Improving SLP Efficiency and Extendability by Using Global Attributes and Preference Filters

Weibin Zhao and Henning Schulzrinne
Columbia University
New York, NY 10027
{zwb,hgs}@cs.columbia.edu

Abstract

This paper presents two new mechanisms, global attributes and preference filters, that improve the efficiency and extendability of the Service Location Protocol (SLP). Global attributes describe common service properties, which can be used to perform queries across different service types, and can be standardized for service management. Preference filters facilitate processing of search results, such as finding the best match, in SLP servers. They can reduce the amount of data transferred to the client, and better support thin clients with limited resources and capabilities.

1 Introduction

The Service Location Protocol (SLP [8]) provides a flexible framework for service discovery in IP networks. As more applications [15, 16, 1, 21, 17] employ SLP for various discovery purposes, there is a need to extend the functionality of SLP and improve its efficiency in order to meet new discovery requirements. Consider an example of finding all services that support the Stream Control Transmission Protocol (SCTP [18]). As an SLP Service Request (`SrvRqst`) can only query services of the same service type, an SLP User Agent (UA) needs three steps to accomplish this discovery: sending a Service Type Request (`SrvTypeRqst`) to obtain a list of service types, then using a separate `SrvRqst` to query services of each service type that support SCTP, and finally combining the query results together. Assuming there are n service types, then $n+1$ queries are needed, which is inefficient when there are many service types. Consider another example of finding a printer with the minimum queue length. Since SLP search filters (same as those in LDAP [10]) do not support the "minimum" function, an SLP UA needs to get information for all printers, sort them based on the queue length attribute, and choose the one with the minimum queue length. This procedure is inefficient when there are many printers. These two examples illustrate two discovery scenarios that are not well supported by current SLP, namely discovery across multiple service types and discovering the best match. To remedy this situation, we present two new SLP mechanisms, global attributes and preference filters. Global attributes describe common service properties, which can be used to perform queries across different service types, and can be standardized for service management. Preference filters facilitate processing of search results, such as finding the best match, in SLP servers. They can reduce the amount of data transferred to the client, and better support thin clients with limited resources and capabilities.

The rest of this paper is organized as follows. In Section 2, we describe how to define and use global attributes. In Section 3, we present the design and use of preference filters. We discuss implementation issues in Section 4, list related work in Section 5, and conclude in Section 6.

2 Global Attributes

In a service discovery system, a common service description framework is needed for services to advertise their attributes and for clients to specify their requests. A match of a client request with a service advertisement leads to a discovery. To facilitate service discovery, service attributes need to be standardized. One way to achieve this is to employ service templates. As different service types may have different attributes, service templates are often defined in terms of service types. However, some service attributes are independent of service types. For instance, it is increasingly common that services can be accessed via different transport protocols, such as TCP, UDP and SCTP. The transport protocol attribute describes a common service property independent of service types. We refer to a service attribute that is specific to a service type as a *local attribute*, and a service attribute that is independent of service types as a *global attribute*. Currently, SLP only supports local attributes: each service type defines its attribute set via a service template [7], and a `SrvRqst` always specifies a search filter (attribute predicate) along with a single service type.

2.1 Namespace, Definition and Usage

Enabling global attributes in SLP involves three aspects: assigning a separate namespace to global attributes, defining global attributes via attribute templates, and using them properly.

SLP does not explicitly employ namespaces for attribute naming. An attribute name is unique only within its service type. In other words, two different service types may use the same attribute name for different meanings. This will not cause any problems as long as a local attribute is always used along with its service type, though it would be more clear if each attribute name is prefixed with its service type, such as *nrsm-capacity* and *iptel-gw-capacity*. The situation becomes different when global attributes are used. Since a global attribute can be used with any service type, if it has the same name as a local attribute, then there will be a confusion on which is which. Therefore, a separate namespace is needed for global attributes. According to the common practice in SLP, any global attribute name shall begin with the "*service-*" prefix. Note that XML [3] also uses prefixes to define its namespaces.

We propose that a global attribute is defined using an attribute template (see [20] for a detailed specification), which is a simplified version of the service template [7]. Any service that uses a global attribute should import the attribute's definition into its service template, similar to the C include and Java import mechanisms. In this way, a global attribute only has one definition, and can be used consistently across all service types.

