An Application for Discovery, Configuration, and Installation of SLP Services

-

Evan Hughes, David McCormack,
Michel Barbeau, Francis Bordeleau[1]
hughes@nexus.carleton.ca, dwmccorm@nexus.carleton.ca,
barbeau@scs.carlteon.ca, francis@scs.carleton.ca

Abstract

-

The Service Location Protocol version 2 (SLPv2) provides a framework for service discovery on IP-based networks. It provides a means for services to advertise themselves to potential clients, and for clients to select an appropriate service. Since its publication, SLPv2 has been gaining acceptance.

This paper provides a general background to SLPv2, and an overview of our experiences creating the first user-level service browser for SLPv2 aware services, the SLPv2 Service Browser (SSB). In the paper we describe our experiences using SLPv2 and the standard SLPv2 API, both to create a service client (the SSB) and to modifying existing services: timed, Squid HTTP-proxy, and the CUPS printer daemon.

1. Introduction

Currently one of the most burdensome tasks in computer maintenance is location and configuration of networked services such as printers or mail servers. As the number of network services increase, so does the need for a network service discovery mechanism. A scalable, light-weight, IP based, service discovery method has not been ready for mainstream deployment until now.

Service Location Protocol (SLPv2) has been designed for the discovery of network services on an IP network [3] . This protocol has the potential for becoming the standard for service discovery within IP networks. Presently, there exist two roadblocks which impede the widespread acceptance of SLPv2, the largest being the ease of use by end users. The other is the effort and skills required to make a service SLPv2 aware. These issues have yet to be addressed currently in SLPv2 and are explored in this paper.

The issues of end user usability and complexity of modifying services to be SLPv2 aware were resolved through the browser and API extensions. Initial research into existing end user SLPv2 usability resulted in finding that no generic tools exist for service discovery and installation. Upon this result we created an application for the discovery, configuration, and installation of SLPv2 services. Secondly, the development issues were analyzed by modifying several common service applications to be SLPv2 aware in order to discover how SLPv2 could be improved from a programmer's perspective. This research resulted in the creation of an API to handle service attributes. These two contributions enhance the usability and simplicity of service maintenance, hence reducing the road blocks to widespread usage of SLPv2.

This paper begins with the discussion of a service discovery example and an overview of SLPv2 in Section Two. Our contributions: the SLPv2 Service Browser and SLPv2 API extensions are contained in Section Three. Section Four concludes the paper with a list of further research topics.

2. Service Location Protocol

This section provides an example usage of SLPv2 and overview of protocol: explaining SLPv2's basic functionality, optimizations made to SLPv2 to lessen its resource usage, and finally the current areas of research.

2.1 Typical Service Discovery Example

Consider a typical service discovery scenario. It is five minutes before a meeting, and the sales team is preparing for their presentation. They try to connect to the slide projector. The bulb burns out. The sales team must print out their slides onto transparencies on their clients' network.

This scenario typically results in contacting the network support group, who then determine the appropriate printer and software driver for their computer. Once the printer installation and printing is completed, the meeting ends up starting late.

Now consider the scenario in an environment with SLPv2 deployed in a wireless network. Rather than search for the administrator, search for printing services . A printing service request is formulated in search of a printer capable of printing on colour transparencies. This service request is created and sent out over the wireless network. Seconds later, services begin to appear in the browser, each one satisfying the service request. One of the discovered printers is selected, and the print job is subsequently sent to that printer. As the job is printing, the user queries the returned information to determine the room number of the printer. The sales staff walks over to the printer, grabs their slides, and starts the meeting on time.

This example shows the benefits to users of a SLPv2 aware service. These benefits originate from the reduced complexity of service discovery and configuration, which is what the protocol was designed to achieve.

2.2 Basic Operation

In its most basic form, a SLPv2 network contains two types of entities: user agents (UAs), which represent client applications that wish to find services; and service agents (SAs), which represent server applications that wish to advertise and provide a service.

