RESOLUTION of UNIFORM RESOURCE IDENTIFIERS using Domain Name System

Background Information

Uniform Resource Locators (URLs) are the foundation of the World Wide Web, and are a vital Internet technology. However, they have proven to be brittle in practice. The basic problem is that URLs typically identify a particular path to a file on a particular host. There is no graceful way of changing the path or host once the URL has been assigned. Neither is there a graceful way of replicating the resource located by the URL to achieve better network utilization and/or fault tolerance. Uniform Resource Names (URNs) have been hypothesized as a adjunct to URLs that would overcome such problems. URNs and URLs are both instances of a broader class of identifiers known as Uniform Resource Identifiers (URIs).

The requirements document for URN resolution systems[15] defines the concept of a "resolver discovery service". This document describes the first, experimental, RDS. It is implemented by a new DNS Resource Record, NAPTR (Naming Authority PoinTeR), that provides rules for mapping parts of URIs to domain names. By changing the mapping rules, we can change the host that is contacted to resolve a URI. This will allow a more graceful handling of URLs over long time periods, and forms the foundation for a new proposal for Uniform Resource Names.

In addition to locating resolvers, the NAPTR provides for other naming systems to be grandfathered into the URN world, provides independence between the name assignment system and the resolution protocol system, and allows multiple services (Name to Location, Name to Description, Name to Resource, ...) to be offered. In conjunction with the SRV RR, the NAPTR record allows those services to be replicated for the purposes of fault tolerance and load balancing.

Introduction

Uniform Resource Locators have been a significant advance in retrieving Internet-accessible resources. However, their brittle nature over time has been recognized for several years. The Uniform Resource Identifier working group proposed the development of Uniform Resource Names to serve as persistent, location-independent identifiers for Internet resources in order to overcome most of the problems with URLs. RFC-1737 [1] sets forth requirements on URNs.

During the lifetime of the URI-WG, a number of URN proposals were generated. The developers of several of those proposals met in a series of meetings, resulting in a compromise known as the Knoxville framework. The major principle behind the Knoxville framework is that the resolution system must be separate from the way names are assigned. This is in marked contrast to most URLs, which identify the host to contact and the protocol to use. Readers are referred to [2] for background on the Knoxville framework and for additional information on the context and purpose of this proposal.

Separating the way names are resolved from the way they are constructed provides several benefits. It allows multiple naming approaches and resolution approaches to compete, as it allows different protocols and resolvers to be used. There is just one problem with such a separation - how do we resolve a name when it can't give us directions to its resolver?

For the short term, DNS is the obvious candidate for the resolution framework, since it is widely deployed and understood. However, it is not appropriate to use DNS to maintain information on a per-resource basis. First of all, DNS was never intended to handle that many records. Second, the limited record size is inappropriate for catalog information. Third, domain names are not appropriate as URNs.

Therefore our approach is to use DNS to locate "resolvers" that can provide information on individual resources, potentially including the resource itself. To accomplish this, we "rewrite" the URI into a domain name following the rules provided in NAPTR records. Rewrite rules provide considerable power, which is important when trying to meet the goals listed above. However, collections of rules can become difficult to understand. To lessen this problem, the NAPTR rules are *always* applied to the original URI, *never* to the output of previous rules.

Locating a resolver through the rewrite procedure may take multiple steps, but the beginning is always the same. The start of the URI is scanned to extract its colon-delimited prefix. (For URNs, the prefix is always "urn:" and we extract the following colon-delimited namespace identifier. [3]). NAPTR resolution begins by taking the extracted string, appending the well-known suffix ".urn.net", and querying the DNS for NAPTR records at that domain name. Based on the results of this query, zero or more additional DNS queries may be needed to locate resolvers for the URI. The details of the conversation between the client and the resolver thus located are outside the bounds of this draft. Three brief examples of this procedure are given in the next section.

The NAPTR RR provides the level of indirection needed to keep the naming system independent of the resolution system, its protocols, and services. Coupled with the new SRV resource record proposal[4] there is also the potential for replicating the resolver on multiple hosts, overcoming some of the most significant problems of URLs. This is an important and subtle point. Not only do the NAPTR and SRV records allow us to replicate the resource, we can replicate the resolvers that know about the replicated resource. Preventing a single point of failure at the resolver level is a significant benefit. Separating the resolution procedure from the way names are constructed has additional benefits. Different resolution procedures can be used over time, and resolution procedures that are determined to be useful can be extended to deal with additional namespaces.