A global attribute can appear in any place where a local attribute is appropriate. In a `SrvRqst`, when local attributes are used, exactly one service type must be specified; but when only global attributes are used, multiple service types or a service type wildcard can be specified. Thus, we can query services across multiple or all service types via a single `SrvRqst`. For example, to find all services that support SCTP, we can use a `SrvRqst` that has a service type wildcard, and an attribute predicate of "*service-transport-protocol=sctp*".

2.2 Efficiency and Extendability

Global attributes describe common service properties across different service types, which improves SLP efficiency in two ways. First, a global attribute only needs to be defined once. Afterwards, it can be imported into any service template. This avoids defining the same attribute in multiple service templates, and ensures a consistent definition. Secondly, global attributes enable using a single `SrvRqst` to query services across multiple service types, which is more efficient than using multiple `SrvRqst` messages, one for each service type.

Global attributes can also be used for service management, such as using service identifier to uniquely and persistently identify a service, and using device identifier to identify all services provided by the same device[1]. By standardizing global attributes in this way, we can extend the functionality of SLP to meet new discovery requirements. To illustrate this usage, we show how to introduce service identifier and device identifier into SLP to support some useful discovery scenarios by defining them as global attributes.

[1]Since a device may have multiple IP addresses, a device identifier is needed to identify all URLs that belong to the same device. Otherwise, we cannot tell that a set of URLs containing different IP addresses belongs to the same device or to different devices.

2.3 Service Identifier and Device Identifier

A service identifier is a URI [2] (such as UUID [12] URI) without locator semantics. It is unique[2] and persistent[3]. A service can always be identified via its service identifier even if it has changed all other service attributes. Service identifiers can be used as keys in service registrations and queries, as in Jini [14] and UDDI [13]. For SLP, it employs "service" URLs as service keys. Unfortunately, a service's URLs are not persistent, e.g., a service may change its URLs when it moves. To support URL changes and to find multi-protocol services in SLP, we can use service identifiers as follows.

URL changes: Once a service has changed its URLs, a client cannot find the same service again by using its old URLs; but the client can find the same service again by using its service identifier.

Multi-protocol services: A service may support multiple access protocols, having a separate URL for each access protocol. For example, a multi-protocol printer that supports IPP [9] and LPR access protocols may have two URLs as follows: *service:printer:ipp://mpp.example.com* and *service:printer:lpr://mpp.example.com*. A multi-protocol service advertises each access protocol separately, but all advertisements use the same service identifier to indicate that they belong to the same service. A client can discover all advertisements of a multi-protocol service by specifying the service identifier and the service type in a `SrvRqst`.

Similar to a service identifier, a device identifier is a URI that uniquely and persistently identifies a device. We can use device identifiers to discover multi-function devices as follows.

Multi-function devices: A device may provide multiple types of services, such as scanning and printing services. A multi-function device advertises each service type separately, but all advertisements use the same device identifier to indicate that they belong to the same device. A client can discover all advertisements of a multi-function device by specifying the device identifier and a wildcard service type (or all the service types the device supports) in a `SrvRqst`.

Using service identifiers and device identifiers together, we can determine replicated services as follows.

Replicated services: All replicated servers for a service use the same service identifier but different device identifiers in their advertisements, whereas a device that provides the same service on multiple interfaces (i.e., multi-homed) or at different ports uses the same service identifier and the same device identifier in all its advertisements.

3 Preference Filters

Service Request (`SrvRqst`) is the most important SLP query interface, where a client specifies the properties of desired services via a search filter. When multiple services match the

[2]Two different services will never have the same service identifier.

[3]Once a service identifier is assigned to a service, it will never be changed or re-used.

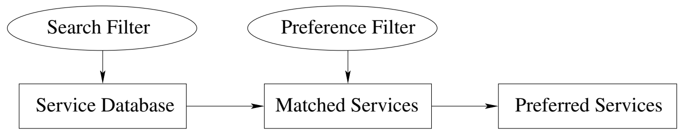

Figure 1: The processing of a `SrvRqst` that has a search filter and a preference filter

search filter, further processing of search results may be needed, such as to limit the number of results returned, to sort the results, or to find the best match. We refer to the processing of search results as preference filtering, which is described by a preference filter. Figure 1 illustrates the processing of a `SrvRqst` that has a search filter and a preference filter. First, the search filter is applied to the service registration database, which generates a set of matched services. Then the preference filter is applied to the matched services, which generates a set of preferred services.

