Customization for SLP Service Request and Reply

draft-zhao-slp-customization-00.txt

Status of This Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Copyright Notice

Copyright (C) The Internet Society (2001). All Rights Reserved.

Abstract

This document presents a lightweight mechanism for supporting customization in SLP where a UA specifies its customization request in the SrvRqst via an SLP extension, and the DA customizes the SrvRply according to the UA request. Two basic customization operations, sorting the result set and bounding the result size, are provided, and some complex customization requests are supported by composing these two basic operations. Furthermore, reference-based comparators are enabled to decide the result order. Customizing SLP SrvRqst and SrvRply messages enhances the basic SLP discovery.
scenario by also considering context information and user preference, and thus it can support value-added services and improve the SLP query efficiency.

1. Introduction

In the Service Location Protocol (SLP [1]), a User Agent (UA) discovers a desired service by specifying its properties (type, scope and attribute predicate) via a Service Request (SrvRqst) message, and a Directory Agent (DA) answers with a Service Reply (SrvRply) message carrying a list of URL entries for the matched services. Although performing discovery based on service properties is sufficient for most applications, there are some applications that need to incorporate context information (such as location) and user preference (such as order and size of the result set) in the SrvRqst and customize the SrvRply to tailor it to the user request.

Customizing SLP SrvRqst and SrvRply messages can provide several advantages. First, it enhances the basic SLP discovery scenario by also considering context information, and thus it can support value-added services, such as location-based discovery (find a service that is close to the user). Second, bounding the result size (number of URL entries) may be useful when the UA has limited resources or the UA uses a low-bandwidth channel. In some cases, a user may just want to find a few services, not tens or hundreds of them. Third, sorting the result set on some attribute(s) by a DA is more efficient than sorting it by a UA when some ordered result set is needed. For example, if a UA wants to find the available printers in order of speed (pages per minute), using the basic SLP queries, it needs to first send a SrvRqst to get a list of printers, then issue an Attribute Request (AttrRqst) for each printer to get its speed, and finally sort the printer list on speed. The overhead of multiple rounds of message exchanges and round-trip delays suggests that a sort at the DA side is more suitable for this discovery request.

In this document, we will present a lightweight mechanism to support customization in SLP where a UA specifies its customization request in the SrvRqst via an SLP extension, and the DA customizes the SrvRply according to the UA request. Two basic customization operations, sorting the result set and bounding the result size, are provided, and some complex customization requests are supported by composing these two basic operations. Furthermore, reference-based comparators are enabled to decide the result order.

The rest of this document is organized as follows: we first define terminology in Section 2, then present a design overview in Section 3. Section 4 defines the Customization extension and Section 5 defines the Comparator extension. We list constants in Section 6 and...
give security considerations in Section 7.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [3].

Customization Request

A customization request is specified via an SLP extension, which MAY be used in a SrvRqst. It describes the preferred way in which the DA SHOULD customize the result set for the corresponding SrvRply.

Reference-based Comparator

A reference-based comparator is a customized comparison function associated with an attribute of a service template [2], which MAY be used to compare the attribute value with a reference value. This comparison function receives two arguments (the second one is the reference value), and it is a non-negative, non-decreasing function of the difference of the two arguments. If two compared values are equal, it returns 0.

3. Design Overview

The processing of a SrvRqst can be viewed as having two stages: matching service properties to obtain a result set (may be empty), then customizing this result set according to the user request. Note that a customization may have effects only if the original result set has more than one URL entries. In other words, if the original result set is empty or has only one URL entry, then any customization will produce no effect, and the customization request can be safely ignored in this case.

3.1. Sorting and Bounding the Result Set

Sorting and bounding the result set are two basic types of customization. The sort operation refers to the natural sorting of alphanumeric characters (sort numbers on value or sort strings on lexicographic order). The bound operation refers to selecting the first N elements (assume bound N) in a list. If the number of elements in the list is less than N, then the bound operation returns the whole list.

We support sort and bound directly, and some complex customization requests by composing these two basic operations. For example, maximum speed (or minimum load) can be expressed as a sort followed
by a selection of the first entry, and minimum load in the top three
fast machines can be expressed as a \{sort, selection, sort, selection\} sequence. The sort can base on one or multiple attributes.

Note that similar customization operations, server side sort [5] and
paged result manipulation [6], are supported in LDAP [4]. However,
LDAP does not address composing these basic operations.

3.2. Using Reference-based Comparator

Sometimes the natural sort is not sufficient, a customized comparator
is needed to decide the order. One example is the reference-based
comparator which compares an attribute value with a reference value,
and returns a non-negative value as the difference metric for the
comparison. If two compared values are equal, it returns 0.

The reference-based comparator is useful for supporting location-
based discovery which needs to compare a service location with a
reference location. The difference metric returned by a reference-
based comparator is application dependent, i.e., application specific
information is needed to decide what is a best match for a reference
value. For example, if there is a printer in room 442 and room 458,
respectively, which printer is closer to a user in room 449?

A generic interface for the reference-based comparator is as follows.

```java
public float comparator(Object s1, Object s2)
/* compare object s1 with s2,
return a non-negative value as the difference metric
if s1 and s2 are equal, then return 0 */
```

4. Customization Extension

This extension is used in the SrvRqst message to specify the
customization request on the result set. Figure 1 gives its format.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Customization Ext. ID = TBD |  Next Extension Offset (NEO)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| NEO, contd. |C|S|B| reserved|   Sort-Order  |   Bound-Size  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Length of Attr-Value List   |         Attr-Value List       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1. Customization Extension
The customization request is specified using three basic operations: calling (C) the reference-based comparator(s) for those attributes that have a reference value, sorting (S) the result set on the specified attributes in the Sort-Order, and bounding (B) the result size to the Bound-Size. An operation is selected by setting its corresponding bit to 1.

