A Service Browser for the Service Location Protocol Version 2 (SLPv2)

Eivind Jåsund, Christian Bettstetter, and Christian Schwingenschlögl

Technische Universität München (TUM)
Institute of Communication Networks
D–80290 Munich, Germany[*]

Abstract

Service discovery protocols automatically find services and their attributes in communication networks. These features are especially helpful for mobile users in foreign networks and groups of users that form spontaneous (ad hoc) wireless networks. This paper presents a service browser for the IETF Service Location Protocol (SLPv2). We describe the main features of the browser and the requirements for functionality and user interface. Furthermore, we briefly comment on the used software development tools and libraries.

Keywords: *Service discovery protocols, service location protocol (SLPv2), service browser.*

1 Introduction

Service discovery protocols enable users of a communication network to find services, applications, and devices that are available in the network. In particular, they allow to search and browse for a service that fulfills a certain task. This feature is especially useful for mobile users (with notebooks, network–enabled PDAs, or mobile phones) in a foreign network and for groups of users that form a spontaneous (ad hoc) wireless network.

Let us give an example: Suppose you are on a business trip, sitting at the airport with your notebook and waiting for the connecting flight. During the previous flight, you have written a technical report. Now a printout must be made, and another copy should be faxed to your company. The airport offers access to a local area network with printer and fax devices. Without service discovery, a considerable configuration effort is necessary to fulfill your task: Your computer must be configured with the names and addresses of the printer and the fax in order to communicate with them. Moreover, appropriate device drivers must be installed. This is a standard procedure for accessing services in networks today.

The main goal of service discovery protocols is to ease this configuration process. Using service discovery mechanisms, services automatically advertise themselves, supplying details about their features and information required to access them. Users or applications can locate a service by asking for a particular service type (e.g. printer) and may make a service selection in case multiple services of the desired type are available. From the network administrator's point of view, these protocols ease the task of setting up and maintaining the network. Especially the introduction of new services and devices is simplified.

The most well-known service discovery protocols currently under development are the Service Location Protocol (SLP) [1][2], Jini [3], Universal Plug and Play (UPnP) [4], and Bluetooth's SDP [5].

In [6] we compared these service discovery protocols and presented our beta implementation of SLPv2. Our protocol implementation demonstrated the fundamental functionality of SLP in practice. The essential features were implemented in C and allowed basic transactions between the three agents via UDP sockets. During the last year, we improved and extended the functionality of the protocol implementation. The current release (TUM–SLP v0.2) includes security mechanisms and service lifetime expiration (leasing) [7]. It is one of the first implementations of SLPv2 that supports authentication mechanisms. The source code is available at *http://www.lkn.ei.tum.de/~mcg/slp/*. We also successfully tested interworking with another SLPv2 implementation.

In this paper we discuss the usability of SLP and present our 'service browser' for TUM–SLP. Via a graphical user interface, our browser offers the most important protocol functions in a straightforward and user–friendly way. Users can

[*]The authors can be contacted via email at Eivind@web.de, Christian.Bettstetter@ei.tum.de, and Christian.Schwingenschloegl@ei.tum.de. Web: http://www.lkn.ei.tum.de

perform queries for particular services and their attributes or obtain a list of all services of the network. The found services are displayed in a tree, and users can browse through the attributes.

The structure of this paper is as follows: The basic functionality of the SLPv2 protocol is reviewed in Section 2. Section 3 describes the basic software design, the requirements for functions and user interfaces, and the used development tools and libraries. Section 4 presents the main features of our browser, in particular the different methods for searching a service. Section 5 briefly describes other service browsers for SLP, before Section 6 concludes this paper and gives an outlook on further work.

