Network Working Group
Request for Comments: 2016
Category: Experimental

L. Daigle
P. Deutsch
B. Heelan
C. Alpaugh
M. Maclachlan
Bunyip Information Systems, Inc.

October 1996

Uniform Resource Agents (URAs)

Status of this Memo

Abstract

This paper presents an experimental architecture for an agent system that provides sophisticated Internet information access and management. Not a generalized architecture for active objects that roam the Internet, these agents are modeled as extensions of existing pieces of the Internet information infrastructure. This experimental agent technology focuses on the necessary information structures to encapsulate Internet activities into objects that can be activated, transformed, and combined into larger structured activities.

Acknowledgements

Several people have shared thoughts and viewpoints that have helped shape the thinking behind this work over the past few years. We'd like to thank, in particular, Chris Weider, Patrik Faltstrom, Michael Mealling, Alan Emtage, and the participants in the IETF URI Working Group for many thought-provoking discussions.

Sima Newell provided insightful comments on the document -- thanks to her it is much more readable!

Introduction

This document outlines an experimental agent system architecture that was designed for the purpose of addressing high-level Internet activities through encapsulation of protocol-specific actions. Originally presented to the Uniform Resource Identifier (URI) working group at the IETF, this technology was seen as taking a step beyond resource location and resource naming. By providing a structured mechanism for abstracting characteristics of desired information and

distancing the necessary access incantations from the client, the notion of a Uniform Resource Agent (URA) was created.

The evolution of Internet information systems has been characterized by building upon successive layers of encapsulated technologies. Machine address numbers were devised, and then encapsulated in advertised machine names, which has allowed the evolution of the Domain Name System (DNS) [RFC1034, RFC1035]. Protocols were developed for accessing Internet resources of various descriptions, and then uniform mechanisms for specifying resource locations, standardized across protocol types, were developed (URLs) [RFC1738]. Each layer of Internet information primitives has served as the building blocks for the next level of abstraction and sophistication of information access, location, discovery and management.

The work described in this paper is an experimental system designed to take another step in encapsulation. While TCP/IP protocols for routing, addressing, etc, have permitted the connection and accessibility of a plethora of information services on the Internet, these must yet be considered a diverse collection of heterogeneous resources. The World Wide Web effort is the most successful to date in attempting to knit these resources into a cohesive whole. However, the activity best-supported by this structure is (human) browsing of these resources as documents. The URA initiative explores the possibility of specifying an activity with the same kind of precision accorded to resource naming and identification. By focusing on activities, and not actions, URAs encapsulate resource access mechanisms based on commonality of information content, not protocol similarity.

An invoker -- human or otherwise -- may delegate an entire set of tasks to a fully-instantiated URA. The nature of the tasks is completely specified by the agent, because it encapsulates knowledge about relevant Internet resources and the information required in order to access them. In this way, URAs insulate invokers from the details of Internet protocols while allowing them to carry out high-level Internet activities (such as searching a set of web pages and news groups relevant to a given topic). Also, by formally specifying a high-level Internet activity in an agent, the same activity can be repeated at a later date by the same invoker, someone else or even another agent. Moreover, the agent object may easily be modified to carry out another related task.

More detail describing the underlying philosophy of this particular approach can be found in [IIAW95].

Examples

As a very simple example, consider the client task of subscribing to a mailing list. There are many mechanisms for providing users with information necessary to complete a subscription. Currently, all applications which provide the ability to subscribe to mailing lists must contain protocol-aware code to carry out the task once the requisite personal data has been solicited from the user. Furthermore, any application program that embeds the ability to subscribe in its code necessarily limits the set of mailing lists to which a client can subscribe (i.e, to those types foreseen by the software's creators). If, instead, there is an agent to which this task can be delegated, all applications can make use of the agent, and that agent becomes responsible for carrying out the necessary interactions to complete the subscription. Furthermore, that agent may be a client to other agents which can supply particular information about how to subscribe to new types of mail servers, etc. URAs have been explored as an agent technology to address just these types of issues.

