Distributed Storage in RIB

Thomas Brant Boehmann
March 23, 2003

Computer Science Department
University of Tennessee
Knoxville, TN 37996-1301

ICL Technical Document Number: ICL-UT-03-01

ABSTRACT

Recently, the Logistical Computing and Internetworking (LoCI) Lab at the University of Tennessee has created an abstraction of distributed network storage known as the *Network Storage Stack*. Many applications would benefit from making use of the foundations provided by LoCI. This paper is intended to serve as an explanation of how the Repository in a Box (RIB) toolkit, which was created by the Innovative Computing Lab (ICL) as part of the National HPCC Software Exchange project, can increase functionality by becoming an application at the top of the *Network Storage Stack* and making use of the Logistical Runtime System (LoRS) and exNodes.

1. INTRODUCTION

Myriads of applications have storage requirements that can be difficult to meet. The Repository in a Box (RIB) toolkit is one such application. This metadata repository could benefit from storage as well as bandwidth relief, for example for archiving and retrieval of large data files resulting from simulations. Such relief can be provided by using distributed storage. Distributed storage can easily be facilitated using logistical networking techniques provided by the *Network Storage Stack* with the Internet Backplane Protocol (IBP) as its base.

2. REPOSITORY IN A BOX

The Innovative Computing Laboratory (ICL) at the University of Tennesseehas created software known as Repository in a Box (RIB) as part of the National HPCC Software Exchange project[5]. RIB is a software package for creating metadata repositories and archiving related data files. RIB allows users to enter data into a user friendly interface; the datathen get pushed to a RIB server over http. The information is stored in a database for future retrieval from a web-based catalog.

By default, RIB uses a simplified version of IEEE Standard 1420.1, **Basic Interoperability Data Model (BIDM)**, as the data model for storing and exchanging metadata. The purpose of the BIDM is to define the minimal set of information about assets that reuse libraries should be able to exchange in order to support interoperability [2]. Other data that would be useful but are not included in the BIDM but can be added to RIB's data model by an administrator via a simple interface after defining the necessary extension to the BIDM.

The BIDM consists of five classes that each have a set of attributes and relationships. An attribute is a field that contains a value. A relationship is a field that is used to relate one class to another class. RIGObject is the root class of the BIDM model. All sublasses inherit the *Name* attribute from the RIGObject class [5]. An illustration of the classes and their relationships using the graphic notation of James Rumbaugh's *Object-Oriented Modeling and Design* [3] can be viewed in Figure 2.

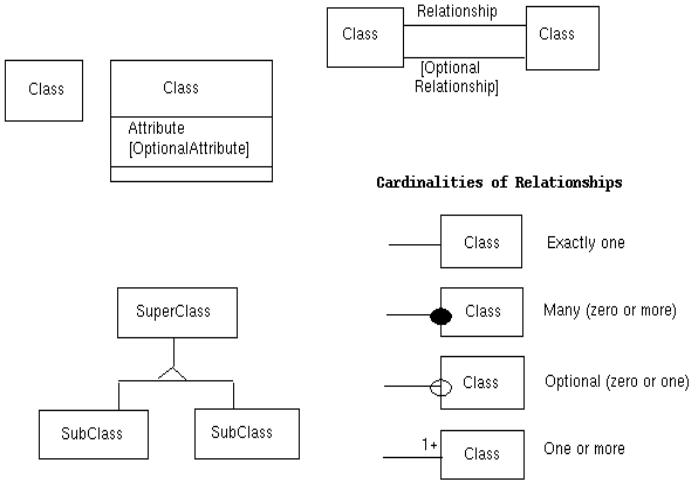

Figure 1. Legend for Data Model Notation [5]