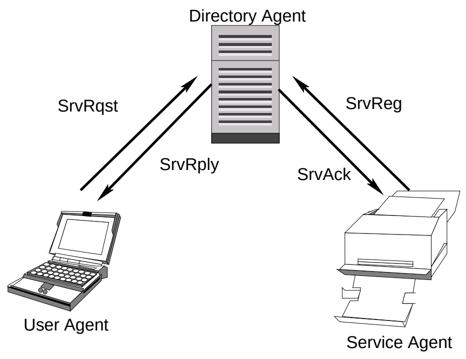


Figure 1: SLP architecture

2 Service Discovery with SLP

The Service Location Protocol version 2 (SLPv2) [8, 1, 2] is a proposed standard of the IETF (Internet Engineering Task Force), which has been designed for service discovery in IP–based networks. Its system architecture consists of three components:

- User Agents (UA) are searching for a service on behalf of the user or application.
- Service Agents (SA) advertise service information (e.g., location and characteristics).
- Directory Agents (DA) store information received from SAs and respond to service requests from UAs.

Figure 1 illustrates the basic functionality of the protocol if there exists a DA[1]. When a new service connects to the network, the SA advertises its existence by registering the service to the DA. Using a *SrvReg* message, it informs the DA about its URL (Uniform Resource Locator) and a set of attributes describing the service. If a UA (representing a user or application) wants to find a certain service, it sends a service request message (*SrvRqst*) to the DA. Optionally, he or she can specify some service type and attributes that the service should have (e.g. color printer). If the DA finds a service in its database that matches the request, it returns a unicast reply message (*SrvRply*) back to the UA. This message includes the URLs of the desired services.

Discovery of DA Since the address of the DA is not available to any other agent, the SA and UA must discover its existence. They multicast a request specifying *service:directory-agent* as the desired service to the SLP multicast group address 239.255.255.253 on port 427 (both specified in [8]). All DAs in the network listen to this port and reply with a unicast *DAAdvert* message to the requesting agent. In addition to this so–called active DA discovery, DAs can also advertise themselves by sending unsolicited *DAAdverts* to announce their presence. This discovery method is called passive DA discovery. SAs and UAs can extract the DA's address from the *DAAdvert*, as they do when performing active DA discovery. The third method to obtain the address of a DA is through static discovery via DHCP (Dynamic Host Configuration Protocol [9]).

Scopes Like most of the other service discovery protocols, SLP is administratively scoped. This means that the protocol locates resources (devices and services) available in a network within an administratively defined network domain. These domains are logical and do not need to correspond to the physical environment. Making use of scopes can result in lowering bandwidth usage on networks, because usually the number of participants in each scope is less than in the entire network. Users belonging to a certain scope may only discover services that are offered in this scope; other services are hidden by the DA.

Service Description: Service Type, URL, and Attributes Services are classified by their service type (e.g., printer) and can be located via a service URL. The URL contains the IP address of the service, a port number, and a path. The attributes associated with services are defined in so–called service templates [10]. For a particular service type, these templates specify possible attributes (e.g., color–supported, printer–resolution–supported) and their default values.

Service Selection Searching for a particular service in a network can often lead to a long list of services that is returned to the user. For example, if a UA does not query for a specific service type in a *SrvRqst*, the DA will return a list of *all* network services within the respective scope. Also if a

[1]SLP also allows operation without DA, but this operation mode is not considered in this paper.

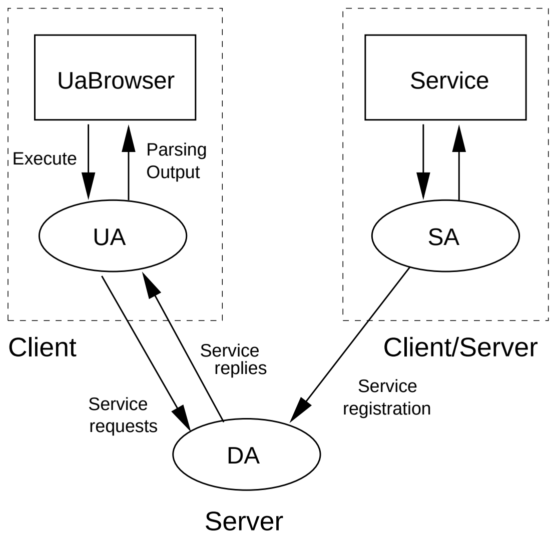


Figure 2: Our SLP solution with browser capabilities

UA restricts its request to a specific service type, the DA will in many cases find more than one service of this type in its database. The user must then decide which service is the best choice for a particular purpose. The SLP protocol does not specify how to make such a decision (it was not intended to do so either). One solution is to provide an application to the user that makes it possible to browse through all found services and compare their attributes. This has been the major motivation for us to develop a user–oriented service browser.