The attribute-value list is used by the call (C) and sort (S) operations. The call operation is applied to attributes that have a reference value. Note that an attribute can have at most one reference-based comparator. The sort operation is based on all specified attributes, and when multiple attributes are present, they are in order of highest to lowest sort key precedence. If a customization request only has the bound (B) operation, then the length of the attribute-value list is zero.

The Sort-Order is given in a bit-field where 0 is used for increasing and 1 is used for decreasing sorting order. The Sort-Order field can specify at most 8 attribute sorting orders corresponding to their positions in the attribute-value list. For example, 0x00 means that all specified attributes are sorted in increasing order, and 0xA0 means that the first and the third specified attribute are sorted in decreasing order.

The Bound-Size specifies the maximum number of URL entries that can be carried by the corresponding SrvRply.

The three basic operations can be used individually or combined in some way. If multiple operations are chosen, they MUST be processed in the following order: call (C), sort (S), and bound (B). Furthermore, if the call operation is selected, then the sort operation SHOULD also be selected since the purpose of using a reference-based comparator is to decide the result order.

When multiple Customization extensions are present in a SrvRqst message, they are processed in sequence.

4.1. Examples

We use a tuple (operation-flag, sort-order, bound-size, attribute-list) to represent a Customization extension. A customization request that needs multiple Customization extensions is illustrated as multiple tuples.

Example 1. sort in decreasing speed:

\[(\text{sort}, 0x80, \text{NA}, \{\text{speed}\})\]
Example 2. bound to three URLs
(bound, NA, 3, NA)

Example 3. one minimum load:
(sort & bound, 0x00, 1, {load})

Example 4. top three fast speed:
(sort & bound, 0x80, 3, {speed})

Example 5. one minimum load in the top three fast speed:
(sort & bound, 0x80, 3, {speed}) and
(sort & bound, 0x00, 1, {load})

Example 6. sort in decreasing speed and increasing load:
(sort, 0x80, NA, {speed,load})

Example 7. the nearest service:
(call & sort & bound, 0x00, 1, {location})

4.2. Client-Server Interaction

A UA MAY use the Customization extension in a unicast SrvRqst sent to
a DA to specify its customization request. However, a UA SHOULD NOT
use this extension in a multicast SrvRqst since each SA will answer a
SrvRply individually, and no customization can be made by SAs.

For a SrvRqst that has the Customization extension, a DA MUST return
an OPTION_NOT_UNDERSTOOD [1] error if the DA does not support the
Customization extension or it cannot perform the requested
customization.

5. Comparator Extension

+------------------+-+---------------------+
| Comparator Ext. ID = TBD | Next Extension Offset (NEO) |
+------------------+-+---------------------+
| NEO, contd. | Length of Attribute Str. | Attr. Str. |
+------------------+-+---------------------+
| U | Language | Length of Comparator URL or Code |
+------------------+-+---------------------+
| Comparator URL or Code |
+------------------+-+---------------------+

Figure 2. Comparator Extension

This extension is used in the SrvReg message to specify a customized
comparison function for an attribute of a service template. Figure 2 gives its format.

If the U bit is set to 1, then the comparator is specified via a URL, otherwise its code is given in this extension. The Language field specifies a platform independent programming language (such as script language) in which the comparator is written. Currently, two languages are defined: Java class (0x01) and Perl script (0x02).

5.1. Client-Server Interaction

For a SrvReg that has the Comparator extension, a DA MUST return an OPTION_NOT_UNDERSTOOD error if the DA does not support the Comparator extension, cannot download the program code from the specified URL, or does not understand the programming language.

6. Constants

<table>
<thead>
<tr>
<th>Extension ID</th>
<th>TBD</th>
<th>(Section 4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comparator</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. Security Considerations

A DA SHOULD authenticate a reference-based comparator before accepting it, which can be done by using standard SLP authentication mechanism for the SrvReg message. If a comparator is specified via a URL, some external security mechanism SHOULD be used for downloading the program. As a registered comparator MAY crash, a DA SHOULD handle failures properly so that a failed comparator will not crash the whole system.

8. References


9. Authors’ Addresses

Weibin Zhao
Henning Schulzrinne
Department of Computer Science
Columbia University
1214 Amsterdam Avenue, MC 0401
New York, NY 10027-7003
Email: {zwb,hgs}@cs.columbia.edu

Chatschik Bisdikian
William F. Jerome
IBM T. J. Watson Research Center
P.O.Box 218
Yorktown Heights, NY 10598-0218
Email: {bisdik,wfj}@us.ibm.com

10. Full Copyright Statement

Copyright (C) The Internet Society (2001). All Rights Reserved.

This document and translations of it may be copied and furnished to
others, and derivative works that comment on or otherwise explain it
or assist in its implementation may be prepared, copied, published
and distributed, in whole or in part, without restriction of any
kind, provided that the above copyright notice and this paragraph are
included on all such copies and derivative works. However, this
document itself may not be modified in any way, such as by removing
the copyright notice or references to the Internet Society or other
Internet organizations, except as needed for the purpose of
developing Internet standards in which case the procedures for
copyrights defined in the Internet Standards process must be
followed, or as required to translate it into languages other than
English.

The limited permissions granted above are perpetual and will not be
revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an
"AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING
TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING
BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION
HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.