Location-Independent Naming for Virtual Distributed Software Repositories*

Shirley Browne[†] Jack Dongarra, Stan Green, Keith Moore
Theresa Pepin, Tom Rowan, and Reed Wade
University of Tennessee
Eric Grosse
AT&T Bell Laboratories

Abstract

A location-independent naming system for network resources has been designed to facilitate organization and description of software components accessible through a virtual distributed repository. This naming system enables easy and efficient searching and retrieval, and it addresses many of the consistency, authenticity, and integrity issues involved with distributed software repositories by providing mechanisms for grouping resources and for authenticity and integrity checking. This paper details the design of the naming system, describes a prototype implementation of some of the capabilities, and describes how the system fits into the development of the National HPCC Software Exchange, a virtual software repository that has the goal of providing access to reusable software components for high-performance computing.

1 Introduction

Well-maintained software repositories are central to software reuse because they make high-quality software widely available and easily accessible. One such repository is Netlib[1], a collection of high-quality publicly available mathematical software[6, 4]. Netlib, in operation since 1985, currently processes over 300,000 requests a day. Netlib is serving as a prototype for development of the National HPCC Software Exchange (NHSE)[2], which has the goal of encompassing all High Performance Computing and Communications (HPCC) software repositories and of promoting reuse of software components developed by Grand Challenge and other scientific computing researchers [5]. Other network-accessible software repositories include ASSET[3], CARDS[4], DSRS[5], ELSA[6], the GAMS Virtual Software Repository[7], and STARS[8]. ASSET, CARDS, DSRS, and ELSA are participating in an interoperability experiment that allows a user of any one of these repositories to access software exported from the other repositories.

The software reuse marketplace is expanding in at least two dimensions. One dimension is the expansion from intra-organizational reuse to inter-organizational reuse. For example, various federal agencies have established their own internal software reuse programs. Several efforts are now underway to promote reuse of software across agencies. Similarly, companies are becoming interested in accessing software produced by academic and government research groups. Another dimension of expansion is from reuse within a particular application domain to interdisciplinary reuse. Reuse of software from other disciplines is being fostered, for example, by efforts to solve interdisciplinary Grand Challenge problems. Solution of such problems will require collaboration by scientists from different disciplines, as well as sharing of software produced by application and computer scientists.

Another recent development that affects the software reuse marketplace is the growth of the World Wide Web (WWW), together with the ease with which individuals may make resources available on a WWW server. A contributor need only make the files composing an resource available on a file server and make available a descriptive HTML file containing pointers to the resource files.

Growth in the popularity of the Internet and the World Wide Web, as well as the wide availability of WWW client and server software, has accelerated the shift from centrally maintained software repositories to virtual, distributed repositories. For example, the GAMS Repository, once a central repository, is now a virtual repository that catalogs software maintained by other repositories [2]. Similarly, the NHSE will provide a uniform interface to a virtual HPCC software repository that will be built on top of a distributed set of discipline-oriented repositories[5], as shown in Figure 1.

The main advantage of distributing a repository is to

*The work described in this paper is sponsored by NASA under Grant No. NAG 5-2736, by the National Science Foundation under Grant No. ASC-9103853, and by AT&T Bell Laboratories.

†Author to whom correspondence should be directed. 107 Ayres Hall, Computer Science Department, University of Tennessee, Knoxville, TN 37996-1301, (615) 974-5886, browne@cs.utk.edu