Caveats

The NAPTR proposal is the first resolution procedure to be considered by the URN-WG. There are several concerns about the proposal which have motivated the group to recommend it for publication as an Experimental rather than a standards-track RFC.

First, URN resolution is new to the IETF and we wish to gain operational experience before recommending any procedure for the standards track. Second, the NAPTR proposal is based on DNS and consequently inherits concerns about security and administration. The recent advancement of the DNSSEC and secure update drafts to Proposed Standard reduce these concerns, but we wish to experiment with those new capabilities in the context of URN administration. A third area of concern is the potential for a noticeable impact on the DNS. We believe that the proposal makes appropriate use of caching and additional information, but it is best to go slow where the potential for impact on a core system like the DNS is concerned. Fourth, the rewrite rules in the NAPTR proposal are based on regular expressions. Since regular expressions are difficult for humans to construct correctly, concerns exist about the usability and maintainability of the rules. This is especially true where international character sets are concerned. Finally, the URN-WG is developing a requirements document for URN Resolution Services[15], but that document is not complete. That document needs to precede any resolution service proposals on the standards track.

Brief overview and examples of the NAPTR RR

A detailed description of the NAPTR RR will be given later, but to give a flavor for the proposal we first give a simple description of the record and three examples of its use.

The key fields in the NAPTR RR are order, preference, service, flags, regexp, and replacement:

- The order field specifies the order in which records MUST be processed when multiple NAPTR records are returned in response to a single query. A naming authority may have delegated a portion of its namespace to another agency. Evaluating the NAPTR records in the correct order is necessary for delegation to work properly.

- The preference field specifies the order in which records SHOULD be processed when multiple NAPTR records have the same value of "order". This field lets a service provider specify the order in which resolvers are contacted, so that more capable machines are contacted in preference to less capable ones.
- The service field specifies the resolution protocol and resolution service(s) that will be available if the rewrite specified by the regexp or replacement fields is applied. Resolution protocols are the protocols used to talk with a resolver. They will be specified in other documents, such as [5]. Resolution services are operations such as N2R (URN to Resource), N2L (URN to URL), N2C (URN to URC), etc. These will be discussed in the URN Resolution Services document[6], and their behavior in a particular resolution protocol will be given in the specification for that protocol (see [5] for a concrete example).
- The flags field contains modifiers that affect what happens in the next DNS lookup, typically for optimizing the process. Flags may also affect the interpretation of the other fields in the record, therefore, clients MUST skip NAPTR records which contain an unknown flag value.
- The regexp field is one of two fields used for the rewrite rules, and is the core concept of the NAPTR record. The regexp field is a String containing a sed-like substitution expression. (The actual grammar for the substitution expressions is given later in this draft). The substitution expression is applied to the original URN to determine the next domain name to be queried. The regexp field should be used when the domain name to be generated is conditional on information in the URI. If the next domain name is always known, which is anticipated to be a common occurrence, the replacement field should be used instead.
- The replacement field is the other field that may be used for the rewrite rule. It is an optimization of the rewrite process for the case where the next domain name is fixed instead of being conditional on the content of the URI. The replacement field is a domain name (subject to compression if a DNS sender knows that a given recipient is able to decompress names in this RR type's RDATA field). If the rewrite is more complex than a simple substitution of a domain name, the replacement field should be set to . and the regexp field used.

Note that the client applies all the substitutions and performs all lookups, they are not performed in the DNS servers. Note also that it is the belief of the developers of this document that regexps should rarely be used. The replacement field seems adequate for the vast majority of situations. Regexps are only necessary when portions of a namespace are to be delegated to different resolvers. Finally, note that the regexp and replacement fields are, at present, mutually exclusive. However, developers of client software should be aware that a new flag might be defined which requires values in both fields.