3 Design and Development of an SLP Browser

This section describes design aspects and software tools which were used for developing a service browser based on our SLP implementation. We denote this browser as UaBrowser. Our current SLPv2 release (TUM-SLP v0.2) consists of one DA, one or more SAs, and one or more UAs. All agents can be run on the same or on different Linux computers. The presence of the DA is mandatory for our browser implementation.

The UaBrowser is designed to work on top of the UAs, separated from the rest of our protocol implementation. Furthermore, it is still possible to run TUM–SLP without UaBrowser, because users do not always have the resources needed for using the graphical features combined with the service application. Figure 2 illustrates this concept.

3.1 Functional and User Interface Requirements

In our design we make a distinction between functional and user interface requirements. By considering the functional aspects, the following requirements are used:

- Search by service requests, attribute requests, and service type requests
- Find all services within one scope
- Browse through all found services in the scope
- Browse through all secure services registered at the DA
- Get all attributes of a service
- Perform periodic service discovery (period set by user)
- Show messages sent between the SLP entities after a request

To provide convenience for the end user, there are also several requirements concerning the user interface. First, the program should be easy to learn and use, even if the user has no prior knowledge of service discovery. The interface should bring consistency (the same things should not be done in different ways) and give feedback to the user if some action goes wrong. Furthermore, it should be able to bring important SLP–related information, e.g., showing the transactions between the agents or service lifetimes.

3.2 Development Tools and Libraries

Since our TUM–SLP implementation runs on Linux, we had to find some appropriate tools based on the X–Windows system for developing the browser application.

We decided to use the Qt class libraries [11] on the free desktop KDE (K Desktop Environment) [12]. It was used by the KDE project group for the development of a window manager on X–Windows. The Qt tool kit consists of a C++ class library and tools for designing graphical user interfaces. It can be obtained from Trolltech (*http://www.trolltech.com*, *ftp://ftp.troll.no*) and is included in most Linux distributions. Some reasons for the choice of Qt and C++ are:

- Qt is programmed in C++. It uses different classes for each component. This gives the advantage that object oriented languages have, e.g., reuse and subclassing. The classes of Qt are well defined and structured.
- A detailed documentation is given, and example programs are available. The manual is written in HTML with hyperlinks to all member functions.

- Qt is available free of charge for the Unix/X–Windows platform. A Microsoft Windows version is also available, but it was not free of charge at the time of our project start. As the rest of our SLP implementation is based on the Linux environment, this was an adequate criteria for our purposes.

Qt provides a special signal and slot mechanism to connect widgets in order to create a graphical user interface. In our experience, this was a nice and efficient alternative to usual C–style callback functions. The libraries also include utility classes to handle strings and other data structures, interprocess communication, and socket programming, to name a few.

Although our current browser implementation runs under KDE, it should easily be portable to other platforms, since we only use Qt libraries and no KDE–specific libraries.