[1] Accessible from a World Wide Web browser at **http://www.netlib.org/**

[2] Accessible at **http://www.netlib.org/nse/**

[3] Accessible at **http://source.asset.com/**

[4] Accessible at **http://dealer.cards.com/**

[5] Accessible at **http://ssed1.ims.disa.mil/srp/dsrspage.html**

[6] Accessible at **http://rbse.mountain.net/ELSA/elsa_lob.html**

[7] Accessible at **http://gams.nist.gov/**

[8] Accessible at **http://www.stars.ballston.paramax.com/index.html**

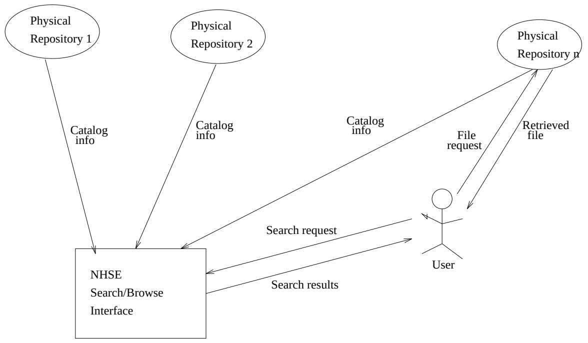

Figure 1: Virtual Repository Architecture

allow the software to be maintained by those in the best position to keep it up-to-date. Also, copies of popular software packages may be mirrored by a number of sites to increase availability (e.g., if one site is unreachable, the software may be retrieved from a different site) and to prevent bottlenecks.

Despite the benefits, distributed maintenance and mirroring of software poses the following challenges.

- Maintaining the quality of software and of indexing information and presenting a uniform searching and browsing interface become much more difficult.
- The WWW mechanism of specifying a file by its Uniform Resource Locator (URL) is inadequate for ensuring the consistency and currency of mirrored copies, as a URL for an independently mirrored copy of a software package may point to an out-of-date copy and give no indication that it is not up-to-date. Furthermore, mirror copies of a file cannot be located from a URL reference, since each copy has a different URL.
- Consistency between a set of files that are meant to be used together must be maintained. For example, the Netlib Software Repository provides dependency checking that allows the user to retrieve a top-level routine plus all routines in its dependency tree (i.e., those routines that are called directly or indirectly by the top-level routine). Another example is a graphical parallel programming environment that relies on an underlying parallel communications support package. The problem becomes more complex when different pieces might be retrieved from different physical repositories. Ideally, the user should be able to have a consistent set retrieved automatically without having to scan documentation to verify that compatible pieces have been retrieved.
- As the number of reuse libraries grows, users cannot be expected to access each of them separately using a different interface. Thus, scalable interoperability between separately managed repositories is needed.
- In the environment of accessing a few well-established repositories that the user knows and trusts, a user is assured of the integrity and authenticity of a retrieved file because these properties are provided by the administrative procedures of that repository. With a large number of less familiar repositories, however, it becomes necessary to establish interoperable trust mechanisms and to reduce the number of parties with whom the user must establish trust.
- The more decentralized and smaller the individual repositories become, the less practical it becomes for each individual repository to provide the full range of search and authentication services.

Most of the above problems can be alleviated by implementing a location-independent naming system that includes mechanisms for authenticity and integrity checking. We have designed a naming system that provides for two levels of naming. The binding between a lower-level name (called a LIFN) and file contents is unchangeable and verifiable. A lower-level name may be resolved to multiple, mirrored copies. In the case where it represents a set of files, the name may be resolved to a list of other names. A higher-level name (called a URN) is associated with a cataloging record that includes the lower-level name as well as other descriptive information. This record may be cryptographically signed by the publisher so that users may verify the authenticity of a retrieved resource. At any given time, a higher-level name is associated with exactly one lower-level name, but this binding may change over time. Higher-level names allow for long-lived human-readable references, while lower-level names permit reliable caching and mirroring as well as permitting precise references when needed. Location-independent names will be the basis of transparent mirroring. They will also provide a unique key to which third parties may attach value-added information such as additional cataloging information and quality assessments. This paper describes the design of our naming system. We also describe our implementation of a prototype name-to-location service and of a modified WWW client that does name resolution. A glossary of acronyms and terms used in this paper is included as an appendix.

2 Related Work

The use of a public-key encryption technique for authenticating the source of a software component and for ensuring that the component has not been altered subsequent to its publication is proposed in [9]. Cryptographic information, in the form of a digital signature created by signing the hashed digest of the contents of a component, is included within the component's unique identifier. The proposed method is intended to prevent not only changes by unauthorized parties, but also changes by the original author – i.e., the author is not permitted to modify a component without assigning a new unique identifier. The method assumes that each author has been assigned a globally unique Author ID, has chosen an asymmetric public/private key pair, and has publicized the public key to the community of potential reusers. A newly chosen symmetric encryption key is used to encrypt the component itself. Then the symmetric key, the hashed digest of the component, and the Author ID are concatenated and encrypted using the asymmetric private key, and the result is concatenated to the clear-text version of the Author ID to create the unique identifier for the component. The method does not address name-to-location resolution, other than to say that the encrypted component is made available along with the unique identifier and any other cleartext information. The proposed unique identifier is similar to our LIFN, and encryption of the hash digest and Author ID is similar to our method of having the author cryptographically sign a catalogue record that includes the author name and the file's MD5 signature. Our method allows a choice of encryption algorithms, however, and allows the digital signature used for authentication to be generated independently and at a different time from the component's identifier.