Relationship to Other Internet Agents

A number of Internet-aware agent and transportable code systems have become popular -- Java [JAVA], TCL [TCL] and Safe-TCL, Telescript [TELE], and the TACOMA system [TACOMA], to name a few of them. To understand the scope of the problem that URAs tackle, it is helpful to understand how these systems differ from the URA approach. Some of these agent systems, like Java, focus on providing mechanisms for creating and distributing (inter)active documents in the World Wide Web. Others, like TACOMA, have more general intentions of providing environments for mobile, interacting processes.

While each of these systems makes its individual contribution to solving the transportation, communication, and security issues normally associated with agent systems, they yield more objects that exist within the Internet information space. That is, while they may permit individual users to have a more sophisticated interaction with a particular information resource, they do not address the more general Internet problems of naming, identifying, locating resources, and locating the same or similar resources again at a later date. It is this set of problems that URAs specifically set out to address.

In order to create these URA objects that encapsulate a set of Internet activities, it is necessary to specify their operating environment and design structure. Together, these form an experimental architecture for URAs, which can be evaluated in a preliminary way through a prototype implementation. The remainder of this paper describes such an experimental architecture, and outlines

a prototype application built to test the concepts involved in the creation and execution of URAs.

The Experimental Architecture

The main goal in designing the URA architecture was to provide a mechanism for separating client need descriptions from the specifications of mechanisms for satisfying those needs. For example, from the client's perspective, the need to find MIDI music files is quite distinct from the particular Internet resource actions that might be necessary to find them at a given point in time. This one need might be best met by integrating information from several very different sources. Also, the client may have the same need on a different day, but there may be new or different resources to call on to satisfy it.

A further goal was to provide very structured specifications of the Internet actions carried out by a particular URA. By making the structure of an action explicit, it becomes possible to operate on portions of an agent structure without requiring an understanding of the complete semantics of its activity.

At the centre of the URA architecture is the concept of a (persistent) specification of an activity. For purposes that should become clear as the expected usage of URAs is described in more detail, we choose to support this concept with the following requirements of the architecture:

- there is a formalized environment in which these specifications are examined and executed and otherwise manipulated. This is referred to as a URAgency.

- the activity specifications are modular, and independent of a given URAgency environment. Thus, they exist as object constructs that can be shared amongst URAgencies. There is a standardized _virtual_ structure of these URA objects, although different types may exist, with different underlying implementations.

Basic URAgency Requirements

In the most abstract sense, a URAgency is a software system that manipulates URA objects. In the terminology of objects, a URAgency identifies the types of URAs it handles, and is responsible for applying methods to objects of those types. For the purposes of this experimental work, the only methods it is required to support are those to get information about a given URA, and to execute a URA.

The expected result of applying the "get information" method to a URA is a description of some or all of the URA following the standardized virtual structure of a URA object, outlined below.

The appropriate way to "execute" a URA is to supply information for the individual URA data segments (in effect, to permit the creation of an instance of a virtual object), or to identify a URA instance. Again, the information is to be supplied in accordance with the virtual structure below.

A URAgency claiming to handle a particular type of URA must have the ability to map the implementation structure of that type of URA into and out of the standard virtual URA structure. The URAgency must also know how to activate the URA, and it must satisfy any runtime dependencies for that type of URA.

For example, a URA type may consist of a Pascal program binary which, when run with particular command line arguments, yields information in the standard URA object structure. Activating this type of URA might consist of executing the Pascal binary with an input file containing all the necessary data segments. A URAgency claiming to handle this sort of URA type must first be able to provide an environment to execute the Pascal binary (for whatever platform it was compiled), and also be able to interact with the Pascal binary according to these conventions to get information about the URA, or execute it.