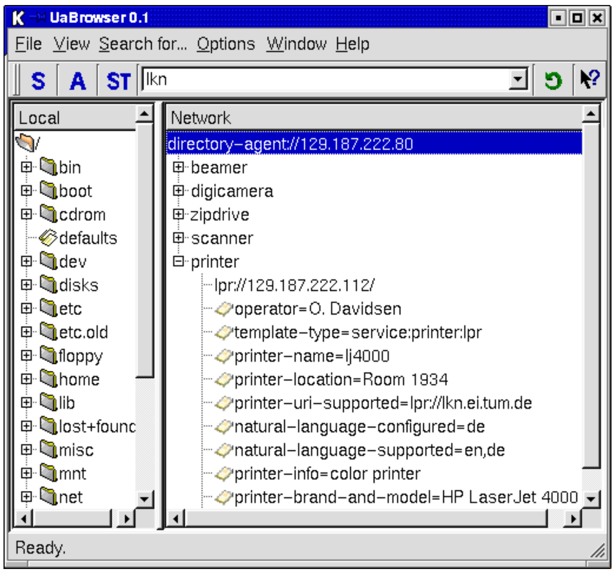
As an integrated development environment we use KDevelop [13]. It is designed for C/C++ application development and helps to create and maintain a project and to develop user interfaces in conjunction with the Qt Designer from Trolltech. KDevelop provides good access to the source code and the API documentation.

4 The UaBrowser

Upon startup, the UaBrowser automatically searches the network for an existing DA (active DA discovery) within the scope used in the previous session. As shown in Fig. 3, the user sees the main application window with its menus, a toolbar, a directory tree, a service tree, and the status bar. If a DA is found within this scope, it will appear with its IP address as root in the service tree (see Fig. 3). The user can now search for desired services in several ways and browse through the attributes of these services as described in the following.

4.1 Searching for Services by Scope

From a combo box (placed in the middle of the toolbar) or from a dialog (opened from the menu) the user can choose a scope among all scopes that are available for him/her at the DA. The UaBrowser then returns all services within this scope that are registered with the DA at this time. This process gives an overview of available services and provides a good basis for using the browsing facilities. As illustrated in Fig. 3, the results from the request are listed in the service tree below the DA entry.



```
/
⊞ bin
⊞ boot
⊞ cdrom
  defaults
⊞ dev
⊞ disks
⊞ etc
⊞ etc.old
⊞ floppy
⊞ home
⊞ lib
⊞ lost+founc
⊞ misc
⊞ mnt
⊞ net
```

```
directory-agent://129.187.222.80
⊞ beamer
⊞ digicamera
⊞ zipdrive
⊞ scanner
⊟ printer
    lpr://129.187.222.112/
      operator=O. Davidsen
      template-type=service:printer:lpr
      printer-name=lj4000
      printer-location=Room 1934
      printer-uri-supported=lpr://lkn.ei.tum.de
      natural-language-configured=de
      natural-language-supported=en,de
      printer-info=color printer
      printer-brand-and-model=HP LaserJet 4000
```

Figure 3: UaBrowser main window

4.2 Searching for Services by Service Type

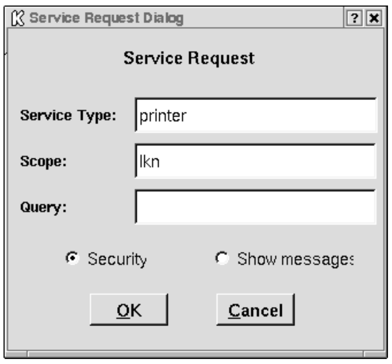
Searching for services only by scope is especially useful in small networks with few services. In larger networks, e.g., in large intranets with hundreds of services, this kind of search is not applicable anymore. Thus, in addition to a general service request, more specialized service requests are possible in SLP. In fact, service request messages *(SrvRqst)* can find services by type and scope, and also a search filter for attributes can be employed [8].

If the user wants to send a service request message to the DA, he or she opens the ‘service request dialog’ and fills in the desired service type and scope (Fig. 4). The user may also input an attribute string in the query field, to indicate that the returned service should contain this string in its attribute list. In Fig. 4 we search for all services of type ‘printer’ in scope ‘lkn.’

Services of the requested type are returned from the DA to the UA. In the example of Fig. 5 two printers have been found. The lifetime indicates for how long the service will be available in the network. The retrieved services are also shown in the service tree window.

