                         Resource Transponders

Status of this Memo

Abstract

   Although a number of systems have been created in the last several
   years to provide resource location and navigation on the Internet,
   the information contained in these systems must be maintained and
   updated by hand. This paper describes an automatic mechanism, the
   resource transponder, for maintaining resource location information.

Author's Note:

   This document is being circulated as sort of a research paper;
   consequently there are no protocol specifications or anything of the
   sort. I hope that we can go from here and actually design them if
   there's consensus that they are potentially useful. Once we have some
   idea of the required functionality, we can then go out and
   standardize them.

Disclaimer

   This paper represents only the opinions of the author; it does not
   represent the consensus of the IIIR Working Group, although it is
   recognized by them as one legitimate approach to a solution of the
   problem.

1. Introduction

   In the past few years, we've seen the invention and growth of a
   number of information location systems on the Internet, e.g., archie,
   Gopher, and WAIS. However, as these systems have become widely
   deployed, a number of maintenance and security problems have arisen
   with them. Some of the major ones:

   1) Out of necessity, most of these systems contain pointers to the
      desired resources rather than the resources themselves. Therefore,
      if a resource becomes obsolete, is modified, or is moved, the

location system must be updated by hand. Some systems (archie in
particular) proactively create updated indexes by contacting every
resource on a certain time schedule (every 30 days or so) but this
means that the system can be up to 30 days out of date, and this
process can be highly inefficient depending on the percentage of
information that has changed.

2) Conversely, anyone who maintains a resource that they wish indexed
   must keep track of every directory which contains a pointer to
   that resource, so that if it is modified, all the directories can
   be updated. This obviously is an optimistic scenario.

3) Many organizations which have installed these systems do not have
   the the available resources or expertise to maintain the
   information in the systems. Thus we have long periods where the
   information drifts, then a short period when the information is
   updated again.

4) Even though these systems are almost always out of date today,
   this problem will become increasingly harder for humans to manage
   by hand as everyone on the net becomes their own publisher. Also,
   as the net speeds up and people rely more and more on accurate
   information, human-induced delays in updates of these systems will
   become increasingly intolerable.

5) Most, if not all, of these systems provide no security whatsoever;
   if a pointer to a resource appears in a locator system, then it is
   assumed to be meant for public consumption. There are many
   potential information providers who would like to use publicly
   deployed information systems to publish to a very selected
   clientele, and do not wish to allow the whole net access to their
   resources.

2. Requirements for a Solution

   There are several objectives which must be met by any proposed
   solution to these problems:

   1) We need to decrease the personnel resources needed for indexing
      and pointer maintenance.

   2) We need to increase the reliability and accuracy of the
      information held in resource location systems.

   3) We need to provide some mechanisms for security, particularly by
      mediating access to the resources.

   4) We need to make it easy for non-experts, such as librarians,
      archivists, and database maintainers, to announce their new
      resources to the various resource location services.

   Many of these problems can be solved by a 'resource transponder'
   mechanism.

3. Resource Transponders

   The resource transponder system works by adding two new layers to
   every resource: metainformation and an agent to update a resource
   location system (RLS) with that metainformation. The metainformation
   layer is physically attached to every resource, so that when the
   resource is moved or altered, the metainformation is immediately
   available to update the RLS. The agent layer may also be attached to
   the resource or may not be; the implications of both of these options
   are discussed in detail below.

3.1 Metainformation

   The metainformation layer of a given resource contains any
   information which might be required to create a pointer to this
   resource, and any information which may be useful for indicating how
   to catalog or index the resource. For example, the metainformation
   layer of a text document might contain such things as the Uniform
   Resource Name (URN) of the document (this is sort of a ISBN number
   for electronic resources), the title of the document, a Uniform
   Resource Locator (URL) for the document (this is a combination net
   address and access method indicator, used for retrieval), the size of
   the document, etc. Thus the metainformation layer contains data about
   the resource to which it is attached.