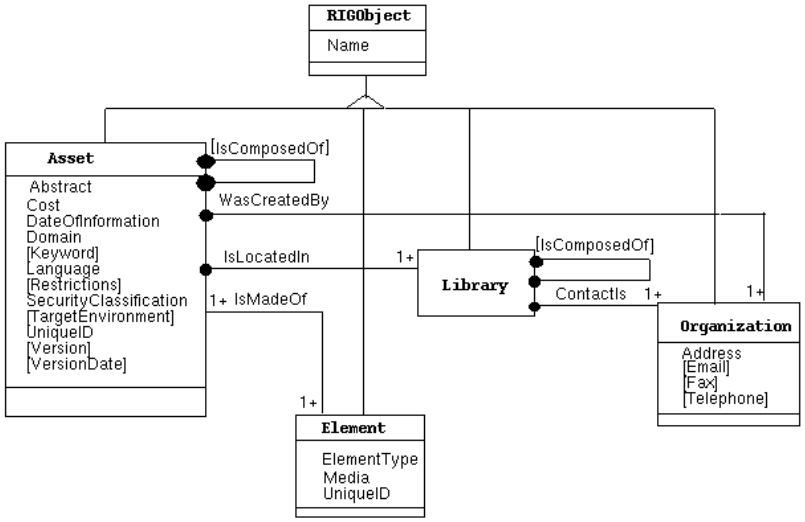

Figure 2. Basic Interoperability Data Model [5]

- **Asset** contains information about items of interest that are stored in a reuse library, such as design documentation, specifications, source code, test suites, or any other unit of information of potential value to the reuser.

- **Element** consists of a single file.

- **Library** is made up of **Assets**.

- **Organization** may be a person, company, research group, etc., that creates and manages an **Asset** or **Library**.

The *Element* object is of primary concern to this project. An object of class *Element* provides metadata about a single file. In recent versions of RIB, a facility has been added to upload files to the RIB server. Later, the user has the ability to attach the actual file to the *Element* itself. This is very useful when viewing the catalog, because the file can be downloaded directly from the catalog.

3. LOGISTICAL NETWORKING

The Logistical Computing and Internetworking (LoCI) Laboratory at the University of Tennessee is devoted to information logistics in distributed computer systems and networks. LoCI has designed and implemented a network storage abstraction called the *Network Storage Stack*. A pictorial view of the *Network Storage Stack* can be viewed in Figure 3.

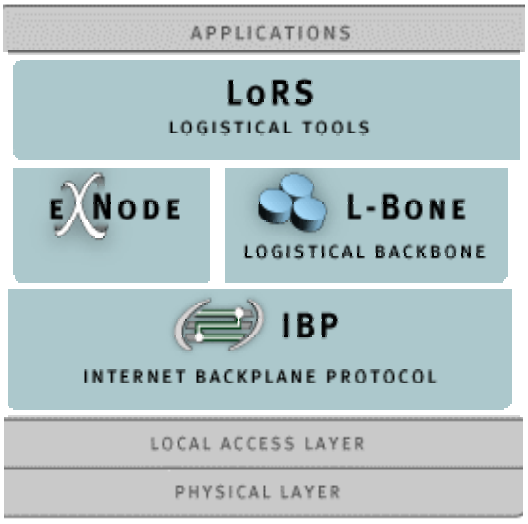

Figure 3. Network Storage Stack [4]

3.1 Internet Backplane Protocol

While the bottom two layers are simply the hardware and operating system layers of storage, the lowest layer of interest is the *Internet Backplane Protocol* (IBP).

IBP is a server daemon and a client library that allows storage owners to insert their storage into the network and allows generic clients to allocate and use this storage. The unit of storage is a time-limited, append-only byte-array. IBP's byte-array allocation is like a network *malloc()* call – clients request an allocation from a specific IBP Depot, and if successful, are returned encrypted text strings (called *capabilities*) for reading and writing [1].

3.2 L-Bone

The next layer contains the *L-Bone*, for resource discovery and proximity resolution, and the *exNode*, a data structure for aggregation. The L-Bone (Logistical Backbone) is a directory or registry that allows clients to perform IBP depot discovery. IBP depots register themselves with the LBone, and clients may then query the L-Bone for depots that have various characteristics, including minimum storage capacity and duration requirements, and basic proximity requirements [1].

