# Service Location Protocol: A Java Prototype

**Jack Caldwell**

*Columbia University*

**April 28, 1998**

## ABSTRACT

The Internet continues to grow at exponential rates, offering a significant number of services to clients; however, finding the service that meets a client's criteria and connecting to a specific service provider, like a mail server, requires explicit knowledge of the service provider's host name or IP address. Furthermore, the process of finding and selecting services cannot easily be automated. What is needed is a dynamic self-discovery mechanism whereby clients can locate services without prior knowledge of where that service is located, or which servers can meet the client's specific service criteria. The Service Location Protocol (SLP) internet draft, proposes a specification for providing this capability, enabling access to services in a self-configuring environment. *My research focuses on prototyping a minimal implementation of this protocol, in order to prove its value in the local area network, as well as determine optimal values for parameters specified in the protocol* .

# TABLE OF CONTENTS

**ABSTRACT** ..... **1**

**TABLE OF CONTENTS** ..... **2**

**INTRODUCTION** ..... **2**

**PROTOCOL OVERVIEW** ..... **3**

- MINIMAL CONFIGURATION ..... 3
- SERVICE TYPE SPECIFICATION ..... 3
- EXTENDED IMPLEMENTATION ..... 4
- SERVICE REGISTRATION LIFETIME ..... 4

**PROGRAM DOCUMENTATION** ..... **4**

- WHY JAVA? ..... 4
- INSTALLATION ..... 5
- APPLICATION DESIGN ..... 6
- PROCESS ARCHITECTURE ..... 6
- PROTOCOL PARAMETERS ..... 7

**CONCLUSION** ..... **8**

- RECOMMENDED PROTOCOL ENHANCEMENT ..... 8

**FUTURE WORK** ..... **8**

**RELATED WORK** ..... **8**

- RDU ..... 8
- LDAP ..... 9

# INTRODUCTION

Clients of distributed, network-enabled services are increasingly mobile, and most of them either cannot administer their computing devices or prefer not to do so each time they move into a new local area network. Users prefer simpler access to services in a self-configuring environment that does not require knowledge of the services's configuration details. Administrators prefer the benefits of modifying network addresses for distributed services, even dynamically assigning network addresses, without having impact on every client of that service. Additionally, the protocol provides some degree of fault tolerance, since similar service providers may coexist in a local environment as shared resources, as clients discover which providers are online at any given discovery period. The SLP internet draft proposes a protocol enabling a user to connect to an environment with no prior knowledge of which services are locally available, nor where those services are located.

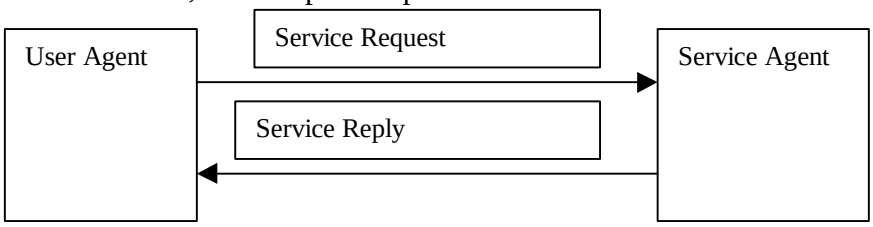
This project produces a minimal peer to peer implementation of the SLP specification, although the protocol defines a scalable architecture allowing centralized caching of services.

The next section, **Protocol Overview**, gives an overview of the SLP specification. The design of the prototype is discussed in the section, **Design**. The results of prototype testing are described in **Results and Analysis.** Alternative mechanisms for locating services are discussed in the section, **Related Work**. The final sections **summarize** this project and recommend **Future Work** related to this topic.