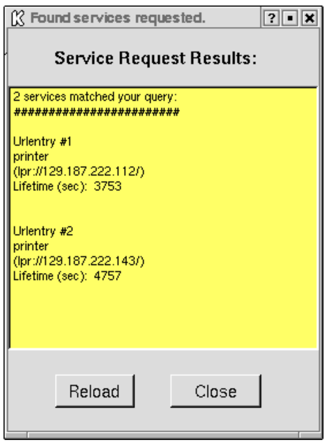
4.3 Obtaining Attributes of Services

As an optional feature, SLP allows UAs to find and query attributes of services. This is done by sending out an attribute request message (*AttrRqst*). Basically, two different request types are possible: first, a query of the attributes of a specific service (via the service URL) and, second, a query by service type. In the second case, the DA returns all attributes of all



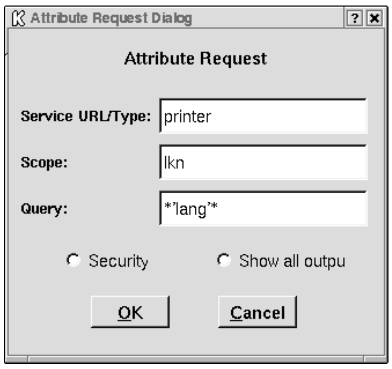
Service Type: printer
Scope: lkn
Query:
Security [✓] Show messages

Figure 4: Service request dialog



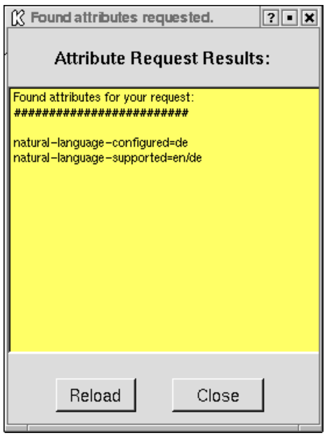
```
2 services matched your query:
########################

Urlentry #1
printer
(lpr://129.187.222.112/)
Lifetime (sec):  3753


Urlentry #2
printer
(lpr://129.187.222.143/)
Lifetime (sec):  4757
```

Figure 5: Search results from the service request



Service URL/Type: printer
Scope: lkn
Query: *'lang'*
Security Show all outpu

Figure 6: Attribute request dialog



```
Found attributes for your request:
########################

natural-language-configured=de
natural-language-supported=en/de
```

Figure 7: Search results from the attribute request

services of the requested type in the network.

To perform an attribute request with our browser, the user opens the ‘attribute request dialog’ (Fig. 6) and inputs the desired service URL or service type. As with the service request, the user must specify a scope.

In order to get the whole attribute list of each found service, the user can leave the attribute query field empty. When searching for a special attribute, the user can use wildcards, represented by an asterisk (’*’).

In Figure 6, an attribute search request is issued with ’printer’ as service type, ’lkn’ as scope, and ‘*lang*’ as query. The purpose of this query is to see if there is a device on the network with a service template containing language descriptions. Figure 7 shows the result: Two attributes have been found: ‘natural-language-configured = de’ and ’natural-language-supported = en/de’.

The attributes of services are also shown in the service tree in the main application window. By clicking on one of the services, the attributes of this service entry are shown in the next tree level (see Fig. 3, printer item).

Note that in our implementation, only one attribute can be specified in the attribute query field. This query is based on string comparison between the entered attribute in the dialog and the attribute list stored in the DA. The LDAPv3 (Lightweight Directory Access Protocol) string syntax [14] is not yet supported. This syntax definition allows more flexible queries, based on complex logical expressions, e.g.,

(printer-type = bw) & (paper-size = A4) &
(! (printer-resolution <= 600 dpi))

This statement searches for a black–and–white printer that is able to print A4–sized paper with a resolution of more than 600 dots per inches.

5 Related Work

We have found two service browsers for SLP, which have some similar facilities and functions. The first one is for commercial purpose.

Mac OS 9 Network Browser With Mac OS 9, Apple integrated the SLP protocol into its operating system [15]. Their SLP–supported network browser [16] is used to find servers, e.g., FTP servers and Web servers on the Internet.