To find a service, a UA composes a service request (SrvRqst) that describes the desired service which it sends via multicast to the rest of the hosts on the network. SAs that represent services compliant with the SrvRqst send back unicast responses (SrvRply) that identify the location of each service as a Uniform Resource Indicator (URI). This interaction is shown in Figure 1. Note that although the UA multicasts the SrvRqst to all SAs: only SAs with a service compliant to the SrvRqst are shown.
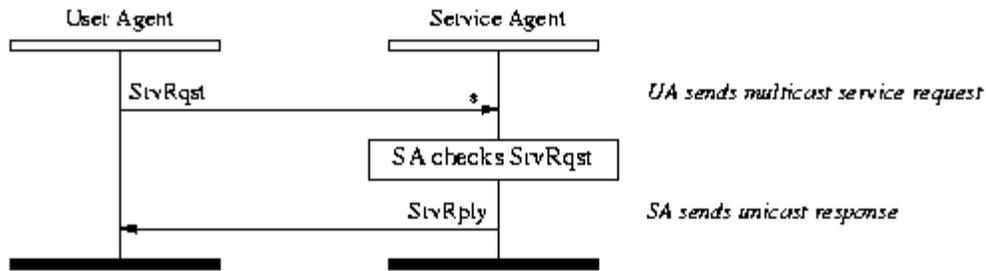
Figure 1: Interaction diagram showing messages passed between User Agent and Service Agent during service discovery.

Since every SrvRqst compliant SA returns a SrvRply, the UA may receive many service URIs. In such a situation the UA may wish to find the most appropriate URI to use (i.e., the server having the lowest load, fastest processor or smallest waiting queue). Although SLPv2 does not provide a framework for comparing services, it does allow the UA to request (on behalf of its client application) the attributes of individual services. It is the client applications' responsibility to compare the attributes.

2.3 Reduction of Bandwidth Usage

In situations where there is a large number of SAs and UAs on a network, it is conceivable that the network could become flooded with SrvRqst and SrvRply messages. SLPv2 employs two mechanisms to lower bandwidth usage: administrative scopes and directory agents (DAs).

Administrative scopes logically partition the services available on a network. A scope is defined as a text string held by UA and SA. When a UA sends a SrvRqst, it includes its scopes as part of the request. Only SAs having scopes in common with those in the SrvRqst reply. Scoping lowers bandwidth usage by lessening the number of SAs that respond to a SrvRqst.

DAs act as proxies to multiple SAs. Instead of SAs listening for SrvRqst, SAs register their services with one or more DAs. The DA then takes on the SAs role of listening for and evaluating SrvRqsts and replying with SrvRplys. Instead of UAs multicasting their requests, they unicast their SrvRqst to one or more known DAs. Each receiving DA searches through its set of registered services and sends a unicast reply. Figure 2 describes the interaction between UA, DA and SA.
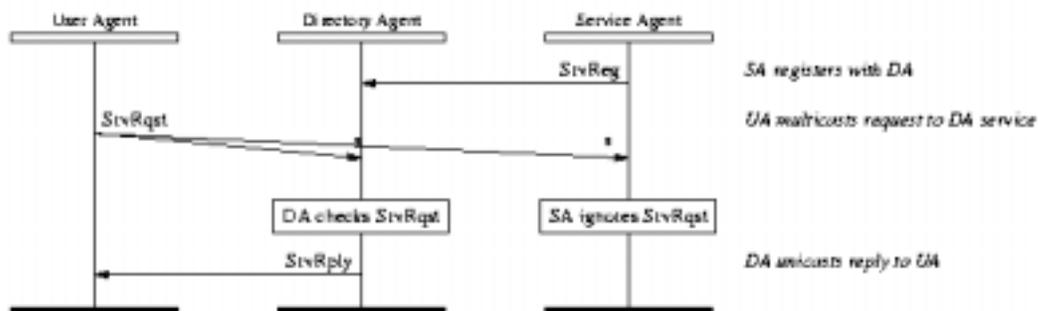

Figure 2: Interaction diagram of a UA, DA and SA exchanging messages.

Both methods of bandwidth conservation require SLPv2 aware hosts to have some extra information: for scoping, each SA and each UA must know which scopes they belong to; for DA proxying, each SA and UA must know the URI of the DAs on the network. SLPv2 allows UAs and SAs to get scoping and DA information from multiple sources: initialize-time configuration (by mechanisms such as DHCP), static configuration (by the system administrator or user), or discovery (using mechanisms within SLPv2).

