from JXTA peers. When a sufficient condition has been met to flush responses back to the requesting peer, the JXTA router sends the request peer a query response message.

HTTP Router

For HTTP networks, the router works as follows:

¥ Opens a connection to each provider over HTTP
¥ Sends the message to the provider over this connection
¥ Awaits responses from providers over this connection

To promote efficiency, the HTTP router also uses KEEP_ALIVE to maintain a connection to each provider already queried. Multiple requests to this provider are made over a single connection, remembering for a given provider the queue of requests. The goal is to prevent repeated opening and closing of connections to providers. Experience shows this scheme to provide substantial performance improvements over a naive approach.

Platform Bindings

As described previously, JXTA Search is network and message format agnostic. Currently, two platform bindings are supported: JXTA and Web bindings.

JXTA Search over JXTA

JXTA Pipes provides a simple way to transport Query Request, Query Response, and Registration messages. JXTA Search's QRP protocol suite maps to JXTA Pipes in a straightforward manner. In each case, the JXTA Search message is enveloped by a JXTA Message.

For Query Request messages, the JXTA Message has two tag/value pairs: `request` and `responsePipe`. The actual Query Request message is stored as the value of the `request` tag. The pipe advertisement for the pipe the peer wishes to receive the responses on is stored as the value of the `responsePipe` tag. Using an output pipe, a peer delivers the Query Response JXTA Message to the input pipe of a JXTA Search peer.

Query Response messages have only one tag/value pair: `responses`. When a JXTA Search peer has obtained an answer to a Query Request, it opens an output pipe to the pipe specified in the Query Request message's `responsePipe` tag and sends the Query Response JXTA Message with the `responses` tag filled with the response.

Registration messages have two tag/value pairs: `registration` and `responsePipe`. The registration document is stored inside the `registration` tag. The pipe advertisement for the pipe the peer wishes to receive the responses on is stored as the value of the `responsePipe` tag. Using an output pipe, the peer sends this message to a JXTA Search hub. The peer receiving the registration processes the registration and sends back a success or failure code to the pipe specified by the `responsePipe` tag in the registration message.

JXTA Search over HTTP

The JXTA Search network is currently implemented over HTTP transports as well as JXTA. Using HTTP, the Query Request is sent as an HTTP post to a *provider adapter* which processes the request. For example, the following posts the query message to the adapter *JxtaSearch.jsp*:

```
POST /JxtaSearch.jsp HTTP/1.0
Content-Type: text/xml
<?xml version='1.0'?>
....
```

Queries are sent to providers with HTTP. A POST request is used.The content type of the request is *text/xml*. The body of the request contains the query, an XML document.

Although designed to be queried by end users or applications, JXTA Search also provides a consumer focused Web front-end for querying providers and presenting responses. This front-end performs the following functions:

¥ *Aggregation of responses*. Provider responses are returned by the router and aggregated by the front-end.
¥ *Presentation of responses*. Responses are presented in raw HTML format as they are received by the router from providers.
¥ *Query ranking*. Responses are ranked according to the relevance of the query to the responses.
¥ *Provider signup facilities*. Providers may sign-up to register endpoints and monitor statistics.

Examples

JXTA Search is intended as a network protocol and framework for query routing in generic applications. There are two reference implementations   JXTA and HTTP hubs   intended for use in JXTA and Web networks. There are currently two applications for these domains: a simple JXTA information provider and consumer, and a Web site which acts as a consumer, sending queries to a JXTA Search HTTP hub. See *http://search.jxta.org/demo* for a working example.

Jxta Information Provider and Client

One example provided that simulates the behavior of a future JXTA information provider is the implementation of a stock quote server on a JXTA peer. This server connects to *Yahoo!finance* upon receipt of a query, and parses the result from the Web site. In the future, this server could be implemented on a JXTA peer located at *Yahoo* or another provider. In this example, the JXTA peer registers for all symbols in the NYSE, NASDAQ and AMEX exchanges, and responds to requests in the standard *http://search.jxta.org/text* queryspace in the format understood by the Web search client.

