IEEE International Conference on Telecommunications (ICT), Bucharest, June 2001

Service Recommendation using SLP

Evan Hughes, David McCormack,
Michel Barbeau, and Francis Bordeleau

Carleton University, School of Computer Science
1125 Colonel By Drive
Ottawa, ON Canada
K1S 5B6

Email: {ehughes,dmccormack}@texar.com, {barbeau,francis}@scs.carleton.ca

April 10, 2001

Abstract

As more services become available on networks, and as ad hoc network grows in popularity, the problem of service location will grow. Service location is the detection of a service through automated means, and the subsequent configuration of the service user to use that service.

A number of service discovery protocols currently exist, but there is no known framework for the evaluation and comparison of these protocols based solely on their service discovery characteristics. This paper presents the Service Discovery Model, a generic meta-model describing all common tasks of service discovery.

We then use the SDM illustrate the Service Location Protocol and extend it to support comparisons between services.

1 Introduction

Service discovery is a growing field within networking. There are currently a number of protocols that perform service discovery, such as Bluetooth's Service Discovery Protocol [blu,], Jini [Jin, 1998], Salutation [sal,], Universal Plug and Play [upn,], and the Service Location Protocol [Guttman et al., 1999]. However, there are few metrics or frameworks for evaluating and comparing each protocol. Some work has been done comparing specific protocols [Govea and Barbeau, 2000], however there are no known models of theoretical service discovery.

This paper presents a model for comparison of service discovery protocols called the Service Discovery Model (SDM), and then uses the SDM to illustrate the Service Location Protocol (SLP). The model is used as a tool for discussing an open issue within that protocol termed service selection facilition. Two strategies for service selection facilitation are discussed: the SLP Service Browser (SSB) and service recommendation.

In Section 2 we present the Service Discovery Model. In Section 3, we present the Service Location Protocol in terms of the Service Discovery Model. Section 4 explains service selection facilitation, the SSB and service recommendation. The final section concludes and outlines our further work.

2 Service Discovery Model

The Service Discovery Model (SDM) is a generalized model of service discovery. In any successful service discovery scenario, there are at least two actors; the service user, called the User Agent (UA); and the service provider, called the Service Agent (SA). There may be an additional third party called a DA; it acts as a lookup service, locating one or more SAs for the UA (called the Directory Agent, or DA). Any kind of agent is referred to generically as a *node*.

The SDM abstracts the activities of service discovery into three discreet phases, starting when the user decides it wants a service, and ending when the user is able to use that service, or when the it learns that there are no services available. Although the three phases are chronological, they are not synchronous: different parts of the same network involved in the same service discovery interaction may be at different phases of the SDM at the same time.

The SDM is divided into three discreet phases: the Query Phase, the Selection Phase, and the Configuration Phase.

2.1 The Query Phase

The goal of the Query Phase is for the UA to inform all interested nodes that it is looking for a service. Interested nodes are SAs that may be offering a candidate service, or DAs that know of SAs offering a candidate service. The UA begins the Query Phase by spontaneously sending a query message, which indicates that it is looking for a service of a particular type. The query may contain extra qualifiers about the requested service, such as a minimum quality of service, or a desired location.

The Query Phase continues until it is terminated by the UA. The UA may use any strategy to decide when to terminate the Query Phase; including timeouts, confirmations from the intended recipients, or it may simply rely on the best effort of the network to deliver the message.

2.2 The Selection Phase

The goal of the Selection Phase is to provide the UA with an SA that it may use. The SAs are selected by processing of the query. The Selection Phase is divided into two subphases: the Paring Subphase and the Decision Subphase.

Paring Subphase The purpose of the Paring Subphase is to determine which subset of the available SAs satisfies the UA's query. We call this set the *satisfier set*. Before the contents of the query are known, the satisfier set contains all known services. Each of these services are somehow tested, and those that do not satisfy the query are tested and removed from the satisfier set.

Although the Paring and Decision Subphases do not need to be synchronous, the Paring Subphase may not outlast the Decision Subphase.

Decision Subphase The objective of Decision Subphase is to locate which service in the satisfier set the UA should use. The complexity of this decision depends on the UAs requirements and the facilities of the protocol. There are two trivial cases: either the satisfier set is empty, meaning that there are no services available to the UA; or there is exactly one service in the satisfier set, in which case that one service is selected.[1] In other cases, where the satisfier set contains many available services, other means may be employed to decide which service the UA is to use. The decision process may be as simple as randomly picking one service; or as complex as finding the service most suited to the UA's task.

The Decision Subphase ends when the UA knows which service it is to use.