2.4 The Evolution of SLPv2

Although SLPv2 is a stable standardized protocol, a number of extensions and enhancements are currently being considered. These include mesh enhancement for DAs, and service notification and description. Mesh enhancement for DAs is described in draft by Zhao and Guttman[13]. Currently, DAs do not interact with one another. Each SA is responsible for registering its services with each DA. Mesh enhancement would provide a standard way for DAs to share known services between one another, meaning that every DA would store the same information. Service discovery in SLPv2 is currently request driven: when a UA wants to find a service of a certain type or with certain characteristics, it must poll the network with SrvRqsts. Service notification and subscription [6] would allow a UA to lodge a notification request with one or more DAs. The DA would then inform the UA whenever a service compliant with the notification request appears.

2.5 Competing Technologies

In addition to SLPv2, there exist a number of emerging communication technologies that offer some form of service discovery as part of their built-in infrastructure. These include: Apple Talk [8], Bluetooth [1], CORBA [9], Jini [4], Universal Plug and Play [12], and Server Message Block [7,12].

It should be noted that these protocols are designed for communication between (spatially or logically) diverse components.

The Service Location Protocol is complemented by a standard API which can be used to control it. The API is described in [5]: it provides an interface to SLPv2 that greatly simplifies the implementation of both UAs and SAs. The interface allows a client application to query the network for SLPv2 aware SAs, query a service for its attributes, and make a service available on the network.

When the API is used to advertise a service, it automatically detects known DAs. If there are one or more DAs, the API will register the service with those DAs. If there are no DAs, the API will act as an SA, responding to SrvRqsts and attribute requests. This process occurs transparently to the application.

3. SLPv2Contributions

There are three contributions to the resolution of the issues addressed in the introduction. The first, the extension of the SLPv2 API to improve attribute serialization, is followed by details on the SLPv2 Service browser and it creation. This section concludes with a discussion on the modifications performed to make an implementation of an Internet Printing Protocol SLPv2 aware.

3.1 SLPv2API Extension

In our experience, the SLPv2 API was sufficient for almost all SLPv2 programming. The weakness we found, was a lack of functionality for creating and examining the attributes associated with a service. In response, we have created an SLPv2 attribute API, which we plan to release as an Internet Draft.

The SLPv2 attribute API is necessary because of SLPv2's well developed attribute model. In SLPv2 there are five types of attributes: integer, string, keyword, opaque and boolean. Each type has a limited range of values and, in the case of opaque, keyword and boolean, special considerations for serialization. Serialization is a concern since (in standard SLPv2) it is the application's responsibility to build and serialize the attribute list. We extended the existing SLPv2 API with functionality for attribute manipulation and moved all serialization concerns into the API.

With the API extension we are presenting, we add two data structures and a number of functions. The data structures represent an attribute list and an iterator that enable client applications to easily loop over the members of the attribute list. The added functions allow applications to manipulate members of the attribute list.

In our experience, the API extension greatly simplifies the task of making services SLPv2 aware. It abstracts away messy string manipulation, and allows the programmer to focus on the core functionality of the application software.

3.2 SLPv2 Service Browser

It is occasionally desirable to be able to view the set of all services available on a network. The SLPv2 Service Browser (SSB) allows a user to browse the SLPv2 aware network services in a independent manner.

The SSB is a user interface built on top of the SLPv2 API [5]. It uses the standardized API for searching and querying functionality. In addition, it uses the attribute API to manage received attributes. It has the ability to run service-specific commands (such as installing a printer, connecting to a news server, etc.).

3.2.1 Browser Functionality

The SSB provides the user with three views of the services available on the network. The first view displays all services in the current administrative scope, Figure 3 shows the SSB after it has found a number of proxies, a timed server, and two printers. The second view allows the user to query for services with specific attributes, scopes, and/or service types. Figure 4 shows the results of a query for service type "proxy", in the "default" scope, with the predicate "(load=0)". The third view displays the attributes of available services. Figure 5 shows the SSB displaying the attributes of an SLPv2 aware CUPS server.