Web Search Client

The example Web search client uses a servlet and JSP implementation and resembles a standard Web search engine. The Web client sends queries in the *http://search.jxta.org/text* queryspace and expects responses in a simple XML format described in the protocol specification located at *http://search.jxta.org/protocol.html*. The Web client formats the results received as a list of items in HTML.

To demonstrate the potential of JXTA Search on the Web, the Web client also contains a *proxy engine* which understands the format of certain Web search interfaces. As a result, queries and responses can be sent to and received from certain sites which are not yet running JXTA Search adapters.

In addition, the Web client contains utilities for providers to sign up and monitor registration, and for users to send specific preferences to the hub.

Summary

JXTA Search is a novel approach for query routing in distributed networks. Using a simple XML protocol combined with powerful but simple indexing matching engines JXTA Search provides developers with the capability to connect multiple consumer and provider applications together for the purposes of information discovery and exchange. With the growth of distributed networks such as JXTA and the predominance of Web and other network services, we anticipate a growing need for flexible solutions such as JXTA Search.

A number of applications lend themselves to a distributed search model, including:

¥ *Consumer Web search*. JXTA Search is well suited to a consumer Web search engine in part because it is orthogonal to current crawler-based approaches. JXTA Search can be used in conjunction with a *Google* or *Inktomi* search engine as a complementary discovery engine. While crawler-based approaches are ideal for static content, JXTA Search is ideal for deep, dynamic content such as news, product information and auctions.

¥ *B2B networks*. JXTA Search is also well suited to B2B networks, such as exchanges and supply chain networks. A peer-to-peer approach is potentially more efficient than the conventional data synchronization approach of replicating buyer and seller data at the exchange. Using a private network version of JXTA Search, trading partners can search for information across a range of partner databases all connected via a common query protocol. In addition, since JXTA Search enables the specification of arbitrary queryspaces for searching, partners can rapidly adapt existing corporate databases to communicate via the query routing network.

¥ *Extranet applications.* JXTA Search may prove valuable in the integration of extranet resources between business partners. Consider the case of a customer complaining to a vendor about a problem with a PC. The customer service representative (CSR) likely is faced with the problem of searching multiple partner databases to find a solution to the problem. Using JXTA Search, CSRs could rapidly integrate all Web-enabled databases from their partners and search them in a consistent fashion.

Opportunities

JXTA Search is now available as open source software under the Sun JXTA license. Sun encourages developers and users to download, use, and extend the code base and help foster use of the JXTA Search network.

There are several opportunities for extending JXTA Search, including:

¥ *Open standard support*. Although JXTA Search is capable of supporting arbitrary XML, there is potential to integrate JXTA Search with existing standard initiatives, such as RDF for describing meta-data and queryspace vocabularies, and XMLRPC.

¥ *Private networks*. It would be interesting to add private network support to JXTA Search. This would enable B2B networks and extranet applications. This may include quality of service (QoS) provisioning, security via public key infrastructures, and explicit B2B queryspace support.

¥ *Spam prevention*. Spam may become a problem for public JXTA Search applications if providers register for terms for which their content is not representative. Spam may be addressed in a variety of ways, including comparison of the site registration to an inferred registration, tracking of searches made and results returned, and allowing consumer input (voting).

Resources

More information, including source code, documentation, mailing lists and example demonstrations, can be found on the JXTA Search main Web site located at *http://search.jxta.org*.

Acknowledgements

First, I would like to thank the entire JXTA Search engineering team: John Beatty, Sherif Botros, Tom Camarda, David Doolin, Yaroslav Faybishenko, Gene Kan and Cody Oliver. I would also like to thank the JXTA engineering team, in particular Li Gong for valuable feedback. The JXTA Search team would also like to thank Mark Andreessen, Doug Cutting, Rael Dornfest, David Galbraith, Spencer Kimball and Graham Spencer for their valuable contributions.

Draft 1.0 May 29, 2001