2.3 Configuration Phase

The goal of the Configuration Phase is to modify the UA's configuration such that it can use the SA selected in in the Selection Phase. Configuration may be as simple as opening a socket to the service, or it may be as complex as downloading and installing drivers.

The Configuration Phase ends when the UA is capable of interacting with the SA.

3 Service Discovery and SLP

The Service Location Protocol is a lightweight service discovery protocol designed to work on an enterprise IP network. It has been formalized in [Guttman et al., 1999].

[1] We assume that the UA needs only one service. If n instances of services are necessary, the trivial set size is n.

3.1 The Service Location Protocol

An SLP network consists of three types of entities: user agents (UAs), service agents (SAs), and directory agents (DAs). There are two SLP service discovery scenarios: standard service discovery, in which service discovery is brokered by the DA; and DAless service discovery, where the UA communicates directly with all SAs on the network. Each scenario is discussed below.

3.1.1 Service Discovery in SLP

SLP's standard service discovery scenario is illustrated in Figure 1. In this scenario, both UAs and SAs know of DAs – this information is discovered via DAless service discovery, discussed in Section 3.1.2. It is assumed that before this scenario begins, all SAs on the network have registered their services with the DAs on the network.

Query Phase The scenario begins with the UA composes and sends a service request message (SrvRqst) via TCP unicast to one of the DAs on the network. The SrvRqst contains a service type (stating the service required) and a predicate (describing the characteristics of the required service).

Paring Subphase The DA compares the service type of the SrvRqst with that of all registered services. The DA evaluates the predicate on each service of the requested type. The satisfier set consists of all services of the given type that conform to the predicate. The DA sends the satisfier set to the UA using a unicast service reply message (SrvRply).

Decision Subphase The UA decides which service to use. SLP does not dictate how the decision is made. Often the UA will simply use the first service returned. If the SrvRply is empty, the service discovery process ends here.

Configuration Phase SLP does not specifically state how a UA is to configure itself. However, it does allow UAs to download the characteristics of services in the form of *attributes*. The attributes can contain any data, including configuration information (e.g. the URIs of drivers) and parameters (e.g. public keys for security).

The final interaction of Figure 1 is an association, or some logical connection between the UA an the SA. Note however, that the UA may choose to ignore the SA, or simply not connect to it. SLP does not dictate that a service must be used after it is found.

3.1.2 DAless Service Discovery in SLP

SLP also provides a simpler form of service discovery that does not require a DA. Instead of the UA communicating with the DA via unicast, it communicates with all SAs using UDP multicast. Each SA performs the Paring Subphase locally. If the SA's service is the same as that requested in the SrvRqst,

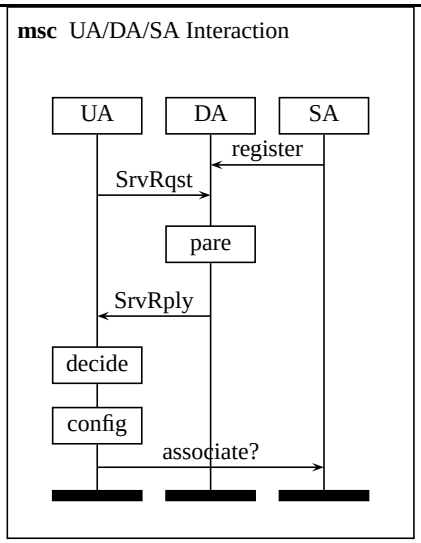

Figure 1: SLP interactions in context of the SDM.

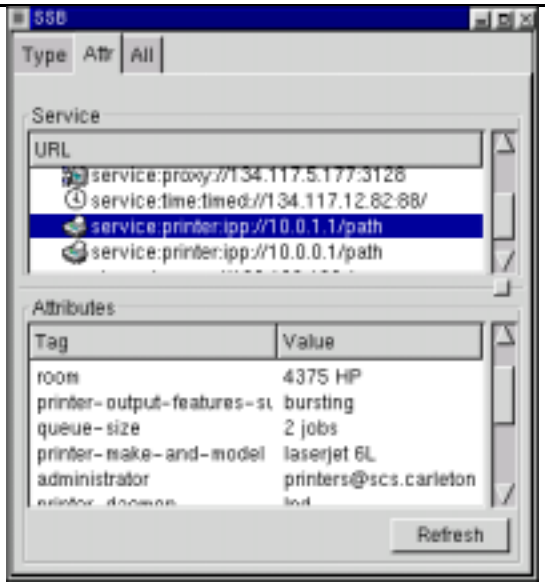

Figure 2: SLP Service Browser.

and its attributes satisfy the predicate (in other words, the SA finds itself in the satisfier set), it sends a unicast response to the UA. The UA treats the results in the same manner as those received from a DA.