Example 1

Consider a URN that uses the hypothetical DUNS namespace. DUNS numbers are identifiers for approximately 30 million registered businesses around the world, assigned and maintained by Dunn and Bradstreet. The URN might look like:

```
urn:duns:002372413:annual-report-1997
```

The first step in the resolution process is to find out about the DUNS namespace. The namespace identifier, "duns", is extracted from the URN, prepended to urn.net, and the NAPTRs for duns.urn.net looked up. It might return records of the form:

```
duns.urn.net
;;      order pref flags service           regexp         replacement
 IN NAPTR 100  10  "s"  "dunslink+N2L+N2C"  ""   dunslink.udp.isi.dandb.com
 IN NAPTR 100  20  "s"  "rcds+N2C"          ""   rcds.udp.isi.dandb.com
 IN NAPTR 100  30  "s"  "http+N2L+N2C+N2R"  ""   http.tcp.isi.dandb.com
```

The order field contains equal values, indicating that no name delegation order has to be followed. The preference field indicates that the provider would like clients to use the special dunslink protocol, followed by the RCDS protocol, and that HTTP is offered as a last resort. All the records specify the "s" flag, which will be explained momentarily. The service fields say that if we speak dunslink, we will be able to issue either the N2L or N2C requests to obtain a URL or a URC (description) of the resource. The Resource Cataloging and Distribution Service (RCDS)[7] could be used to get a URC for the resource, while HTTP could be used to get a URL, URC, or the resource itself. All the records supply the next domain name to query, none of them need to be rewritten with the aid of regular expressions.

The general case might require multiple NAPTR rewrites to locate a resolver, but eventually we will come to the "terminal NAPTR". Once we have the terminal NAPTR, our next probe into the DNS will be for a SRV or A record instead of another NAPTR. Rather than probing for a non-existent NAPTR record to terminate the loop, the flags field is used to indicate a terminal lookup. If it has a value of "s", the next lookup should be for SRV RRs, "a" denotes that A records should sought. A "p" flag is also provided to indicate that the next action is Protocol-specific, but that looking up another NAPTR will not be part of it.

Since our example RR specified the "s" flag, it was terminal. Assuming our client does not know the dunslink protocol, our next action is to lookup SRV RRs for rcds.udp.isi.dandb.com, which will tell us hosts that can provide the necessary resolution service. That lookup might return:

```
;;                          Pref Weight Port Target
rcds.udp.isi.dandb.com IN SRV 0    0    1000 defduns.isi.dandb.com
                       IN SRV 0    0    1000 dbmirror.com.au
                       IN SRV 0    0    1000 ukmirror.com.uk
```

telling us three hosts that could actually do the resolution, and giving us the port we should use to talk to their RCDS server. (The reader is referred to the SRV proposal [4] for the interpretation of the fields above).

There is opportunity for significant optimization here. We can return the SRV records as additional information for terminal NAPTRs (and the A records as additional information for those SRVs). While this recursive provision of additional information is not explicitly blessed in the DNS specifications, it is not forbidden, and BIND does take advantage of it [8]. This is a significant optimization. In conjunction with a long TTL for *.urn.net records, the average number of probes to DNS for resolving DUNS URNs would approach one. Therefore, DNS server implementors SHOULD provide additional information with NAPTR responses. The additional information will be either SRV or A records. If SRV records are available, their A records should be provided as recursive additional information.

Note that the example NAPTR records above are intended to represent the reply the client will see. They are not quite identical to what the domain administrator would put into the zone files. For one thing, the administrator should supply the trailing '.' character on any FQDNs.

Example 2

Consider a URN namespace based on MIME Content-Ids. The URN might look like this:

```
urn:cid:199606121851.1@mordred.gatech.edu
```

(Note that this example is chosen for pedagogical purposes, and does not conform to the recently-approved CID URL scheme.)

The first step in the resolution process is to find out about the CID namespace. The namespace identifier, cid, is extracted from the URN, prepended to urn.net, and the NAPTR for cid.urn.net looked up. It might return records of the form:

```
cid.urn.net
 ;;      order pref flags service       regexp          replacement
```

```
   IN NAPTR 100   10   ""     ""  "/urn:cid:.+@([^\.]+\.)(*)$/\2/i"    .
```

We have only one NAPTR response, so ordering the responses is not a problem. The replacement field is empty, so we check the regexp field and use the pattern provided there. We apply that regexp to the entire URN to see if it matches, which it does. The \2 part of the substitution expression returns the string "gatech.edu". Since the flags field does not contain "s" or "a", the lookup is not terminal and our next probe to DNS is for more NAPTR records: lookup(query=NAPTR, "gatech.edu").