Functional requirements for Uniform Resource Names (URNs) are proposed in [12] by the IETF Uniform Resource Identification (URI) Working Group. According to [12], the function of a URN is to provide a globally unique, persistent identifier used for recognition of and for access to characteristics of a resource or to the resource itself. URN assignment is delegated to naming authorities, the names of which are persistent and globally unique, and who may assign names directly or delegate their authority to sub-authorities. Global uniqueness of URNs is guaranteed by requiring each naming authority to guarantee uniqueness within its portion of the URN namespace. It is left up to each naming authority to determine the conditions under which it will issue a URN (for example, whether or not to issue a new URN when the contents of a file change). Some test implementations of URNs are underway by members of the URI Working Group at Georgia Tech and Bunyip Corporation [9]. The Georgia Tech testbed uses the whois++ protocol for URN to URC resolution. A URC, or Uniform Resource Characteristic, is a catalog record which includes locations, or URLs, at which the resource may be accessed. The URC server supports searching by other attributes, in addition to URN lookup, via the whois++ protocol. A modified version of Mosaic that does URN to URC resolution is available. A proxy server based on CERN httpd that does cacheing by URNs is also running at Georgia Tech.

As part of the Computer Science Technical Report (CSTR) project [8], which is developing an architecture for distributed digital document libraries, the Corporation for National Research Initiatives (CNRI) is implementing a name-to-location resolution service called the Handle Management System (HMS) [10]. CNRI's *handle* is a name for a *digital object* and is analogous to IETF's URN. The HMS includes a Handle Generator that a naming authority may run and use to create globally unique handles, Handle Servers that process update requests from naming authorities and query requests from clients to resolve handles, and a Handle Server Directory that maps a handle to the appropriate Handle Server. The distribution of handles to Handle Servers is based on a hashing algorithm. An electronic mail interface is used by handle administrators to add, delete, and modify handle entries in the Handle Server database. Clients use a UDP datagram interface to request location data associated with a handle. A modified version of Mosaic that does handle resolution is available from CNRI. The types of location information stored by Handle Servers include URL, repository name, email address, and X.500 Distinguished Name. Use of a repository name by a client requires another round of name-to-location resolution. CNRI's *properties record* that describes the properties of a digital object is analogous to IETF's URC. The properties record is not stored by the HMS, but rather by an Information and Reference (IR) Server that is to be maintained by each repository. Each naming authority may also maintain an IR server containing a properties record for each digital object within its authority.

3 Publishing and Name Assignment

Internet-accessible resources are currently referenced using Uniform Resource Locators (URLs). Because URLs are locations rather than names, their use as references presents at least two problems. One problem is that files get moved, changing their URLs. Then pointers that contain the old URLs become stale. One can leave a forwarding address at the old URL, but forwarding addresses are an awkward and inelegant solution. Another problem with using URLs as references is that mirrored copies of files cannot be located from a URL reference, since each copy has a different URL.

It has been widely recognized that a solution to the above problems is to assign location-independent names to files and to provide a name-to-location service that, given a name, returns a list of locations for that name. A resource provider who moves some files need only delete the old name-to-location bindings and register the new bindings with the name-to-location service. Likewise, a site that mirrors a copy of a file need only register its location with the name-to-location service. Then a user attempting to retrieve the file corresponding to a location-independent name may query the name-to-location service for a list of alternative locations to be tried.

Our work is similar to the IETF's Uniform Resource Identifier Working Group's work on Uniform Resource Names (URNs) [12] and to CNRI's work on unique document identifiers for digital libraries [8]. However, neither of these groups has addressed the reliability and consistency issues addressed by our two-level naming system. Our system includes a lower-level name a called Location Independent File Name (LIFN) and a higher-level name called a Uniform Resource Name (URN).