4 Service Selection Facilitation in SLP

SLP does not have a defined behavior during the Decision Subphase: there is no standard strategy for deciding which member of the satisfier set the UA should use. We term this problem *service selection facilitation* [Barbeau, 2000].

We present two very different solutions to this problem. The first is the SLP Service Browser, a graphical front-end to the SLP-aware services on a network. The second is an extension to the SLP protocol called *service recommendation*.

4.1 The SLP Service Browser

The SLP Service Browser (SSB) is an application that allows a user to browse all SLP-aware services on the network. The user may search for all services, compose queries (including both service type and predicate), and launch service-specific commands on selected services. The SSB is designed in such a manner that it is useful to users who know nothing about SLP, as well as those who are well versed in SLP's functionality.

Figure 2 shows the attribute pane of the SSB. The top panel shows the list of known services. When the user selects a service, the attributes of that service are shown in the lower panel. The user can click on any displayed service and launch commands on that service. Commands are implemented as shell scripts, which may use the attributes of the service as variables. This means that SLP-unaware applications can be started by the SSB, effectively making the SSB a bridging technology between SLP-unaware clients, and SLP-aware services.[2]

The SSB answers the problem of service selection facilitation by allowing the user to manually compose queries and inspect the results. This allows the user to choose a service on any criteria. Although this solution is only valid when users are knowledgeable both about the SSB and the service they intend to use, it does simplify the users task, in that the SSB performs automatic service discovery and configuration.

4.2 Service Recommendation

SLP provides a means for comparing the attributes of services to constants. SLP does not provide a mechanism for comparing services to one another. If a UA receives a large satisfier set at the start of the Decision Subphase, it has no predefined mechanism for finding the service most suited to its task. Service recommendation is an extension to SLP to allow UAs to find SAs that offer the "best" level of service.

Currently, it is possible for UAs to perform makeshift service recommendation: the UA finds the satisfier set using a SrvRqst, but instead of immediately selecting a service, the UA queries each SA in the satisfier set for its attributes. As the attributes are received, the UA compares them, discarding those that are less than the "best" service the UA has thus far seen. After all of the attributes have been received, the UA is left with the service it is to use.

This makeshift solution faces two problems: the hardware resources available to the UA, and the nonstandard implementation of the comparison. In the discussion surrounding SLP, it is often assumed that UAs are resource poor (comparatively slow processor, little memory, and a slow network connection),

[2]Since [Kempf and Guttman, 1999] defines a means for representing SLP-unaware services with SAs, the combination of the SSB and the technology described in [Kempf and Guttman, 1999] can be used to launch and use any client on any service.

DAs are resource rich (fast processor, large memory, fast processor), and SAs are somewhere in between [srv, 2000]. Under these constraints, it is unreasonable to assume that the UA will have sufficient resources to complete the Decision Phase in a timely and correct manner. Meanwhile, the nonstandard comparative mechanism means that every UA implementor must implement their own version of the comparison: which is error prone and discourages code reuse.

Both of these issues can be dealt with by a two-fold solution. First, to lessen the burden of service discovery on the UA, move the responsibility of the Decision Subphase from the UA to the DA. Second, to ease the lot of the programmer, extend the predicate syntax to allow a comparison between services.

We present this solution by discussing first the current syntax of SLP predicates, and what must be done to extend them. Then, we discuss the capabilities of the new syntax. Finally, we conclude this section with an example of service recommendation at work.

4.2.1 Current SLP Predicate Support

SLP uses the LDAPv3 syntax for predicates [Howes, 1997]. LDAPv3 predicates include type-sensitive comparisons on attributes; and boolean operations on the results of comparisons. Supported comparisons include normal relational operators on string and integer attribute values, as well as wild-card comparisons on strings. Comparisons are represented as infix operations (named attributes are always the first operand, constants are the second), while boolean operations are prefix.

4.2.2 Service Recommendation and SLP Predicates

To allow the UA to generate arbitrary metrics expressing service quality, we introduce the concept of functions into the predicate syntax. A function optionally takes the place of an attribute in the predicate. Each function takes a mathematical expression as an argument. The argument consists of constants, mathematical operators, functions, and named attribute values.

In all cases, the predicate is evaluated for all services registered with the DA. When a function is reached, the DA synchronizes the predicate evaluation with that of all other services in the satisfier set; meaning that the DA evaluates the expression on each service in a logically simultaneous manner. The services are then sorted by the value that the expression evaluated to. The rank of the service is the return value of the function.

We propose a number of functions, including:

`rankHigh(`x`)` Evaluates x to a value, and returns the service's rank relative to the service with the greatest value. Sorts in ascending order.