3.3 ExNode

The exNode is a data structure for aggregation, similar to the Unix inode (Figure 4). Whereas the inode aggregates disk blocks on a single disk volume to compose a file, the exNode aggregates IBP byte-arrays to compose a logical entity like a file. Two major differences between exNodes and inodes are that the IBP buffers may be of any size, and the extents may overlap and be replicated (Figure 5) [1].

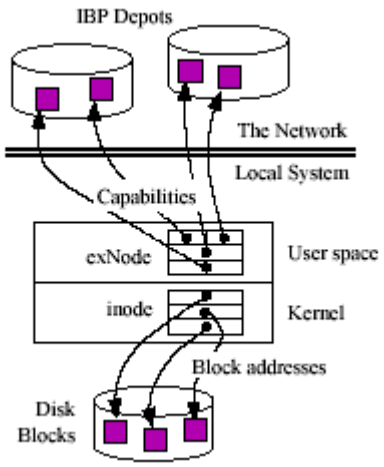

Figure 4: The exNode in comparison to the Unix inode [1]

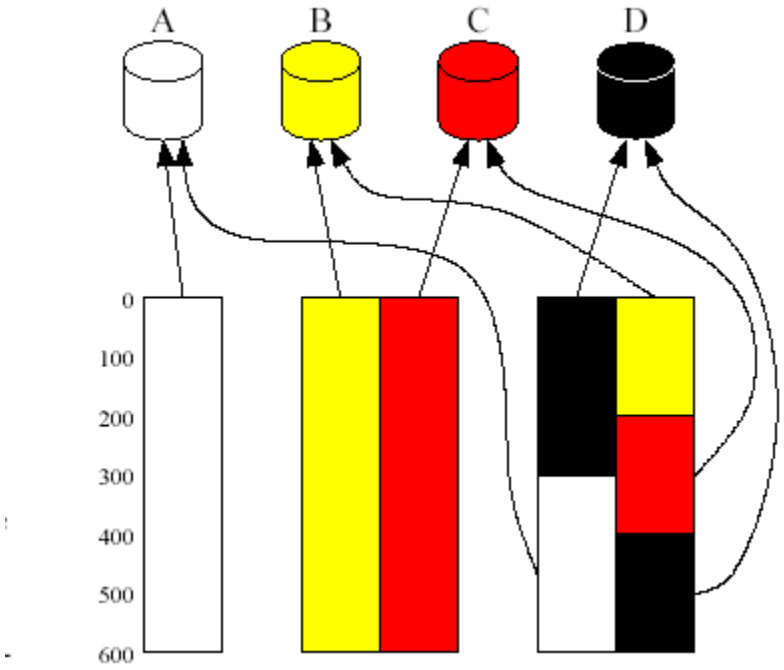

Figure 5: Sample exNodes of a 600-byte file with different replication strategies [1].

3.4 Logistical Runtime System

At the next level of the *Network Storage Stack* are tools that perform the actual aggregation of network storage resources, using the lower layers of the stack. These tools take the form of client libraries that perform basic functionalities, and stand-alone programs built on top of the libraries [1]. These tools are known as the *Logistical Runtime System* (LoRS). The basic functionalities of the LoRS tools include the following:

Upload: This takes a local file and uploads it into the network and returns an exNode for the upload.

Download: This takes an exNode as input and downloads a specified region of the file that it represents into local storage.

List: This takes an exNode as input and provides a long listing of the stored file's name and size and metadata about each segment or fragment of the file (individual IBP capability sets) including offset, length, available bandwidth and expiration.

Refresh: This takes an exNode as input and extends time limits of the IBP buffers that compose the file.

Augment: This takes an exNode as input, adds more replica(s) to it (or to parts of it), and returns an updated exNode.