An important question is whether the byte contents of the file referred to by a location-independent name should be fixed or be allowed to change. If the byte contents are allowed to change, then a further question arises as to what should be the consistency requirements for alternative locations for the same name. Valid arguments for both cases can be made for different situations. For example, for software resources it is desirable to have an unambiguous reference to the fixed byte contents for the purpose of attaching a review or reporting experimental or performance results. Fixed contents also make it possible to compute a file digest that may be cryptographically signed by the author of the resource, allowing verification of the integrity of a retrieved file. On the other hand, it is desirable to have a reference to a software package that need not be changed every time a bug fix or minor revision takes place, especially if the cataloging information (e.g., title, author, abstract) does not change. The cataloging information for a software package might contain a reference to a Web page describing and/or documenting the package. The author of the Web page would like to be able to update the page without having to change all the references to it. A non-software example where it would be desirable to allow contents to change is a name that refers to a file containing the "current weather map".

Because both types of name are needed, we have implemented both. The type of name that refers to fixed byte contents is called a Location Independent File Name, or LIFN. Once a LIFN has been assigned to a particular sequence of bytes, that binding may not be changed. The type of name for which the contents to which it refers may change is called a Uniform Resource Name, or URN.

We divide the file access system into two levels. The upper level is where publishing, cataloging, and searching activities take place. These upper-level activities are concerned with the semantic, or intellectual, contents of files. The lower level is where distribution, mirroring, and caching

[9] More information is available at **http://www.gatech.edu/iiir/**

[10] More information is available at **http://www.cnri.reston.va.us/**

activities occur. These lower-level activities are not concerned with the semantic contents of files, only with ensuring that files may be accessed efficiently and that the byte contents of files are not corrupted.

The above arguments about the need for two types of name pertain to the upper level. At the lower level, there is a need for LIFNs, but not for URNs. Mirror sites use LIFNs and their associated file digests to ensure that their copies of files have not been corrupted. A cache site needs to be able to tell a user or client program whether it holds a copy of a requested file, and for this purpose it can answer whether or not it holds a copy of a particular LIFN.

The above considerations led us to implement LIFNs at the lower level of the file access system and URNs at the upper level, but to make LIFNs visible at the upper level as well. A publisher will be responsible for assigning both a URN and a LIFN to any resource for which cataloging information is provided. For other files, only LIFNs need be provided. At any given time, a URN that refers to a file or a set of files is associated with exactly one LIFN. A URN may be associated with a set of different LIFNs over the URN's lifetime, but we require that the set be in the form of a linear sequence, with the sequence order given by increasing time.

The LIFN and URN name spaces are subdivided among several *publishers*, also called *naming authorities*, who are responsible for ensuring the uniqueness of names assigned within their portions of the name spaces. A name is formed by concatenating the registered naming authority identifier with a unique string assigned by the naming authority. The LIFN and URN are formatted as

```
LIFN:<publisher id>:string
URN:<publisher id>:string
```

The **publisher id** portion of the name is used to locate appropriate URN and LIFN servers for that publisher. Given a URN, a URN server returns a Uniform Resource Citation (URC) for that URN that includes its currently associated LIFN, as well as other cataloging information. Given a LIFN, a LIFN server returns a list of locations for that LIFN. More information about accessing URCs and files from their URNs and LIFNs may be found in Section 4.

The publisher provides cataloging information for each URN it assigns. The catalog record includes information such as title, author, abstract, etc. A recommended set of attributes for software assets is given by the Reuse Library Interoperability Group (RIG) Basic Interoperability Data Model [1]. In addition, the catalogue record for a URN includes its currently associated LIFN, as well as an MD5 or similar fingerprint for that LIFN. This fingerprint is a 128-bit quantity resulting from applying the MD5 function to the contents of the file. The function is designed to make it computationally infeasible to find a different sequence of bytes that produces the same fingerprint [10]. To enable authentication, the entire description may be cryptographically signed, as discussed in Section 5. Portions of the catalog record may be exported to resource discovery servers, such as a Harvest Broker [3], which provide search services based on resource descriptions. The URN exported to the search service provides a unique long-lived key, so that descriptions may be unambiguously associated with a resource, and so that a resource turns up at most once in a list of search hits.