As an alternative example, a URA type may consist of a script in some interpreted language, with the URA object structure embedded as data structures within the script. A URAgency handling this type of URA might have to be able to parse the script to pull out the standard URA object structure, and provide the script language interpreter for the purposes of executing the URA.

URA Object Structure

In order to capture the necessary information for carrying out the type of Internet activity described in the introductory paragraphs of this document, six basic (virtual) components of a URA object have been identified. Any implementation of a URA type is expected to be able to conform to this structure within the context of a URAgency.

The six basic components of a URA object are:

URA HEADER:
: Identification of the URA object, including a URA name, type and abstract, creator name, and the resources required by the URA.

ACTIVATION DATA:
: Specification of the data elements required to carry out the URA activity. For example, in the case of an Internet search for "people", this could include specification of fields for person name, organization, e-mail address.

TARGETS:
: Specification of the URL/URN's to be accessed to carry out the activity. Note that, until URN's are in common use, the ability to adjust URLs will be necessary. A key issue for URAs is the ability to transport them and activate them far from the creator's originating site. This may have implications in terms of accessibility of resource sites. For example, a software search created in Canada will likely access a Canadian Archie server, and North American ftp sites. However, an invoker in Australia should not be obliged to edit the URA object in order to render it relevant in Australia. The creator, then, can use this section to specify the expected type of service, with variables for the parts that can be modified in context (e.g., the host name for an Archie server, or a mirror ftp site).

EXPERIENCE INFORMATION:
: Specification of data elements that are not strictly involved in conversing with the targets in order to carry out the agent's activity. This space can be used to store information from one invocation of a URA instance to the next. This kind of information could include date of last execution, or URLs of resources located on a previous invocation of the agent.

ACTIVITY:
: If URAs were strictly data objects, specifying required data and URL/URN's would suffice to capture the essence of the composite net interaction. However, the variability of Internet resource accesses and the scope of what URAs could accomplish in the net environment seem to suggest the need to give the creator some means of organizing the instantiation of the component URL/URN's. Thus, the body of the URA should contain a scripting mechanism that minimally allows conditional instantiation of individual URL/URN's. These conditions could be based on which (content) data elements the user provided, or accessibility of one URL/URN, etc. It also provides a mechanism for suggesting scheduling of URL/URN instantiation.

The activity is specified by a script or program in a language specified by the URA type, or by the URA header information. All the required activation data, targets, and experience information are referenced by their specification names.

RESPONSE FILTER:

The main purpose of the ACTIVITY module is to specify the steps necessary to take the ACTIVATION DATA, contact the TARGETS, and collect responses from those services. The purpose of the RESPONSE FILTER module is to transform those responses into the result of the URA invocation. This transformation may be along the lines of reformatting some text, or it may be a more elaborate interpretation such as a relevance rating for a retrieved HTML page.

The response filter is specified by a script or program in a language specified by the URA type, or by the URA header information. All the required activation data, targets, and experience information are referenced by their specification names.

See Appendix 1 for a more detailed description of the components of a URA. Appendix 2 contains a sample virtual URA structure.

The Architecture in Action

Having introduced the required capabilities of the URAgency and virtual structure of URA objects, it is now time to elaborate on the tasks and interactions that are best supported by URAs.

URAs are constructed by identifying net-based resources of interest (targets) to carry out a particular task. The activation data component of a URA is the author’s mechanism for specifying (to the invoker) the elements of information that are required for successful execution . An invoker creates an instance of a URA object by providing data that is consistent with, or fills in, this template. Such an instance encapsulates everything that the agent "needs to know" in order to contact the specified target(s), make a request of the resource ("get", "search", etc.) and return a result to the invoker. This encapsulation is a sophisticated identification of the task results.

For example, in the case of a mailing list subscription URA, the creator will identify the target URL for a resource that handles list subscription (e.g., an HTML form), and specify the data required by that resource (such as user name, user mail address, and mailing list identifier). When an invoker provides that information and instantiates the URA, the resulting object completely encapsulates

all that is needed in order to subscribe the user -- the subscription result is identified.