Although preference filtering can be performed by UAs, having Directory Agents (DAs) or Service Agents (SAs) perform this function provides two advantages. First, it can reduce the amount of data transferred to the client, which is especially useful when the client uses a low bandwidth channel, such as a wireless channel. Secondly, it can better support thin clients with limited resources and capabilities. For example, some clients may not be able to receive a large amount of results, or to perform sort and choose the best match.

In the design of SLP preference filters, we choose selection and sort as two basic building blocks, and use them to compose generic preference filters.

3.1 Selection Filters

We designed an SLP *selection* extension [22] to specify selection filters. An SLP UA uses this extension in a `SrvRqst` to limit the number (say, n) of results to be returned. Also an SLP server (DA or SA) uses this extension in a Service Reply (`SrvRply`) to indicate the number (say, m) of search results before any preference filtering. If $n < m$, then only the first n results (in the sorted or unsorted result list) are returned, otherwise all m results are returned. As a special case, a UA may set n to 0 to obtain the number of search results without retrieving the results themselves.

3.2 Sort Filters

We designed an SLP *sort* extension [22] to specify sort filters. This extension carries a sort key list (Figure 2 shows its simplified specification using ABNF [4]). Each sort key has a key name (i.e., an attribute name), a type specifier (string or integer), an ordering specifier (increasing or decreasing), and an optional reference value. Although SLP has five attribute types (integer, string, boolean, opaque and keyword), we only consider integer sort and string sort since keyword attributes never need to

```
sort-key-list  =  sort-key / sort-key "," sort-key-list
sort-key       =  key-name ":" type ":" ordering
                  [":" ref-value]
type           =  "s" / "i"
                  ;"s" for string, "i" for integer
ordering       =  "+" / "-"
                  ;"+" for increasing order
                  ;"-" for decreasing order
```

Figure 2: The simplified sort key list specification in ABNF

be sorted[4], and boolean and opaque attributes can be sorted as strings if needed. Integer keys may have a reference value, as in *speed:i:+:12*, causing the sort to be based on the distance to the reference value, 12.

3.3 Generic Preference Filters

A generic preference filter is a list of selection and sort filters, which observes the following rules. First, two selection filters and two sort filters cannot appear in a row. Second, if the same number of sort and selection filters are used, the last one must be a selection filter. Finally, for two selection filters s_1 and s_2, if s_1 appears earlier than s_2, then the selected number of results specified in s_1 must be greater than that in s_2.

We use *select(n)* to denote a selection filter, and use *sort(sort-key-list)* to denote a sort filter. We show some useful preference filters next. First, using a sort filter followed by a selection filter can support best match, for instance, the minimum load is written as "*sort(load:i:+), select(1)*", the maximum speed is written as "*sort(speed:i:-), select(1)*", and the speed closest to a reference value 12 is written as "*sort(speed:i:+:12), select(1)*". Secondly, a selection filter can select multiple results, for instance, the top three in terms of speed is written as "*sort(speed:i:-), select(3)*". Similarly, a sort filter can sort on multiple keys, for instance, the minimum load among those having the maximum speed is written as "*sort(speed:i:-,load:i:+), select(1)*". Finally, multiple sort and selection filters can be used to construct more complex preference filters, for instance, the minimum load among the top three in terms of speed is written as "*sort(speed:i:-), select(3), sort(load:i:+), select(1)*".

4 Implementation

It is easy to add support for global attributes and preference filters to an SLP server, since only the processing of the `SrvRqst` message needs to be adjusted. When a `SrvRqst` uses local attributes, it should have exactly one service type, and is handled as before. If a `SrvRqst` uses only global attributes, it may have multiple service types or a service type wildcard. In this case, the service type information is ignored during the search, and

[4]SLP keyword attributes have no values.

then those search results that do not match any of the specified service types are discarded.

Similarly, for a `SrvRqst` with a preference filter, the filter is ignored during the search, and then it is applied to the search results. When the filter has multiple selection and sort filters, they must be processed in order, with the output of one filter as the input of the next filter. The output of the last filter is returned to the client.

We have implemented global attributes and preference filters in our release of enhanced SLP; the source code can be found at [5].

5 Related Work

Our work on SLP global attributes was motivated by Guttman's serviceid [6], which describes a way to introduce service identifiers into SLP. Guttman proposes to use service identifiers as SLP service keys and to move URLs into the attribute list. Unfortunately, if a service has multiple URLs and these URLs have different properties, then some form of hierarchical attributes is needed[5], which complicates SLP attribute list structure and service search. Our proposal is simpler by keeping URLs as service keys (in keeping with SLP flat attributes), and defining service identifier as a global attribute. Similar to global attributes, Jini [14] defines a set of common entry classes in the *net.jini.lookup.entry* package. When a Jini search specifies multiple interfaces (such as *Toaster* and *FireAlarm*), it means to find services that implement all specified interfaces (logic "and"). In contrast, when an SLP `SrvRqst` specifies multiple service types, it means to find services of any specified type (logic "or").