For a name to be useful, there must be some means of resolving a name to a location from which the resource can be retrieved or accessed. Thus, the publisher, as well as

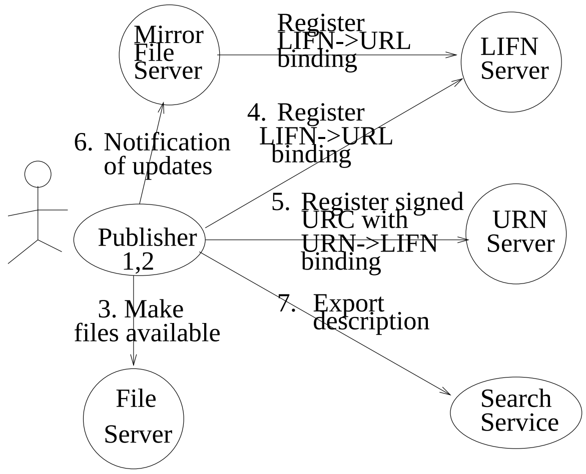

Figure 2: Publishing steps

any other parties that mirror the resource, must register such locations with the appropriate name-to-location lookup services. Such name-to-location services are discussed in Section 4.

Thus, publishing a resource involves the following steps, shown in Figure 2:

1. creating the resource's catalog record in the form of a URC,
2. signing the catalog record with the publisher's private key,
3. making the resource files available on one or more file servers,
4. registering the file locations with the LIFN server,
5. registering the URC with the URN server,
6. informing mirror sites of the new or updated file,
7. exporting relevant portions of the URC to search services.

Steps 1 and 5 have been discussed above. Steps 2 is discussed in Section 5, and Steps 3, 4, and 5 are discussed in Section 4.

4 Name Resolution and File Mirroring

Resources available from the virtual repository will be named by URNs and/or LIFNs, rather than by URLs. Thus, WWW clients will need a means of resolving a URN or LIFN to one or more locations, expressed in the form of a URL, to be able to access the resource. Access to files is provided by conventional file servers, using protocols such as HTTP, Gopher, and FTP.

For a non-file resource, such as a database service, a list of locations is associated directly with the URN for that resource. For a file resource, such as a file containing a piece of software, the relationship between the URN and the locations is indirect, via a LIFN – the URN is associated with a LIFN, and the LIFN is associated with a list of URLs.

The LIFN-to-location mapping service is provided by a network of LIFN servers, collectively called the LIFN database. These servers process queries for locations of

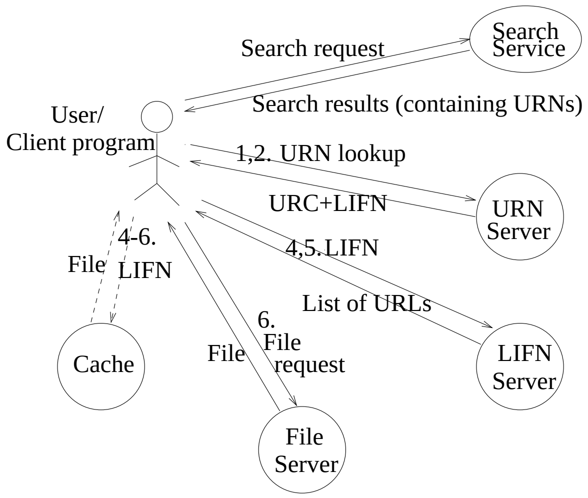

Figure 3: File access steps

LIFNs. They also accept updates from file servers containing new locations for LIFNs, as well as requests to delete old LIFN-to-location mappings. A naming authority may run its own LIFN servers, or it may find another organization willing to provide the service on its behalf.

The URN service is similar to the LIFN service, except that it maps a name either to a list of locations or to a URC that includes a LIFN. For fault tolerance and availability, the URN service is also provided by a network of servers.

Mappings from naming authority identifiers to URN and LIFN servers are stored in the the Domain Name System (DNS) name space, so that a client program can determine which URN (LIFN) server to query for a particular URN (LIFN). Our current client uses an ordinary DNS lookup for IP address records. The publisher identifier is prepended to the string `.LIFN.NETLIB.ORG` (for a LIFN) or `.URN.NETLIB.ORG` (for a URN). The resulting string is treated as if it were the name of an Internet host, and DNS is queried to find the IP addresses of that host. For example, to find a LIFN server for the naming authority `foo`, the client would look up the IP addresses for `foo.LIFN.NETLIB.ORG`. Several IP addresses may be listed for any one naming authority. Our client attempts to query each IP address until it finds one that can satisfy the LIFN or URN lookup request.