Note that the rule does not extract the full domain name from the CID, instead it assumes the CID comes from a host and extracts its domain. While all hosts, such as mordred, could have their very own NAPTR, maintaining those records for all the machines at a site as large as Georgia Tech would be an intolerable burden. Wildcards are not appropriate here since they only return results when there is no exactly matching names already in the system.

The record returned from the query on "gatech.edu" might look like:

gatech.edu IN NAPTR ;; order pref flags service regexp replacement IN NAPTR 100 50 "s" "z3950+N2L+N2C" "" z3950.tcp.gatech.edu IN NAPTR 100 50 "s" "rcds+N2C" "" rcds.udp.gatech.edu IN NAPTR 100 50 "s" "http+N2L+N2C+N2R" "" http.tcp.gatech.edu

Continuing with our example, we note that the values of the order and preference fields are equal in all records, so the client is free to pick any record. The flags field tells us that these are the last NAPTR patterns we should see, and after the rewrite (a simple replacement in this case) we should look up SRV records to get information on the hosts that can provide the necessary service.

Assuming we prefer the Z39.50 protocol, our lookup might return:

```
;;                          Pref Weight   Port Target
z3950.tcp.gatech.edu IN SRV 0    0       1000 z3950.gatech.edu
                     IN SRV 0    0       1000 z3950.cc.gatech.edu
                     IN SRV 0    0       1000 z3950.uga.edu
```

telling us three hosts that could actually do the resolution, and giving us the port we should use to talk to their Z39.50 server.

Recall that the regular expression used \2 to extract a domain name from the CID, and \. for matching the literal '.' characters seperating the domain name components. Since '\' is the escape character, literal occurances of a backslash must be escaped by another backslash. For the case of the cid.urn.net record above, the regular expression entered into the zone file should be "/urn:cid:.+@([^\\.]+\\.)(*)$/\\2/i". When the client code actually receives the record, the pattern will have been converted to "/urn:cid:.+@([^\.]+\.)(*)$/\2/i".

Example 3

Even if URN systems were in place now, there would still be a tremendous number of URLs. It should be possible to develop a URN resolution system that can also provide location independence for those URLs. This is related to the requirement in [1] to be able to grandfather in names from other naming systems, such as ISO Formal Public Identifiers, Library of Congress Call Numbers, ISBNs, ISSNs, etc.

The NAPTR RR could also be used for URLs that have already been assigned. Assume we have the URL for a very popular piece of software that the publisher wishes to mirror at multiple sites around the world:

```
     http://www.foo.com/software/latest-beta.exe
```

We extract the prefix, "http", and lookup NAPTR records for http.urn.net. This might return a record of the form

http.urn.net IN NAPTR ;; order pref flags service regexp replacement 100 90 "" "" "!http://([^/:]+)!\1!i" .

This expression returns everything after the first double slash and before the next slash or colon. (We use the '!' character to delimit the parts of the substitution expression. Otherwise we would have to use backslashes to escape the forward slashes, and would have a regexp in the zone file that looked like "/http:\\/\\/([^\\/:]+)/\\1/i".).

Applying this pattern to the URL extracts "www.foo.com". Looking up NAPTR records for that might return:

www.foo.com ;; order pref flags service regexp replacement IN NAPTR 100 100 "s" "http+L2R" "" http.tcp.foo.com IN NAPTR 100 100 "s" "ftp+L2R" "" ftp.tcp.foo.com

Looking up SRV records for http.tcp.foo.com would return information on the hosts that foo.com has designated to be its mirror sites. The client can then pick one for the user.

NAPTR RR Format

The format of the NAPTR RR is given below. The DNS type code for NAPTR is 35.

Domain TTL Class Order Preference Flags Service Regexp Replacement

where:

Domain
: The domain name this resource record refers to.

Standard
: DNS Time To Live field

ss
: Standard DNS meaning

er
: A 16-bit integer specifying the order in which the NAPTR records MUST be processed to ensure correct delegation of portions of the namespace over time. Low numbers are processed before high numbers, and once a NAPTR is found that "matches" a URN, the client MUST NOT consider any NAPTRs with a higher value for order.