`rankLow(`x`)` The same as `rankHigh()`, but returns the rank of the service in descending order (i.e., rank from the bottom).

	Printer$_1$	**Printer**$_2$	**Printer**$_3$
`speed`	3	10	6
`queueSize`	80	4	10

Figure 3: Printer Attributes

4.2.3 Example

Three printers exist on a network. The corresponding SAs maintain the following attributes: `speed`, and `queueSize`. `speed` is the number of seconds the printer takes to print a page. `queueSize` is the number of pages left in the printer's print queue.

A UA wishes to complete a print job as quickly as possible. We will consider the scenario where service recommendation is available before considering the scenario using unmodified SLP.

The UA initiates the service discovery scenario by sending a SrvRqst with the predicate `(@rankLow(speed * queueSize)=1)`. The argument to the `rankLow()` function determines how many seconds remain before the printer's current tasks are completed. `rankLow()` then orders the services by that value, returning the list rank of each service. When the DA receives the SrvRqst, it evaluates the predicate in the context of each printer. The expression evaluates to 240, 40, and 60 for Printer$_1$, Printer$_2$, and Printer$_3$ respectively. The value 3 is returned by `rankLow()` when the predicate is evaluated for Printer$_1$, 1 is returned when the predicate is evaluated for Printer$_2$, and 2 is returned when the predicate is evaluated for Printer$_3$. Since the predicate evaluates to true only when 1 is returned, the DA only informs the UA of the existence of Printer$_2$.

In terms of metrics, two messages are sent (the SrvRqst and SrvRply), and no comparisons take place in the UA. In general, the service recommendation scenario involves a constant number of messages (2) and comparisons (0).

Now consider a UA trying to make the same decision on a network where service recommendation is not available. As discussed in Section 4.2, the UA must send a SrvRqst to the DA requesting all known printer SAs. The DA responds with a SrvRply containing all three printers. The UA must then query each associated SA for attributes, and then receive each set of attributes and compare them locally.

In this case, two messages are sent to obtain the satisfier set, and an additional six messages (three requests and three responses) are sent to deliver the attributes. Two comparisons must take place in the UA. In general, the number of messages sent in this scenario is $2 + 2n$ (where n is the number of available services) and $n - 1$ comparisons.

5 Conclusion

We have presented a generic model for service discovery: the Service Discovery Model. It provides a framework for the discussion of service discovery protocols and services.

The SDM lead us to a description of service selection facilition in the context of SLP. Service selection facilitation is the problem of selecting a service from the satisfier set. Two solutions to service selection facilition were presented: the SSB and service recommendation.

The SSB is a user-oriented solution to service selection facilition. The user queries for services, evaluates the response, and selects a service. Service recommendation is an application-oriented solution to the same problem. It creates a framework that allows the DA (at the request of the UA) to compare services to each other and use that comparison to decide on inclusion in the satisfier set.

The SDM has not been rigorously tested on other protocols such as Bluetooth's Service Discovery Protocol, Jini, Salutation , and Universal Plug and Play. If the SDM is to stand as a general model for service discovery, it must be shown to be applicable to these other protocols.

In simulations, the service recommendation mechanism of Section 4.2 has been shown to improve the performance of simple applications. Further simulations and empirical tests are necessary to prove its applicability to service discovery.

References

[sal,] Salutation architecture specification.

[blu,] Specification of the bluetooth system.

[upn,] Universal Plug and Play Specification v1.0.

[Jin, 1998] (1998). Jini technology core platform specification. Technical report, Sun Microsystems, Palo Alto, United States of America.

[srv, 2000] (2000). Service Location Working Group Mail Archive. http://srvloc.org/hypermail/index.html.

[Barbeau, 2000] Barbeau, M. (2000). Service discovery protocols for ad hoc networking. Toronto, Canada. CASCON 2000 Workshop on ad hoc communications.

[Govea and Barbeau, 2000] Govea, J. and Barbeau, M. (2000). Comparison of bandwidth usage: Service location protocol and Jini. Technical Report Technical Report TR-00-06, School of Computer Science, Carleton University, Ottawa, Canada. http://www.scs.carleton.ca/ barbeau/Publications/2000/TR_00_06.pdf.

[Guttman et al., 1999] Guttman, E., Perkins, C., Veizades, J., and Day, M. (1999). RFC 2608: Service location protocol. Status: Proposed Standard.

[Howes, 1997] Howes, T. (1997). RFC 2254: The string representation of LDAP search filters. Status: Proposed Standard.

[Kempf and Guttman, 1999] Kempf, J. and Guttman, E. (1999). RFC 2614: An API for Service Location. Status: Proposed IETF Standard.