A number of service discovery systems and directory services support selection and sort on search results, but none of them support generic preference filtering by composing these two basic operations. For example, both Jini [14] and UDDI [13] allow a client to place a limit on the number of search results, but UDDI only supports sorting on name and date, and Jini does not support sorting. LDAP sort control [11] and paging control [19] come closer to our proposed preference filtering. But LDAP aims to send all search results back to the client via the paging control, whereas we simply try to send selected number of search results to the client. Furthermore, we support reference-based sort and more complex preference filtering by composing multiple selection and sort filters.

6 Conclusion

This paper described two simple but useful mechanisms, global attributes and preference filters, that improve SLP efficiency and extendability. We extended SLP attributes from local to global and enabled generic preference filtering on search results in SLP servers. Although we discuss these two mechanisms in terms of SLP, we expect that the rationale behind these mechanisms can be applied to other service discovery systems as well.

[5]Each URL needs to use different attributes or values for its description.

References

[1] M. Bakke et al. Finding iSCSI targets and name servers using SLP. Internet Draft, Internet Engineering Task Force, March 2002. Work in progress.

[2] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (URI): generic syntax. RFC 2396, Internet Engineering Task Force, August 1998.

[3] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible markup language (XML) 1.0 (second edition). W3C Recommendation REC-xml-20001006, World Wide Web Consortium (W3C), October 2000. Available at http://www.w3.org/XML/.

[4] D. Crocker and P. Overell. Augmented BNF for syntax specifications: ABNF. RFC 2234, Internet Engineering Task Force, November 1997.

[5] Service Location Protocol Enhancements. http://www.cs.columbia.edu/~zwb/project/slp.

[6] E. Guttman. The serviceid: URI scheme for service location. Internet Draft, Internet Engineering Task Force, January 2002. Work in progress.

[7] E. Guttman, C. Perkins, and J. Kempf. Service templates and service: Schemes. RFC 2609, Internet Engineering Task Force, June 1999.

[8] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. RFC 2608, Internet Engineering Task Force, June 1999.

[9] R. Herriot, S. Butler, P. Moore, R. Turner, and J. Wenn. Internet printing Protocol/1.1: encoding and transport. RFC 2910, Internet Engineering Task Force, September 2000.

[10] T. Howes. The string representation of LDAP search filters. RFC 2254, Internet Engineering Task Force, December 1997.

[11] T. Howes, M. Wahl, and A. Anantha. LDAP control extension for server side sorting of search results. RFC 2891, Internet Engineering Task Force, August 2000.

[12] Universal Unique Identifier. http://www.opengroup.org/onlinepubs/9629399/apdxa.htm.

[13] Universal Description Discovery and Integration. http://www.uddi.org/.

[14] JINI. http://java.sun.com/products/jini/, 1998.

[15] J. Kempf and G. Montenegro. Finding an RSIP server with SLP. RFC 3105, Internet Engineering Task Force, October 2001.

[16] J. Naugle, K. Kasthurirangan, and G. Ledford. TN3270E service location and session balancing. RFC 3049, Internet Engineering Task Force, January 2001.

[17] Todd Poynor. Automating infrastructure composition for internet services. In *The 15th Systems Administration Conference (LISA 2001)*, San Diego, California, December 2001.

[18] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream control transmission protocol. RFC 2960, Internet Engineering Task Force, October 2000.

[19] C. Weider, A. Herron, A. Anantha, and T. Howes. LDAP control extension for simple paged results manipulation. RFC 2696, Internet Engineering Task Force, September 1999.

[20] W. Zhao and H. Schulzrinne. Defining and using global service attributes in SLP. Internet Draft, Internet Engineering Task Force, June 2002. Work in progress.

[21] W. Zhao and H. Schulzrinne. Locating internet telephony gateways via SLP. Internet Draft, Internet Engineering Task Force, March 2002. Work in progress.

[22] W. Zhao, Henning Schulzrinne, Erik Guttman, Chatschik Bisdikian, and William Jerome. Selection and sort extension for SLP. Internet Draft, Internet Engineering Task Force, June 2002. Work in progress.