Thus, the steps involved in resolving a URN so as to access a copy of the file it names are as follows, as shown in Figure 3:

1. Use DNS to locate an appropriate URN server.
2. Query the URN server to retrieve the URC which contains the currently associated LIFN.
3. Authenticate the URC if desired.
4. Use DNS to locate an appropriate LIFN server.
5. Query the LIFN server to retrieve a list of locations.
6. Choose a location from which to retrieve the file.

In practice, Steps 4 through 6 will often be replaced by using the LIFN to access a local cache server. Because the binding between a LIFN and the byte contents it points to is fixed, the cached copy is sure to be correct.

A file server can mirror a file by acquiring a copy of it and posting an update to a LIFN server for the file's naming authority. If a file server moves or deletes a file, then it would post that information as well. It is not necessary to keep all LIFN servers for a particular naming authority perfectly synchronized. Such synchronization would entail too much overhead. Instead, location updates are posted to a any LIFN server and propagated to other peer servers using a batch update protocol.

Updates to the URN server are posted by the publisher and by others authorized by the publisher to update the catalog record for a given URN. In order to ensure a consistent linear history of updates to the catalog record for a URN (e.g., the sequence of LIFNs associated with that URN), replicated URN servers use a master-slave update protocol.

One of the most important aspects of our use of LIFNs is that it assures the user of retrieving the most up-to-date copy of a file referenced by a URN, without the overhead of a replica control protocol between file servers mirroring that file, which in general will not all be under the control of the URN's naming authority. This assurance is modulo the time required for the master-slave update protocol for the replicated URN servers, but if the user insists on contacting the master URN server, he is ensured of getting the most up-to-date copy.

5 Authenticity, Integrity, and Consistency of Resources

Authentication of a resource verifies that the resource was published by its purported publisher. Verifying the integrity of a file ensures that the file has not been modified. Provisions for authenticity and integrity checking are necessary for a software repository because there have been instances of software packages stored on a public repository that were modified by intruders to introduce security holes which were then spread to other systems[11]. Our authentication and integrity mechanisms are similar to those described in [11] and [9].

Recall from Section 3 that a publisher cryptographically signs the catalog record for a resource. In the case of a file resource, this record includes the file's LIFN and MD5 fingerprint. Any client in possession of the publisher's public key can verify the authenticity of the resource description. Publishers are expected to widely advertise their public keys to make it difficult for an attacker to substitute rogue keys. In addition, publishers may have their keys certified by trusted third parties to further establish their authenticity, as in [11].

Assuming that the association between a LIFN and a file signature (e.g., the MD5 fingerprint) is known to be correct (either because the signature is part of the LIFN or because of the description authentication described in the preceding paragraph), a client may perform an integrity check on a retrieved file by computing the signature for the file and comparing it with the one known to be associated with the file's purported LIFN. Recall from Section 4 that a LIFN server returns a list of locations for a given LIFN but does not guarantee the correctness of those locations. A location may be incorrect if it no longer exists or if the contents of that location are wrong. In the former case, no file will be returned from that location. The latter condition may be detected by the client performing an integrity check.

To ensure consistency within a group of related files, we allow a URN to refer to a set of files. There are at least two

[11]For an example, see the CERT advisory at **ftp://ftp.cert.org /pub/cert_advisories/CA-94:07.wuarchive.ftpd.trojan.horse**

cases where this might occur. One case is where a resource consists of a number of related files, for example the files making up a software package. Of course, such a set of files could be made available instead as a single tar file. If a file can be used in more than one package, however, or if some files are also of use individually, it might be preferable to make the files available separately. Another case is when there are alternative versions of a file – for example, multiple precisions of a Fortran routine, or multiple formats of an image.

The first case is handled by ordering the files making up the resource and considering the ordered list of LIFNs for these files to be the contents of another file which we call the **composite-parts-list** for the resource. The composite-parts-list file itself has a LIFN, and it is this LIFN that is associated with the URN for the resource. The second case is handled in a similar manner, but the file containing the ordered list of LIFNs is called the **alternative-parts-list** for the resource. The parts-list may contain additional information, such as how the alternative parts vary. After retrieving a parts-list, the client program will invoke a special module for handling it, similar to how current browsers invoke viewers for image or sound files. This module will assist the user in retrieving the component files and saving or displaying them locally.