Carleton University SSB Another service browser implementation, called SSB, is presented in [17][18]. SSB is based on the SLPv2 application programming interface (API) described in [19]. It uses the standardized interface for search and query functionality and employs an own API to manage received attributes. The SLP implementation from OpenSLP [20] was used for this browser.

6 Conclusion and Outlook

The service location protocol SLPv2 enables automatic service discovery in dynamic networking scenarios. In this paper we presented a service browser for SLPv2, which enables user–friendly access to our protocol implementation TUM–SLP. We described our software design, the functional and user interface requirements, the development tools, and finally explained the main searching and browsing features. Users can search for all services within a scope, search by type only, and query attributes of services. All results are displayed in a tree–style view and can be browsed in order to make a service selection.

Our work is one of the first implementations that demonstrates how a browser application can be deployed to give a better end–user usability for service discovery.

Enhancements to the service browser can be made, for example, by supporting service type requests (*SrvTypeRqst*) that find all service types in a scope. Further work concerning the the TUM–SLP protocol implementation is e.g. full LDAPv3 support.

References

[1] E. Guttman, “Service location protocol: Automatic discovery of IP network services,” *IEEE Internet Computing*, pp. 71–80, July 1999.

[2] J. Kempf and P. S. Pierre, *Service Location Protocol for enterprise networks*. Wiley, 1999.

[3] “The community resource for Jini technology.” http://www.jini.org, 2001.

[4] “Universal plug and play forum.” http://www.upnp.org, 2001.

[5] “Bluetooth specification (version 1.1) - core, part E: Service discovery protocol (SDP),” 2000.

[6] C. Bettstetter and C. Renner, “A comparison of service discovery protocols and implementation of the service location protocol,” in *Proceedings EUNICE 2000*, (Twente, Netherlands), Sept. 2000.

[7] M. Vettorello, C. Bettstetter, and C. Schwingenschlögl, “Some notes on security in the Service Location Protocol Version 2 (SLPv2),” in *Proc. Workshop on Ad hoc Communications*, in conjunction with 7th European Conference on Computer Supported Cooperative Work (ECSCW’01), (Bonn, Germany), Sept. 2001.

[8] E. Guttman, C. Perkins, J. Veizades, and M. Day, “Service location protocol, version 2 (IETF RFC 2608),” June 1999.

[9] R. Droms, “Automated configuration of TCP/IP with DHCP,” *IEEE Internet Computing*, pp. 45–53, July 1999.

[10] E. Guttman, C. E. Perkins, and J. Kempf, “Service Templates and Service: Schemes.” Internet RFC 2609, June 1999.

[11] B. Lehner, *KDE– und Qt Programmierung: GUI–Entwicklung für Linux*. Addison–Wesley, 2 ed., 2001.

[12] “KDesktop Environment Webpage.” http://www.kde.org.

[13] “KDevelop Website.” http://www.kdevelop.org/, 2001.

[14] J. Kempf, R. Moats, and P. S. Pierre, “Conversion of LDAP schemas to and from SLP templates (IETF RFC 2926),” Sept. 2000.

[15] A. B. Oppenheimer, “The Service Location Protocol and the Macintosh.” http://www2.opendoor.com/shareway/slp.html, Sept. 1999.

[16] Apple, “Mac OS 9 network browser.” http://www. apple.com/macos/feature9.html, 2001.

[17] E. Hughes, D. McCormack, M. Barbeau, and F. Bordeleau, “An application for discovery, configuration, and installation of SLP services,” in *Proceedings 5th Mitel Workshop on Innovation in Technology and Application (MICON 2000)*, (Ottawa), Aug. 2000.

[18] E. Hughes, D. McCormack, M. Barbeau, and F. Bordeleau, “Service recommendation using SLP,” in *Proceedings IEEE International Conference on Telecommunications*, (Bucharest), June 2001.

[19] J. Kempf and E. Guttman, “An API for service location (IETF RFC 2614),” June 1999.

[20] “OpenSLP Website.” http://www.openslp.org/, 2001.