URAs are manipulated through the application of methods. This, in turn , is governed by the URAgency with which the invoker is interacting. However, because the virtual structure of URAs is represented consistently across URA types and URAgencies, a URAgency can act as one of the targets of a URA. Since methods can be applied to URAs remotely, URAs can act as invokers of URAs. This can yield a complex structure of task modules.

For example, a URA designed to carry out a generalized search of book-selling resources might make use of individual URAs tailored to each resource. Thus, the top-level URA becomes the orchestrating URA for access to a number of disparate resources, while being insulated from the minute details of accessing those resources.

A Prototype Implementation

The experimental work with URAs includes a prototype implementation of URA objects. These are written in the Tcl scripting language. A sample prototype Tcl URA can be found in Appendix 3.

The URAgency that was created to handle these URAs is part of the Silk Desktop Internet Resource Discovery tool. Silk provides a graphical user interface environment that allows the user to access and search for Internet information without having to know where to look or how to look. Silk presents a list of the available URAs to carry out these activities (e.g., "search for tech reports" or "hotlist"). For each activity, the user is prompted for the activation data, and Silk’s URAgency executes the URA. The Silk software also supports the creation and maintenance of URA object instances. Users can add new URAs by creating new Tcl scripts (per the guidelines in the "URA Writer’s Guide", available with the Silk software. See [SILK]). The Silk graphical interface hides some of the mechanics of the underlying URAgency. A more directly-accessible version of this URAgency will become available.

Conclusions

This work was originally conceived as an extension to the family of Uniform Resource Identifiers (URIs): Uniform Resource Locators (URLs), Uniform Resource Characteristics (URCs), and the proposed Uniform Resource Names (URNs). The approach of formalizing the characteristics of an information task in a standardized object structure is seen as a means of identifying a class of resources, and contributes to the level of abstraction with which users can refer to Internet resources.

Although still in its experimental stages, this work has already evoked interest and shown promise in the area of providing mechanisms for building more advanced tools to interact with the Internet at a more sophisticated level than just browsing web pages.

One of the major difficulties that has been faced in developing a collection of URAs is the brittleness induced by interacting with services that are primarily geared towards human-users. Small changes in output formats that are easily discernible by the human eye can be entirely disruptive to a software client that must apply a parsing and interpretation mechanism based on placement of cues in the text. This problem is certainly not unique to URAs -- any software acting upon results from such a service is affected. Perhaps there is the need for an evolution of "service entrances" to information servers on the Internet -- mechanisms for getting "just the facts" from an information server. Of course, one way to provide such access is for the service provider to develop and distribute a URA that interacts with the service. When the service's interface changes, the service provider will be moved to update the URA that was built to access it reliably.

Work will continue to develop new types of URAs, as well as other URAgencies. This will necessitate the creation of URAgency interaction standards -- the "common virtual URA object structure" is the first step towards defining a lingua franca among URAs of disparate types and intention.

References

[IIAW95] Leslie L. Daigle, Peter Deutsch, "Agents for Internet Information Clients", CIKM'95 Intelligent Information Agents Workshop, December 1995.
Available from
<http://www.bunyip.com/products/silk/silktree/uratree/iiaw95.ps>

[JAVA] "The Java Language: A White Paper" Available from
<http://java.sun.com/1.0alpha2/doc/overview/java/index.html>

[RFC1034] Mockapetris, P., "Domain Names - Concepts and Facilities", STD 13, RFC 1034, November 1987.

[RFC1035] Mockapetris, P., "Domain Names - Implementation and Specification", STD 13, RFC 1035, November 1987.

[RFC1738] T. Berners-Lee, L. Masinter, M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[SILK] Bunyip's Silk project homepage:
<http://www.bunyip.com/products/silk/>

[SILKURA] Silk URA information:
<http://www.bunyip.com/products/silk/silktree/uraintro.html>

[TACOMA] Johansen, D. van Renesse, R. Schneider, F. B., "An Introduction to the TACOMA Distributed System", Technical Report 95-23, Department of Computer Science, University of Tromso, Norway, June 1995.