# PROTOCOL OVERVIEW

## *Minimal Configuration*

The minimal configuration of SLP requires two agent processes: the *User Agent* (UA) acts on behalf of a client to acquire service information, and the *Service Agent* (SA) acts on behalf of a service provider to disseminate information about the location and attributes of the service. In its most basic form, SLP is peer to peer:




However, the initial service request is multicast to the SLP group address, since the UA does not know where the Service Agents are located. The UA must be prepared to receive multiple responses, since every SA within range that meets the service criteria of the request will respond. Starting the process of service discovery requires only the knowledge of a single well-known multicast group address, defined exclusively for SLP. Once a UA knows where and how to connect to a specific SA, subsequent requests are unicast directly to the SA.[1] Thus, multicasting is limited to initial service discovery.

## *Service Type Specification*

Another important element of SLP is its specification of a process for defining properties and attributes of services. Every definition of a service requires a unique service type, along with attributes that describe and constrain the service. This definition, referred to as a service scheme, must be reviewed, standardized, and archived with IANA. Once a service is defined and registered, the protocol also provides a mechanism for advertisement of the service. SLP specifies a dynamic mechanism for adding (and removing) services – even new services the authors could not have imagined.

These service schemes are independent of host addresses and domain names. It is the protocol which binds an instance of the service with details of how to connect with that service. Since the protocol relies on service agents being proactive and registering themselves with directory agents, a service provider may be dynamically assigned a network address. The service provider is not required to use a well-known port, registered with InterNIC, since the clients of this service will

[1] SLP also specifies the use of broadcast whenever multicast capability is unavailable in the local environment.

access it through knowledge of its unique service type. This is essentially the primary benefit of the protocol – the separation of application services from network location details.

All service requests include a predicate that is specified in terms of the service scheme attributes. The asterisk (*) character may be used within the predicate to indicate substring matching on an attribute.

### *Extended Implementation*

The extended implementation of SLP includes a Directory Agent (DA), which acts as a centralized repository of service information – a kind of service switchboard. Service agents actively seek all directory agents using DA discovery, multicasting requests for the directory agent service type. A service registration is unicasted to each DA discovered. DA's actively advertise their service by multicasting advertisements. A user agent attempts to discover a DA initially. If successful, the UA unicasts service requests directly to the DA.

The use of multicasting allows a UA to discover any DA within its time to live range. This mechanism provides a limited fault tolerance capability, since a client's UA effectively contacts every potential DA. In some configurations, however, a UA may only have the capability to contact a single DA. If no DA is present to receive a service request, then the UA multicast its service request again, with the service type of the request set to the ultimate target service -- for example, 'internet telephony gateway' -- instead of 'directory service'. Any SA within range responds to the request directly. Thus, SLP handles temporary DA unavailability.

### *Service Registration Lifetime*

SLP is a *soft* state protocol, since service registrations may expire, or time out, reducing the possibility that clients may acquire stale information. Each service registration has a *lifetime* attribute associated with it, which determines how long an SA or DA maintains a service registration , prior to expiring and removing the registration. It is the responsibility of each SA to register periodically with DA's within the SA's range, thus renewing the service providers registration. In addition, the SA should remove its registration, prior to a known service shutdown. Failure to renew the registration is assumed by the DA as an indication the SA, or service provider, is no longer capable of providing the service. An expired service registration is removed from the DA's cache.

## PROGRAM DOCUMENTATION

### *Why Java?*

The Java™[1] language is used to implement the prototype. Specifically, JDK 1.1.6 is the development version, installed on Sun workstations and servers running SunOS 2.6. This Java application runs on any platform that supports the Java interpreter and runtime environment for JDK 1.1.5 or later. This version is necessary since it supports multicasting. At the time of this writing most platforms, with the exception of the Macintosh, support this version of JDK. The

[1] Java is a registered trademark of Sun Microsystems.

Java runtime implementations currently available link tightly with platform-native communications libraries that do most of the CPU and memory-intensive communications work.

Using the object-oriented idioms of the language, each agent is abstracted as a separate class, encapsulating its own specific features. Common functionality is abstracted into utility classes that are used by all agents. All classes defined in this prototype are organized in the SLP package. Test programs are not defined in the package.