Each view provided by the browser aids the user in system management. The second is of particular interest because it allows the user to execute commands on a service. Each command takes the form of a shell script, where service attributes are inserted. Using this facility the user can query the network for a service, locally select one of the found services, and invoke a command on it.
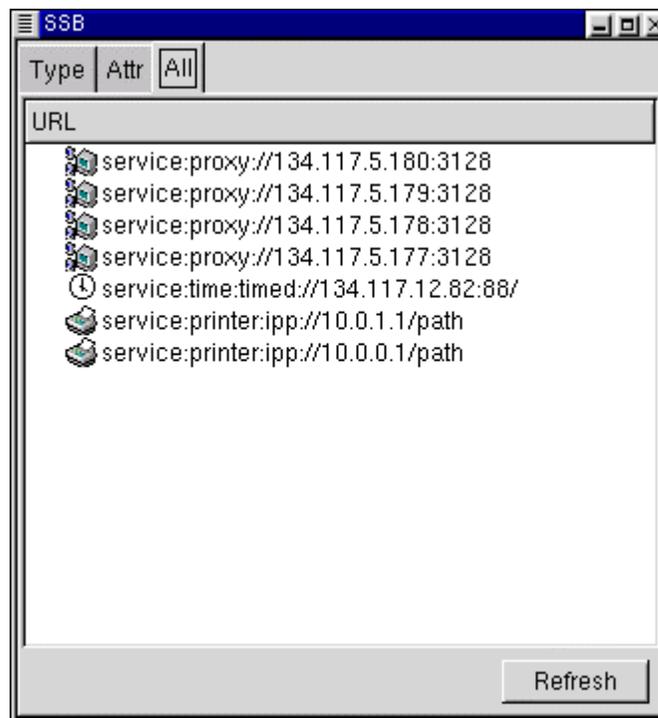
Figure 3: SSB Service view.
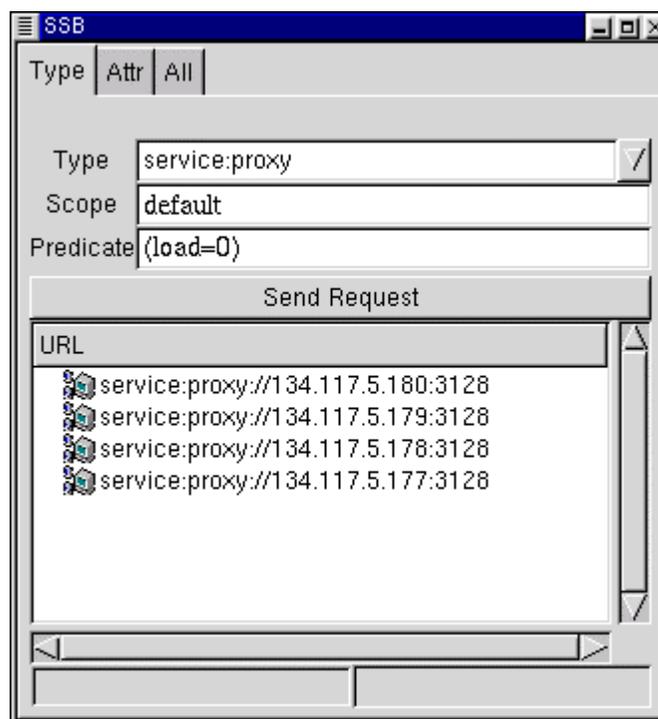
Figure 4: SSB query view.
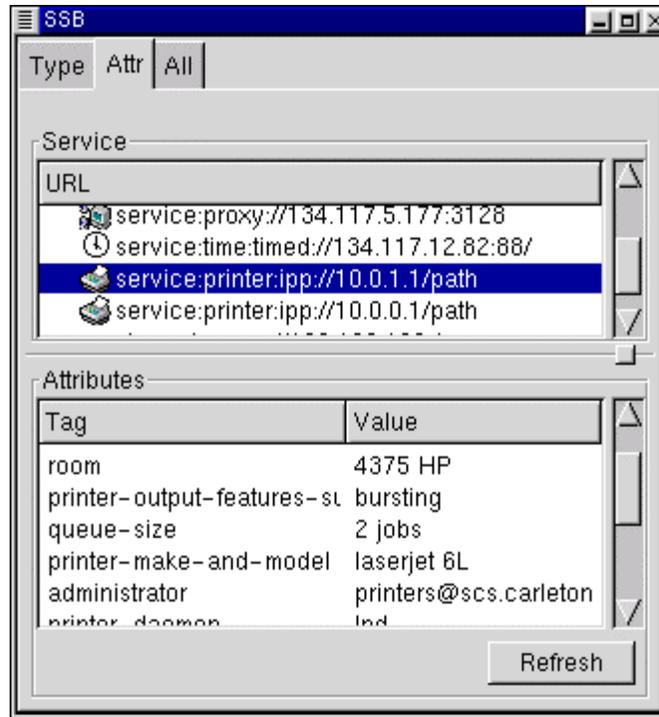
Figure 5: SSB attribute view.