[TCL] Ousterhout, J. K. "Tcl and the Tk Toolkit", Addison Wesley, 1994.

[TELE] White, J. E., "Telescript Technology: The Foundation for the Electronic Marketplace", General Magic White Paper, General Magic Inc., 1994.

Authors' Addresses

Leslie Daigle
Peter Deutsch
Bill Heelan
Chris Alpaugh
Mary Maclachlan

Bunyip Information Systems, Inc.
310 St. Catherine St. West
Suite 300
Montreal, Quebec, CANADA
H2X 2A1

Phone: (514) 875-8611
EMail: ura-bunyip@bunyip.com

Appendix 1 -- Virtual URA Structure

This appendix contains a BNF-style description of the expected virtual structure of a URA object. This "virtual structure" acts as the canonical representation of the information encapsulated in a given URA. It is expected that more information may optionally be contained in the elements of the components -- the elements listed here are offered as the "minimum" or "standard" set.

N.B.:

```
        []-delimited items are optional
        %% denotes a comment
        \0 represents the empty string
        |  is "or"
        {} are literal characters
```

This form is used for convenience and clarity of expression -- whitespace and ordering of individual elements are not considered significant.

```
<VIRTUAL_URA> := {<virtual-ura-structure>}

<virtual-ura-structure> := { URAHDR <ura-header> }
                           { ACTDATA <activation-data> }
                           { TARG <targets> }
                           { EXPINFO <experience information> }
                           { ACTSPEC <activity> }
                           { RESPFILT <response filter> }

<ura-header> := { name <ura-name> }
                { author <ura-author> }
                { version <ura-version> }
                [ { lang <lang-dependencies> } ]
                [ { parent <parent-of-instance> } ]



<activation-data> := <act-data-element><activation-data> | \0

<act-data-element> := {
                       { name <data-elt-name> }
                       { response <data-elt-value> }
                       { prompt <data-elt-prompt> }
                       [ { required <boolean> } ]
                       [ { default <data-default-val> } ]
                      }

<targets> := <target-service><targets> | \0
```

```
<target-service> := {
                     { name <targ-url> }
                     { protocol <url-protocol> }
                     { url <url-spec> }
                     [ { <url-type-specific-data> } ]
                    }

<url-spec> :=  <complete-url> | <url-constructor>

<complete-url> := %% a complete, valid URL string
                    (e.g., http://www.bunyip.com/)

<url-constructor> := {
                      { scheme <url-scheme-spec> }
                      { host <url-host-spec> }
                      [ { port <url-port-spec> } ]
                      { selector <url-selector-spec> }
                     }

<url-scheme-spec> := {
                      { name <scheme-name> }
                      { response <scheme-value> }
                      { prompt <scheme-prompt> }
                     }
<url-host-spec> := {
                      { name <host-name> }
                      { response <host-value> }
                      { prompt <host-prompt> }
                    }
<url-port-spec> := {
                      { name <port-name> }
                      { response <port-value> }
                      { prompt <port-prompt> }
                    }
<url-selector-spec> := {
                      { name <selector-name> }
                      { response <selector-value> }
                      { prompt <selector-prompt> }
                       }


<experience information> := {
                             { name <data-elt-name> }
                             { response <data-elt-value> }
                            }

<activity> :=  <compound-string>
```

```
<response filter> := <compound-string>


%% Without requiring more detail...

<compound-string> := <string>\n<compound-string> | \0
<boolean> := 0 | 1
<ura-name> := <string>
<ura-author> := <string>
<ura-version> := <string>
<lang-dependencies> := <string>
<parent-of-instance> := <string>
<data-elt-name> := <string>
<data-elt-value> := <string>
<data-elt-prompt> := <string>
<data-elt-default> := <string>
<data-default-val> := <string>
<targ-url> := <string>
<url-protocol> := http-get | http-post | ...
<url-type-specific-data> := <string>
<scheme-name> := <string>
<scheme-value> := <string>
<scheme-prompt> := <string>
<host-name> := <string>
<host-value> := <string>
<host-prompt> := <string>
<port-name> := <string>
<port-value> := <string>
<port-prompt> := <string>
<url-selector-name> := <string>
<url-selector-value> := <string>
<url-selector-prompt> := <string>
```