All strings within SLP messages are encoded using UTF-8, a transformation format, used to more efficiently represent Unicode™[1] characters on the network, since the majority of character sets can be encoded using 1 byte. Java supports simple UTF-8 operations directly. The *DataInputStream* class supports the operation readUTF(), while the *DataOutputStream* class supports writeUTF().

Threads are used extensively to allow concurrent operations, as well as implement timers. The setSoTimeout(int) method on *DatagramSocket* is used to specify the time in milliseconds to wait on a blocking socket operation. This mechanism is used in implementing SA/DA discovery, as well as waiting for service replies.

The *DatagramSocket* class is used to transmit all SLP messages and gain the best network performance. Since all SLP messages used in this prototype are small enough to fit into the maximum transmission unit of UDP, the *DatagramPacket* class is used to implement all SLP messages transmitted among agents. TCP connections are not supported by this prototype; however, commercial implementations should support TCP connectivity, for those cases where SLP requests or replies cannot fit into the standard UDP packet.

As mentioned above, UDP multicasting is used for self-discovery of directory agents and/or service agents. The *MulticastSocket*, a subclass of *DatagramSocket* which is based on UDP multicasting, is used for service discovery requests. The joinGroup(inetAddress) and leaveGroup(InetAddress) methods of *MulticastSocket* are used to join and leave the SLP multicast address. Its setTTL() method is also used to specify the number of network hops a *DatagramPacket* can traverse before it must die.

### *Installation*

The code for this prototype may be downloaded from:
http://home.worldnet.att.net/~carriecaldwell/SLP.tar.gz
After downloading, just do the following, assuming you are working in a Unix environment:

```
$ gunzip SLP.tar.gz
$ tar xvf SLP.tar
```

To run the prototype requires that you include the directory <install_directory>/SLP in your CLASSPASTH environment variable.

[1] Unicode is a trademark of Unicode Consortium, http://www.unicode.org.

### *Application Design*

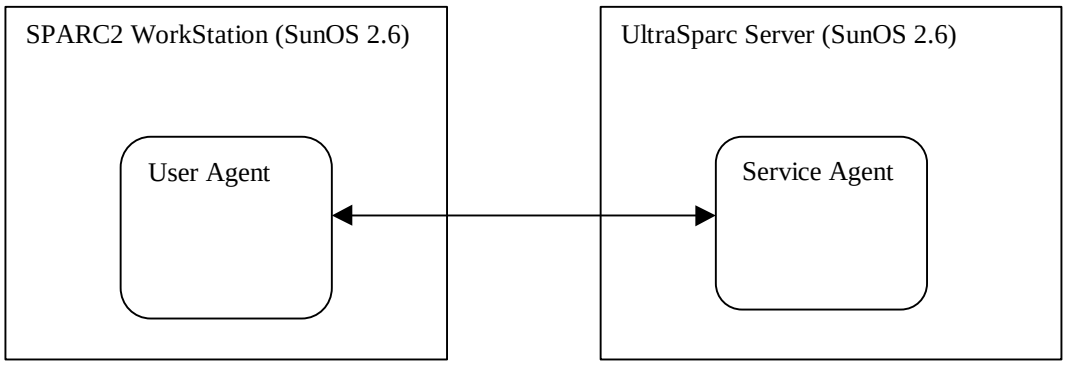
Client protocol is modeled at the lowest level as a DatagramClient.[1] This class uses two other application classes, DatagramClientReplySlave and DatagramClientSendSlave, which handle the blocking operations of sending and receiving UDP datagrams, by extending Java's Thread class and defining **run** and **wait** methods. The DatagramClientReplySlave sets up a separate thread that starts waiting for a datagram to arrive, and it times out if a datagram is not received within the specified waiting time. DatagramClient sets up and controls DatagramClientReplaySlave. The DatagramClientSendSlave sets up a separate thread that sends a datagram, waits a fixed period of time, then resends the datagram, continuing this loop until notified to stop by DatagramClient. The DatagramClient creates a DatagramClientReplySlave, which starts listening for replies. Then, the DatagramClient creates a DatagramClientSendSlave, which begins sending the datagram. DatagramClient invokes a blocking method, waitOnReply(), waiting for DatagramClientReplySlave to return. Finally, when waitOnReply() returns, DatagramClient modifies the state of DatagramClientSendSlave, causing it to terminate sending out datagrams.