6 Prototype Implementation

The naming system is being implemented as part of the Bulk File Distribution (BFD) package. BFD is part of the implementation of the National HPCC Software Exchange (NHSE), which is being developed by the Center for Research in Parallel Computing (CRPC), a consortium of universities and national laboratories formed to make high performance and parallel computing accessible to engineers and scientists. BFD URN and LIFN servers will run at all the CRPC participating sites, as well as at other major NHSE sites, such as Oak Ridge National Laboratory.

A BFD client is a WWW browser that, in addition to having the capability to retrieve a file given its URL, also has the capability to retrieve a file given its URN or LIFN. A version of NCSA Mosaic 2.4 for X Windows that has been modified to support BFD is available at `http://www.netlib.org/nse/bfd/`. A BFD client library that can be incorporated into other Web browsers will be available soon.

The prototype implementation of BFD uses query and update protocols based on Sun's Remote Procedure Call (RPC) mechanism over UDP. RPC was chosen because it is very lightweight (one packet for request, and one for reply), widely supported on UNIX platforms, and easy to implement on other platforms (at least for the portions of RPC needed by BFD). The BFD RPC requests are sent to a server at a fixed port number, rather than using the RPC portmapper, to avoid the overhead of an extra RPC call.

To locate a LIFN server, BFD uses an ordinary DNS lookup for IP address records. `LIFN.NETLIB.ORG` is the implicit root of the LIFN name tree. For example, to find a LIFN server for the naming authority `foo`, a client looks up the IP addresses for `foo.LIFN.NETLIB.ORG`. IP addresses were used instead of new DNS records types because experiments showed that many DNS servers would not accept unknown record types. Several IP addresses may be listed for any one naming authority.

The BFD LIFN database is a simple key/data database in which the unique keys are LIFNs. Sending a BFD LIFN server a query containing a LIFN causes a list of URLs to be returned, possibly along with other information. Sending a BFD LIFN server an update containing a LIFN/URL pair (and possibly additional location-specific descriptive information) causes that pair to be added to the database.

The URN database and protocols have been implemented in an analogous manner. The current URN server stores only the LIFN attribute in the URC for a URN.

To test the system, LIFNs were assigned to the software components making up the LAPACK directory in Netlib (around 2500 files total). Each of these LIFNs was of the form

```
lifn:netlib:<signature>
```

where `<signature>` is the ascii form of the MD5 signature of the file. The URLs listed for the LIFNs were of the form

```
<protocol>://<hostname>/<path>/<lifn>
```

When a client program requests such a URL from the file server, the file server either returns a file that is correct for the given LIFN, or it returns an error indicating that the file corresponding to that LIFN was not found. The overhead for assigning these LIFNs involved running a script that computed the MD5 signatures and generated the LIFNs, created a directory that aliased the ascii form of the MD5 signatures to the actual file locations, and registered the LIFN-to-URL mapping with the LIFN server. For the 2482 files in the test described above, this script took 2 minutes 35 seconds CPU time.

7 Conclusions and Future Work

We have designed a naming system that provides for two levels of location-independent naming. At the lower level, there is an immutable association between a location-independent filename, called a LIFN, and a specific byte stream. A higher-level location-independent name, called a URN, is associated with a particular LIFN at any given time, but with a linear sequence of LIFNs over its lifetime. We have deployed URN and LIFN servers that provide URN and LIFN lookup services, and we have made available a modified version of Mosaic that can retrieve files named by URNs or LIFNs.

We have described mechanisms, based on a public key encryption system, for verifying the authenticity of LIFN and URN servers, of trusted file servers, and of resource descriptions. Although we have not yet implemented such mechanisms, we plan to do so soon. We will initially use the PGP public-key encryption system [13].

Our naming system will help provide a uniform interface to a virtual distributed software repository, such as the National HPCC Software Exchange, while preserving the advantages of distributed maintenance of contributed software and of file mirroring. Our consistency, authenticity, and integrity mechanisms will provide assurances that software components retrieved from independent sources are consistent with their verifiable descriptions. Use of LIFNs will allow value-added descriptions, such as critical reviews, to be unambiguously associated with the exact file or set of files that was reviewed. Referring to a LIFN also allows a researcher to unambiguously specify the exact piece of software used to produce and report experimental results.