Appendix 2 -- Sample Virtual URA Representation

A valid virtual representation of a Silk Tcl URA is presented below. The actual URA from which it was drawn is given in Appendix 3.

```
{
   {URAHDR
      {name {DejaNews Search}}
      {author {Leslie Daigle}}
      {version  {1.0}}
   }

   {ACTDATA
      {name        {Topic Keywords}}
```

```
      {prompt      {Topic Keywords}}
      {response    {}}
   }

   {EXPINFO
      {name        {Comments}}
      {prompt      {Comments}}
      {response    {}}
   }

   {ACTSPEC
       {proc mapResponsesToDejanews {} {
           set resp ""
           if {[uraAreResponsesSet {Topic Keywords}]} {
             lappend resp [list query [uraGetSpecResponse {
             Topic Keywords}]]
           }

           return $resp

         }
      proc uraRun {} {
        global errorInfo

        foreach serv [uraListOfServices] {
          set u [uraGetServiceURL $serv]

          switch -- $serv {
            dejanews {
              if [catch {
                set query [mapResponsesToDejanews]
                if {$query != {}} {
                    set result [uraHTTPPostSearch $u $query]
                    if {$result != ""} {
                      set list [dejanews_uraHTTPPostCanonicalize
                      $result]
                      puts $list
                    }
                  }
              }] {
                puts stderr $errorInfo
              }
            }

            default {
              # can't handle other searches, yet.
            } } } }
      }
```

```
   }

   {RESPFILT
      {
       proc dejanews_uraHTTPPostCanonicalize {htmlRes} {

         set result {}
         set lines {}
         set clause {}
         set garb1 ""
         set garb2 ""


         # Get the body of the result page -- throw away leading and
         # trailing URLs

         regexp {([^<PRE>]*)<PRE>(.*)</PRE>.*}
                 $htmlRes garb1 garb2 mainres

         set lines [split $mainres "\n"]

         foreach clause $lines {

           if [regexp
           {<DT>.*(..\/..).*<A HREF="([^"]*)">([^<]*)</A>.*<B>([^<]*).*}
                $clause garb1 dt relurl desc grp] {

             lappend r [list HEADLINE [format "%s    (%s, %s)"
                        [string trim $desc] \
                    [string trim  $grp] $dt]]
             lappend r [list URL [format
                        "http://www.dejanews.com/cgi-bin/%s" $relurl]]
             lappend r [list TYPE "text/plain"]

             lappend result $r
           }
         }
         return $result
       }
      }


   }

}
```