The UserAgent class extends DatagramClient, which provides the capabilities for sending, receiving and retrying requests to ServiceAgents (or Directory Agent). Additional methods in this class build requests and handle responses from SA's that reply to a service for a given service type.

Server protocol is modeled as a DatagramServer, responsible for listening on the SLP multicast address for SLP requests. The ServiceAgent class extends DatagramServer, setting up a separate thread that calls DatagramServer's waitForRequest() method. Each datagram received is handled in a separate thread by handleRequest(). The ServiceAgent also defines a method for registering services with a Directory Agent .

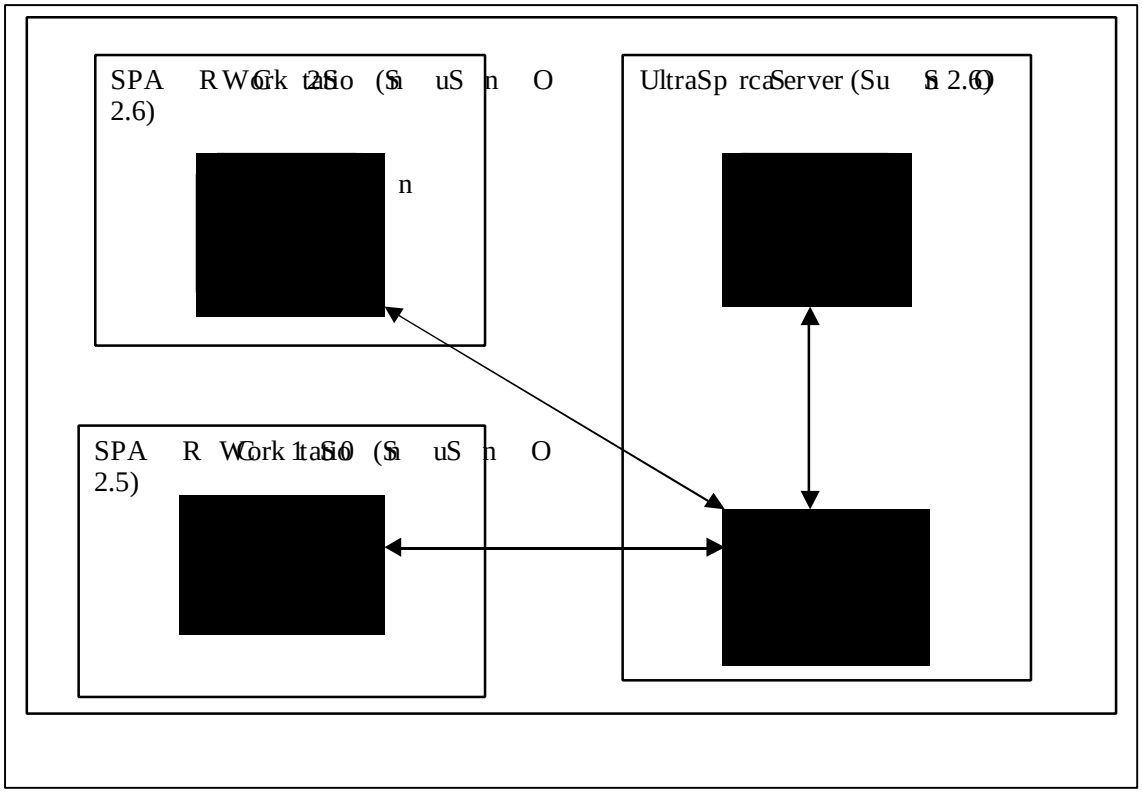
### *Process Architecture*

The following diagram illustrates process and host boundaries of the prototype. Separate processes are defined for each agent, which uses its appropriate agent class. Each agent process may be replicated on the host, although there may be only one instance of a DA on a given host.