As part of the BFD package, we plan to implement a replication daemon that acquires new files from remote servers, deletes files that are no longer wanted, and informs

LIFN servers of the changes. These functions are similar to those provided by several existing mirror programs, such as the Netlib repository mirroring scheme described in [7], but with the addition of interacting with the LIFN database. The BFD replication daemon will be designed to perform its tasks very efficiently. Planned features include on-the-wire compression, checkpoint/restart, multiple file multiplexing (to allow for gradual transfer of very large files), integrity checking, and a protocol that works well over high bandwidth-delay links.

A collection manager program will also be part of the BFD package. The collection manager will decide which files to acquire and which ones to keep or throw away, based on access statistics and site-specific criteria. The results of such decisions will then be fed to one or more replication daemons.

We are involved in discussions with the IETF URI Working Group and with CNRI that we hope will lead to a merging of the different technologies the three groups are developing for name-to-location resolution, meta-information lookup, and searching.

A Glossary of Acronyms and Terms

BFD Bulk File Distribution

BIDM Basic Interoperability Data Model

CNRI Corporation for National Research Initiatives

CRPC Center for Research in Parallel Computing

CSTR Computer Science Technical Reports, a digital library project

DNS Domain Name System

FTP File Transfer Protocol

GAMS Guide to Available Mathematical Software

Harvest An information discovery and access system

HMS Handle Management System, a name-to-location resolution service being developed at CNRI

HPCC High Performance Computing and Communications

HTTP HyperText Transfer Protocol

IETF Internet Engineering Task Force

IP Internet Protocol

LAPACK A linear algebra software package

LIFN Location Independent File Name

MD5 A message digest algorithm

Mosaic A World Wide Web browser

NCSA National Center for Supercomputing Applications

Netlib A mathematical software repository

NHSE National HPCC Software Exchange

PGP Pretty Good Privacy, a public key encryption package

RIG Reuse library Interoperability Group

RPC Remote Procedure Call

URC Uniform Resource Characteristic

URL Uniform Resource Locator

URN Uniform Resource Name

WWW World Wide Web

References

[1] Standard reuse library Basic Data Interoperability Model (BIDM). Technical Report RPS-0001, Reuse Library Interoperability Group, 1993.

[2] R. F. Boisvert. The architecture of an intelligent virtual mathematical software repository system. *Math. & Comp. in Simul.*, 36:269–279, 1994.

[3] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado - Boulder, Aug. 1994.

[4] S. Browne, J. Dongarra, S. Green, K. Moore, T. Rowan, and R. Wade. Netlib services and resources. Technical Report UT-CS-94-222, University of Tennessee Computer Science Department, Feb. 1994.

[5] S. Browne, J. Dongarra, S. Green, K. Moore, T. Rowan, R. Wade, G. Fox, K. Hawick, K. Kennedy, J. Pool, and R. Stevens. The National HPCC Software Exchange. *IEEE Computational Science and Engineering*, 1995. (to appear).

[6] J. J. Dongarra and E. Grosse. Distribution of mathematical software via electronic mail. *Commun. ACM*, 30(5):403–407, May 1987.

[7] E. Grosse. Repository mirroring. *ACM Trans. Math. Softw.*, 21(1), Mar. 1995.

[8] R. R. Larson. Design and development of a network-based electronic library. In *Proc. ASIS Mid-Year Meeting*, pages 95–114, Portland, Oregon, May 1994.

[9] J. W. Moore. The use of encryption to ensure the integrity of reusable software components. In *Proc. Third International Conference on Software Reusability*. IEEE Computer Society Press, Nov. 1994.

[10] R. Rivest. The MD5 message-digest algorithm. Internet RFC 1321, Apr. 1992.

[11] A. D. Rubin. Trusted distribution of software over the Internet. In *Internet Society 1995 Symposium on Network and Distributed System Security*, 1995. (to appear).

[12] K. Sollins and L. Masinter. Functional requirements for Uniform Resource Names. Internet RFC 1737, Dec. 1994.

[13] P. Zimmerman. PGP user's guide. PGP Version 2.6.2, Oct. 1994.