   This metainformation is expected to be modifiable. For example, the
   metainformation layer may contain a history of where this particular
   copy of a resource has been. Let's say that a resource/transponder
   pair has been moved. When it gets to its new location, the agent can
   then attempt to contact the resource at its old location to determine
   whether the resource is still there (in which case the agent will
   simply cause the new location to be added to the RLS) or whether the
   resource is not there (in which case the agent can tell the RLS to
   add the current pointer and delete the old one).

   A number of other possibilities for the contents of the
   metainformation level are contained in section 4.1.

3.2 Agents

   The agent layer of a given resource contains an executable program
   which is responsible for reading the metainformation attached to the
   resource and using that information to update a RLS. It is also
   responsible for updating the metainformation where necessary and for
   running any indexing programs required by the RLS it is attempting to
   update.

   When the tools required to build agents are constructed and deployed,
   the author expects the agents to begin mediating access to the
   resource, particularly for agents attached to resources which are not
   currently considered active processes, such as text files and
   digitized images. In this futuristic model, someone wishing to read
   a given document would have to first negotiate access to the data
   with the agent; the agent would then be responsible for delivering
   the data to the client. However, it is expected that this type of
   agent will not be widely deployed for some time.

   Different ways of implementing agents are discussed in section 4.2.

4. Models for implementations of resource transponders

   4.1. Models for implementations of the metainformation layer

   The metainformation layer can be impelemented in a number of ways,
   depending on the resource with which it is associated. For an
   'active' resource, such as an on-line catalog or a mail-based
   service, the metainformation can be stored in a file with a well-
   known name in the software distribution. Alternatively, the
   metainformation could be stored as a record in the data which the
   resource serves. For a text document, the metainformation could be
   stored as the first or last N bytes of the document (which would
   break a number of editors and file display techniques, but would
   guarantee that the metainformation is moved with the resource), or
   perhaps as a file with a logically associated name (paper2.meta
   associated with paper2.txt, for example). The problem with this
   second approach is that the user must know that they have to move the
   metainformation with the file itself, or things will start breaking.
   If an agent is explicitly attached to the resource, the agent could
   contain the metainformation internally.

   In any case, the resource transponder system must be able to
   guarantee that the metainformation is moved when the resource is
   moved.

4.2 Models for implementations of the agents

   The agent layer can also be implemented in a number of ways,
   depending on such things as system loads, desired sizes of resources,
   multitasking capabilities, etc.

   The easiest and for many unitasking systems the cleanest way of
   implementing an agent is to have one agent per computer. Then when a
   resource is moved onto that computer, the agent is explicitly
   activated and notified where the new resource is. For example, let's
   say that someone wishes to download a copy of a resource and then let
   the RLS know that that resource is available for public consumption.
   She would download the resource and then run the agent, which would
   then notify the RLS and update the metainformation attached to the
   resource. This model could also be used to track files on a LAN, or
   to provide local location services with no need to run a larger RLS.

   Another model for implementation of the agent is to have one agent
   per resource. In this model, the agent would be moved along with the
   resource and the metainformation. The agent could be implemented in a
   file which would be associated with the resource; in that case the
   agent would have to be explicitly activated when the resource was
   moved. Alternatively, the agent/metainformation/resource system could
   be implemented as one system, or in one file. In this case, the agent
   itself would always be active, and would be responsible for mediating
   access to the resource. When one did a 'telnet' to a resource with
   an active agent, the agent would accept the telnet connection and be
   responsible for providing security and translation for the data. This
   could provide great security for resources while still allowing
   pointers to them to be placed in public RLS's; the data in the
   resource could be encrypted, with the agent responsible for
   decrypting it.

5. Security Considerations

   Security issues are discussed throughout this memo.

6. Author's Address

   Chris Weider
   Bunyip Information Systems, Inc.
   2001 S. Huron Parkway, #12
   Ann Arbor, MI 48104
   USA

   Phone: +1 313-971-2223
   Fax: +1 313-971-2223
   EMail: clw@bunyip.com