Appendix 3 -- Sample Silk Tcl URA

The following is a valid Silk Tcl URA. For more information on the implementation and structure of Silk-specific URAs, see the "URA Writers Guide" that accompanies the distribution of the Silk software (available from <http://www.bunyip.com/products/silk>).

```
# ------------------------------------------------------------------------
#
#                              URA initialization
#
# ------------------------------------------------------------------------

#
# Initialize the URA, its search specs and searchable services.
#

# URA init.

set uraDebug 1

uraInit {
  {name {DejaNews Search}}
  {author {Leslie Daigle}}
  {version {1.0}}
  {description "This URA will search for UseNet News articles."}
  {help "This is help on UseNet News search script."}
}

#
# bug: handling of choices/labels is kind of gross.
#

# Search spec. init.

foreach item {
  {
    {name        {Topic Keywords}}
    {field       Topic}
    {tag         STRING}
    {description {Keywords to search for in news articles}}
    {prompt      {Topic Keywords}}
    {help        {Symbols to look up, separated by spaces.}}
    {type        STRING}
    {subtype     {}}
    {allowed     .*}
    {numvals     1}
    {required    0}
```

```
    {response    {}}
    {respset     0}
  }
} {
  uraSearchSpecInit $item
}

uraAnnotationInit {
  {help        {Enter comments to store with an instance}}
  {numvals     1}
  {subtype     {}}
  {response    {}}
  {name        Comments}
  {required    0}
  {class       ANNOTATION}
  {type        TEXT}
  {description {General comments about this URA.}}
  {respset     1}
  {prompt      Comments}
  {field       {}}
  {allowed     .*}
}

uraResultInit {
  {name {Related Pages}}
  {contents { {
    {HEADLINE {The DejaNews UseNet search service}}
    {TYPE text/plain}
    {URL http://www.dejanews.com}
} }}
}


foreach item {
  {
    {name dejanews}
    {protocol http-post}
    {url http://marge.dejanews.com/cgi-bin/nph-dnquery}
  }
} {
  uraServicesInit $item
}


proc dejanews_uraHTTPPostCanonicalize {htmlRes} {

  set result {}
  set lines {}
```

```
  set clause {}
  set garb1 ""
  set garb2 ""


  # Get the body of the result page
  # -- throw away leading and trailing URLs
  regexp {([^<PRE>]*)<PRE>(.*)</PRE>.*} $htmlRes garb1 garb2 mainres

  set lines [split $mainres "\n"]

  foreach clause $lines {

    uraDebugPuts stderr [format "Line: %s" $clause]

    if [regexp
    {<DT>.*(..\/..).*<A HREF="([^"]*)">([^<]*)</A>.*<B>([^<]*).*} \
         $clause garb1 dt relurl desc grp] {
      uraDebugPuts stderr [format
                           "Date: %s Rel URL: %s Desc: %s Group: %s"
                           $dt $relurl $desc $grp]

      lappend r [list HEADLINE [format "%s    (%s, %s)"
                 [string trim $desc] \
            [string trim  $grp] $dt]]
      lappend r [list URL [format
                 "http://www.dejanews.com/cgi-bin/%s" $relurl]]
      lappend r [list TYPE "text/plain"]

      lappend result $r
    }
  }
  return $result

}


# ---------------------------------------------------------------------
#
#                             Mapping procedures
#
# ---------------------------------------------------------------------

#
# There is one procedure, for each searchable service, to map the search
# spec responses to a form suitable for inclusion into a search URL (or
# whatever form the particular query procedure accepts).
#
```

```
#
#
proc mapResponsesToDejanews {} {
  set resp ""
  if {[uraAreResponsesSet {Topic Keywords}]} {
    lappend resp [list query [uraGetSpecResponse {Topic Keywords}]]
  }

  return $resp

}


#
# bug: need better error reporting
# (i.e. which searches didn't work and why, etc.)
#
proc uraRun {} {
  global errorInfo

  foreach serv [uraListOfServices] {
    set u [uraGetServiceURL $serv]

    switch -- $serv {
      dejanews {
        if [catch {
          set query [mapResponsesToDejanews]
          uraDebugPuts stderr [format "%s: query is '%s'."
          $serv $query]
          if {$query != {}} {
              set result [uraHTTPPostSearch $u $query]
              if {$result != ""} {
                uraDebugPuts stderr [format "%s: result is '%s'."
                $serv $result]
                set list [dejanews_uraHTTPPostCanonicalize $result]
                uraDebugPuts stderr [format "%s: list is '%s'."
                $serv $list]
                puts $list
              }
            }
        }] {
          puts stderr $errorInfo
        }
      }

      default {
        # can't handle other searches, yet.
      }
```

```
    }
  }
}
```