[1] The design pattern of this application class is based on a class from Java Networking and Communications.

The next diagram illustrates a more centralized configuration with a Directory Agent process used for service type caching. The arrows connecting processes in these diagrams are intended to show logical message flow, not implying any long-term connection. However, the protocol does permit the use of TCP whenever message size exceeds the maximum UDP packet size. In these cases, it is the responsibility of the connecting client to tear down the connection upon completion of the transaction. Otherwise, the service agent (or directory agent) must close the connection after CONFIG_CLOSE_CONN seconds have elapsed.



### *Protocol Parameters*

The following table defines the tuneable parameters implemented in this prototype as Java properties. These properties are initially defined and defaulted to the values in the *Default Value* column. Individual properties, however, may be set with the –D option to the Java interpreter, which inserts these properties into the system properties list. When a class that uses one of these parameters is instantiated, it initially looks in the system properties list to check for a new runtime value for the parameter; otherwise, the default value is used.

| Parameter Name | Default Value | Values Tested | Meaning |
|---|---|---|---|

| | | | |
|---|---|---|---|
| CONFIG_MC_RETRY | Each second | About 1 second with backoff. | Multicast query retry interval. |
| CONFIG_MC_MAX | 15 seconds | 15 seconds | Max time to wait for multicast query response. |
| CONFIG_START_WAIT | 3 seconds | 3 seconds | Wait time prior to multicast query for directory agent after reboot. |

## CONCLUSION

The prototype has shown that Service Location Protocol is feasible for dynamic discovery of services in the local area. Implementation of this protocol allows administrators to more flexibly assign services to host in their networks without concern for disruption of configuration information in client hosts, while providing a limited degree of fault tolerance. Client host can move into new networks without the need to reconfigure network address information for commonly used services.

### *Recommended Protocol Enhancement*

The SLP specification may be amended to prevent fake service providers from spoofing a DA. The DA verifies the service by contacting the service provider with a verification message, and the service provider responds with a confirmation message.

## FUTURE WORK

Extending SLP to the wider area internet community is proposed in Wide Area Network Service Location, through the use of service advertisement across wide area multicast groups. In order to reduce the protocol's consumption of network resources, WANSL proposes limiting the rate of advertisement of services across the wide area, proportional to the number of advertising agents in the network. Future work should focus on proving the scaleability of SLP to the wide area internet community, while determining its impact to network resources.

Adding dynamic service discovery to browsers and displaying service attributes to the user is a natural extension, and this capability should be prototyped.

## RELATED WORK

### *RDU*

The Resource Discovery Unit researches and develops tools and processes that enable *resource discovery*, a broad field of study which includes service location. Recent projects include Yarra, which addresses the construction, maintenance and distribution of metadata that enables user location by querying directories of information. This project focuses on definition of the metalanguage and its creation by the publishers of information.

### *LDAP*

Local Directory Access Protocol is an enhanced version of the X.500 standard.

REFERENCES


[1] Erik Guttman, Charles Perkins, John Veizades, Michael Day, Service Location Protocol V2., Internet Draft, March 6, 1998.

[2] Erik Guttman, Charles Perkins, James Kempf, Service Templates and service: Schemes, Internet Draft, March 12, 1998.

[3] John Veizades, Erik Guttman, Charles Perkins, S. Kaplan, Service Location Protocol, RFC 2165, June 1997.

[4] David Flanagan, *Java in a Nutshell, 2nd Edition*, O'REILLY, 1997.

[5] Todd Courtois, *Java Networking & Communications*, Prentice Hall, 1998.