3.3 OpenSLP Contributions

The selection of an SLPv2 implementation lead to the open source implementation of OpenSLP[10]. Preceding development, we explored OpenSLP's current state through the creation of a test suite for the SLPv2 API as defined by Kempf and Guttman [5]. The test suite we created provided us with experience in all the completed functionality of the API.

OpenSLP is not functionally complete because it is a work in progress. We completed two internal functions dealing with attribute serialization which were necessary for the work which we were to perform. Once the additions were completed, they were submitted into the project. All of those modifications are now part of the OpenSLP package.

3.4 SLPv2 Support for the Internet Printing Protocol

The Internet Printing Protocol (IPP) was recently introduced to allow for printing of documents through the Internet. This protocol, introduced in early 1999, will most likely change the face of printing, just as PostScript did. There are many commercial supporters of IPP, but the only open implementation of IPP is the Common Unix Printing Service (CUPS) [2]. As part of our effort, we added SLPv2 support to CUPS because of the high recognition of network printing issues. This support allows users to realize the impact and benefits of SLPv2.

The current implementation of CUPS deploys a printer discovery scheme which is not based on a standard. The addition of SLPv2 support to CUPS allows for printer discovery using a recognized protocol. This additional functionality coexists with the previously deployed service discovery methods, allowing for backward compatibility.

The extensions we made to CUPS were performed both with our API extensions and without. It was found that when using the API extensions, the code was much simpler and shorter because of added functionality provided for attribute creation and manipulation.

4. Conclusions

In our initial analysis of SLPv2, we concluded that there are two potential impediments to SLPv2 gaining general acceptance: end-user usability and programming difficulty.

The end-user usability of SLPv2 depend upon the applications that use it. In the short term, applications can be made SLPv2 aware using general-purpose browsers such as the SSB. The SSB acts as a UA and provides powerful scripting, allowing the user to easily modify the configuration of non-SLPv2 aware service clients. In the long term, it would make sense for clients of the networked services to use the SLPv2 API to directly query the network for services, a process the user should be strongly insulated from.

In a somewhat unscientific test of programming difficulty, we made a number of existing applications SLPv2 aware. Before implementing the attribute API extension, we found the modifications were simple but laborious: making a service SLPv2 aware was easy, but serializing the attributes for service registration was difficult. With the use of the attribute API, we found the process of service modification easy -- the programmer needs only to pull the attribute values out of the application. Programming difficulty is now solely a function of the complexity of the application.

There are many further works that have been discovered which we think are important and that would form a basis for subsequent work. These include survey into service discovery mechanisms, attribute API extension advocation and SSB evolution.

A survey of existing service discovery mechanisms comparing and contrasting Jini, Corba, and SLPv2 with respect to bandwidth, ease of use, functionality, and deployment.

The SSB could be improved through the addition of more functionality, such as attribute filtering and sorting, and cosmetic improvements to the user interface.

The attribute API extensions are currently deployed only in the projects mentioned in this paper. In order for the SLP community to benefit from these enhancements, advocation of this attribute API extension must take place so that all may gain from this contribution.

5.References

1."bluetooth.org", August 2000, http://www.bluetooth.org

2."Common Unix Printing System", August 2000, http://www.cups.org

3.GUTTMAN, E., PERKINS, C., VEIZADES, J., Service Location Protocol, Version 2, Internet Standard RFC 2608.

4."Jini.org -- The Community Resources for the Jini Technology", August 2000, http://www.jini.org

5.KEMPF, J., GUTTMAN, E., An API for Service Location, Internet Standard RFC 2614.

6.KEMPF, J., Notification and Subscription for SLP, Internet Draft draft-kempf-srvloc-notify-03.txt.

7. "Microsoft", August 2000, http://www.microsoft.com

8. "Networking", August 2000,
   http://developer.apple.com/techpubs/mac/Networking/Networking-2.html

9. "OMG CORBA Web Site", August 2000, http://www.corba.org

10. "OpenSLP", August 2000, http://www.openslp.org

11. "SAMBA - Opening Windows to a Wider World", April 2000, http://www.samba.org.

12. "Universal Plug and Play Forum", August 2000, http://www.upnp.org

13. ZHAO, W., GUTTMAN, E., mSLP - Mesh Enhanced Service Location Protocol,
    Internet Draft draft-zhao-slp-da-interaction-07.txt.