Preference
: A 16-bit integer which specifies the order in which NAPTR records with equal "order" values SHOULD be processed, low numbers being processed before high numbers. This is similar to the preference field in an MX record, and is used so domain administrators can direct clients towards more capable hosts or lighter weight protocols.

Flags
: A String giving flags to control aspects of the rewriting and interpretation of the fields in the record. Flags are single characters from the set [A-Z0-9]. The case of the alphabetic characters is not significant.

  At this time only three flags, "S", "A", and "P", are defined. "S" means that the next lookup should be for SRV records instead of NAPTR records. "A" means that the next lookup should be for A records. The "P" flag says that the remainder of the resolution shall be carried out in a Protocol-specific fashion, and we should not do any more DNS queries.

  The remaining alphabetic flags are reserved. The numeric flags may be used for local experimentation. The S, A, and P flags are all mutually exclusive, and resolution libraries MAY signal an error if more than one is given. (Experimental code and code for assisting in the creation of NAPTRs would be more likely to signal such an error than a client such as a browser). We anticipate that multiple flags will be allowed in the future, so implementers MUST NOT assume that the flags field can only contain 0 or 1 characters. Finally, if a client encounters a record with an unknown flag, it MUST ignore it and move to the next record. This test takes precedence even over the "order" field. Since flags can control the interpretation placed on fields, a novel flag might change the interpretation of the regexp and/or replacement fields such that it is impossible to determine if a record matched a URN.

vice
: Specifies the resolution service(s) available down this rewrite path. It may also specify the particular

protocol that is used to talk with a resolver. A protocol MUST be specified if the flags field states that the NAPTR is terminal. If a protocol is specified, but the flags field does not state that the NAPTR is terminal, the next lookup MUST be for a NAPTR. The client MAY choose not to perform the next lookup if the protocol is unknown, but that behavior MUST NOT be relied upon.

The service field may take any of the values below (using the Augmented BNF of RFC 822[9]):

```
     service_field = [ [protocol] *("+" rs)]
     protocol      = ALPHA *31ALPHANUM
     rs            = ALPHA *31ALPHANUM
  // The protocol and rs fields are limited to 32
  // characters and must start with an alphabetic.
  // The current set of "known" strings are:
  // protocol      = "rcds" / "thttp" / "hdl" / "rwhois" / "z3950"
  // rs            = "N2L" / "N2Ls" / "N2R" / "N2Rs" / "N2C"
  //               / "N2Ns" / "L2R" / "L2Ns" / "L2Ls" / "L2C"
```

i.e. an optional protocol specification followed by 0 or more resolution services. Each resolution service is indicated by an initial '+' character.

Note that the empty string is also a valid service field. This will typically be seen at the top levels of a namespace, when it is impossible to know what services and protocols will be offered by a particular publisher within that name space.

At this time the known protocols are rcds[7], hdl[10] (binary, UDP-based protocols), thttp[5] (a textual, TCP-based protocol), rwhois[11] (textual, UDP or TCP based), and Z39.50[12] (binary, TCP-based). More will be allowed later. The names of the protocols must be formed from the characters [a-Z0-9]. Case of the characters is not significant.

The service requests currently allowed will be described in more detail in [6], but in brief they are:

N2L
: Given a URN, return a URL

N2L
: Given a URN, return a set of URLs

N2R
: Given a URN, return an instance of the resource.

N2R
: Given a URN, return multiple instances of the resouce, typically encoded using multipart/alternative.

N2C
: Given a URN, return a collection of meta-information on the named resource. The format of this response is the subject of another document.

N2N
: Given a URN, return all URNs that are also identifers for the resource.

L2R
: Given a URL, return the resource.

L2N
: Given a URL, return all the URNs that are identifiers for the resource.

L2L
: Given a URL, return all the URLs for instances of of the same resource.

L2C
: Given a URL, return a description of the resource.

The actual format of the service request and response will be determined by the resolution protocol, and is the subject for other documents (e.g. [5]). Protocols need not offer all services. The labels for service requests shall be formed from the set of characters [A-Z0-9]. The case of the alphabetic characters is not significant.

Regexp

A STRING containing a substitution expression that is applied to the original URI in order to construct the next domain name to lookup. The grammar of the substitution expression is given in the next section.

Replacement

The next NAME to query for NAPTR, SRV, or A records depending on the value of the flags field. As mentioned above, this may be compressed.

Substitution Expression Grammar

The content of the regexp field is a substitution expression. True sed(1) substitution expressions are not appropriate for use in this application for a variety of reasons, therefore the contents of the regexp field MUST follow the grammar below:

```
  subst_expr   = delim-char  ere  delim-char  repl  delim-char  *flags
  delim-char   = "/" / "!" / ... (Any non-digit or non-flag character other
                 than backslash '\'. All occurances of a delim_char in a
                 subst_expr must be the same character.)
  ere          = POSIX Extended Regular Expression (see [13], section 2.8.4)
  repl         = dns_str /  backref / repl dns_str  / repl backref
  dns_str      = 1*DNS_CHAR
  backref      = "\" 1POS_DIGIT
  flags        = "i"
  DNS_CHAR     = "-" / "0" / ... / "9" / "a" / ... / "z" / "A" / ... / "Z"
  POS_DIGIT    = "1" / "2" / ... / "9"  ; 0 is not an allowed backref value
 domain name (see RFC-1123 [14]).
```

The result of applying the substitution expression to the original URI MUST result in a string that obeys the syntax for DNS host names [14]. Since it is possible for the regexp field to be improperly specified, such that a non-conforming host name can be constructed, client software SHOULD verify that the result is a legal host name before making queries on it.

Backref expressions in the repl portion of the substitution expression are replaced by the (possibly empty) string of characters enclosed by '(' and ')' in the ERE portion of the substitution expression. N is a single digit from 1 through 9, inclusive. It specifies the N'th backref expression, the one that begins with the N'th '(' and continues to the matching ')'. For example, the ERE (A(B(C)DE)(F)G) has backref expressions: \1 = ABCDEFG \2 = BCDE \3 = C \4 = F \5..\9 = error - no matching subexpression

The "i" flag indicates that the ERE matching SHALL be performed in a case-insensitive fashion. Furthermore, any backref replacements MAY be normalized to lower case when the "i" flag is given.

The first character in the substitution expression shall be used as the character that delimits the components of the substitution expression. There must be exactly three non-escaped occurrences of the delimiter character in a substitution expression. Since escaped occurrences of the delimiter character will be interpreted as occurrences of that character, digits MUST NOT be used as delimiters. Backrefs would be confused with literal digits were this allowed. Similarly, if flags are specified in the substitution expression, the delimiter character must not also be a flag character.

Advice to domain administrators

Beware of regular expressions. Not only are they a pain to get correct on their own, but there is the previously mentioned interaction with DNS. Any backslashes in a regexp must be entered twice in a zone file in order to appear once in a query response. More seriously, the need for double backslashes has probably not been tested by all implementors of DNS servers. We anticipate that urn.net will be the heaviest user of regexps. Only when delegating portions of namespaces should the typical domain administrator need to use regexps.

On a related note, beware of interactions with the shell when manipulating regexps from the command line. Since '\' is a common escape character in shells, there is a good chance that when you think you are saying "\\" you are actually saying "\". Similar caveats apply to characters such as '*', '(', etc.

The "a" flag allows the next lookup to be for A records rather than SRV records. Since there is no place for a port specification in the NAPTR record, when the "A" flag is used the specified protocol must be running on its default port.

The URN Sytnax draft defines a canonical form for each URN, which requires %encoding characters outside a limited repertoire. The regular expressions MUST be written to operate on that canonical form. Since international character sets will end up with extensive use of %encoded characters, regular expressions operating on them will be essentially impossible to read or write by hand.

Usage

For the edification of implementers, pseudocode for a client routine using NAPTRs is given below. This code is provided merely as a convience, it does not have any weight as a standard way to process NAPTR records. Also, as is the case with pseudocode, it has never been executed and may contain logical errors. You have been warned.

```
//
// findResolver(URN)
// Given a URN, find a host that can resolve it.
//
findResolver(string URN) {
  // prepend prefix to urn.net
  sprintf(key, "%s.urn.net", extractNS(URN));
  do {
    rewrite_flag = false;
    terminal = false;
    if (key has been seen) {
      quit with a loop detected error
    }
    add key to list of "seens"
    records = lookup(type=NAPTR, key); // get all NAPTR RRs for 'key'

    discard any records with an unknown value in the "flags" field.
    sort NAPTR records by "order" field and "preference" field
        (with "order" being more significant than "preference").
    n_naptrs = number of NAPTR records in response.
    curr_order = records[0].order;
    max_order = records[n_naptrs-1].order;

    // Process current batch of NAPTRs according to "order" field.
    for (j=0; j < n_naptrs && records[j].order <= max_order; j++) {
      if (unknown_flag) // skip this record and go to next one
         continue;
      newkey = rewrite(URN, naptr[j].replacement, naptr[j].regexp);
      if (!newkey) // Skip to next record if the rewrite didn't match
         continue;
      // We did do a rewrite, shrink max_order to current value
      // so that delegation works properly
      max_order = naptr[j].order;
      // Will we know what to do with the protocol and services
      // specified in the NAPTR? If not, try next record.
      if(!isKnownProto(naptr[j].services)) {
        continue;
      }
      if(!isKnownService(naptr[j].services)) {
        continue;
      }

      // At this point we have a successful rewrite and we will know
      // how to speak the protocol and request a known resolution
      // service. Before we do the next lookup, check some
      // optimization possibilities.

      if (strcasecmp(flags, "S")
       || strcasecmp(flags, "P"))
       || strcasecmp(flags, "A")) {
         terminal = true;
         services = naptr[j].services;
         addnl = any SRV and/or A records returned as additional info
                 for naptr[j].
```

```
      }
      key = newkey;
      rewriteflag = true;
      break;
    }
  } while (rewriteflag && !terminal);

  // Did we not find our way to a resolver?
  if (!rewrite_flag) {
     report an error
     return NULL;
  }


  // Leave rest to another protocol?
  if (strcasecmp(flags, "P")) {
     return key as host to talk to;
  }

  // If not, keep plugging
  if (!addnl) { // No SRVs came in as additional info, look them up
    srvs = lookup(type=SRV, key);
  }

  sort SRV records by preference, weight, ...
  foreach (SRV record) { // in order of preference
    try contacting srv[j].target using the protocol and one of the
        resolution service requests from the "services" field of the
        last NAPTR record.
    if (successful)
      return (target, protocol, service);
      // Actually we would probably return a result, but this
      // code was supposed to just tell us a good host to talk to.
  }
  die with an "unable to find a host" error;
}
```

Notes

- A client MUST process multiple NAPTR records in the order specified by the "order" field, it MUST NOT simply use the first record that provides a known protocol and service combination.
- If a record at a particular order matches the URI, but the client doesn't know the specified protocol and service, the client SHOULD continue to examine records that have the same order. The client MUST NOT consider records with a higher value of order. This is necessary to make delegation of portions of the namespace work. The order field is what lets site administrators say "all requests for URIs matching pattern x go to server 1, all others go to server 2". (A match is defined as: 1) The NAPTR provides a replacement domain name or 2) The regular expression matches the URN)
- When multiple RRs have the same "order", the client should use the value of the preference field to select the next NAPTR to consider. However, because of preferred protocols or services, estimates of network distance and bandwidth, etc. clients may use different criteria to sort the records.
- If the lookup after a rewrite fails, clients are strongly encouraged to report a failure, rather than backing up to pursue other rewrite paths.
- When a namespace is to be delegated among a set of resolvers, regexps must be used. Each regexp appears in a separate NAPTR RR. Administrators should do as little delegation as possible, because of limitations on the size of DNS responses.
- Note that SRV RRs impose additional requirements on clients.

Security Considerations

The use of "urn.net" as the registry for URN namespaces is subject to denial of service attacks, as well as other DNS spoofing attacks. The interactions with DNSSEC are currently being studied. It is expected that NAPTR records will be signed with SIG records once the DNSSEC work is deployed.

The rewrite rules make identifiers from other namespaces subject to the same attacks as normal domain names.

Since they have not been easily resolvable before, this may or may not be considered a problem.

Regular expressions should be checked for sanity, not blindly passed to something like PERL.

This document has discussed a way of locating a resolver, but has not discussed any detail of how the communication with the resolver takes place. There are significant security considerations attached to the communication with a resolver. Those considerations are outside the scope of this document, and must be addressed by the specifications for particular resolver communication protocols.