Trim: This takes an exNode, deletes specified fragments, and returns a new exNode.

3.5 Applications

Lastly, applications sit at the top of the *Network Storage Stack*. These are applications that make use of distributed storage and logistical networking through the use of the LoRS tools. Indeed, the goal of this project was to make RIB an application on top of the *Network Storage Stack*.

4. USING LOGISTICAL NETWORKING IN RIB

As previously mentioned, RIB contains objects called *Elements* which contain metadata about individual files. Files can be uploaded via http and be attached to the *Element* and available for download via the web based catalog. In many cases it would be useful if exNodes could be used instead of the actual file, for example for very large data files.

4.1 An exNode Repository

One way to view RIB would be as an exNode repository. Currently files can be uploaded and attached to the corresponding *Element*; however, nothing prohibits that file from being an exNode. This method allows users to upload all their exNodes and keep metadata about the exNode. The problem with this is IBP space is time limited. So in order to keep exNodes from expiring, exNodes must be refreshed. A process can be added to RIB to keep all exNodes automatically refreshed. This process takes the form of a Perl script and a job inserted in the RIB user's cron tab. Another issue is data locality. An exNode could have been created in Australia and uploaded to a RIB server in North America. Performing operations on such an exNode could prove costly. It would be beneficial if copies could be stored in IBP depots that are closer to the RIB server. Another issue related to data locality is that it will be difficult for many RIB administrators to trust servers out of their control. It would be beneficial if the exNodes could point to space that the administrator either controls or trusts. This is where an *augment* operation would be useful. The same process in RIB that keeps all exNodes refreshed also has the capability to automatically augment the exNodes to "trusted" depots. The RIB administrator can configure the LoRS tools to use preferred IDP depots using the .xndrc file.

4.2 Using an external exNode Repository

Other, more advanced software packages that will serve as exNode repositories are under development. These include a server from the LoCI lab called ROLF. Perhaps these repositories, which are external to RIB, will prove more functional than using RIB as an exNode repository. However, RIB's main purpose remains to collect metadata regardless of location of storage. In this case, using an extension to the BIDM known as the Standard NHSE Extension could be helpful. This extension adds a webpage attribute to the top level RIGObject class which is inherited by all objects including the *Element*. Using the webpage attribute, a user could store a URL to the exNode contained in an external repository. By doing so, the URL to the exNode would appear in the catalog and could be navigated to or downloaded with a simple mouse click.

4.3 Using LoRS from the RIB interface

Performing LoRS operations on files contained in RIB is key functionality if RIB is to be used as an exNode repository. Reasons for doing this include the following:

Bandwidth at the client machine is low, so uploading an exNode would be faster than uploading an entire file. Then, a *download* operation could be performed from the RIB server where presumably higher bandwidth is available. The file attached to the Element would then be changed to the "real" file as opposed to the exNode.

A file has been uploaded in the past and based on the use of that file it has become apparent that distributing an exNode would be more useful to the consumer of the file. Thus, an *upload* can be performed on the file, and now the exNode is associated with the element instead of the original file. This could also be done to conserve space on the RIB server.

Simply storing exNodes in an organized manner is useful. Using RIB as an exNode repository allows remote access to a user's exNodes regardless of location. RIB provides a place to store metadata about the exNode (or contents of the exNode). A *list* operation could be performed to see exNode information.

As mentioned before, automatic *refresh* and *augment* capabilities have been added to RIB, but perhaps the RIB administrator has chosen not to enable this feature. Being able to *refresh* and *augment* manually from the RIB interface itself is also useful.

This project has made changes to both the RIB server and the RIB user interface to expose LoRS functionality to the user. First the LoRS tools need to be installed on the server. Second, a set of perl based CGI programs have been added to encapsulate the logic and make the calls to the LoRS tools. Third, the java applet that provides the RIB administrative interface has been modified to provide an interface to the new IBP functionality. You can see the components of RIB and their interactions in Figure 6.

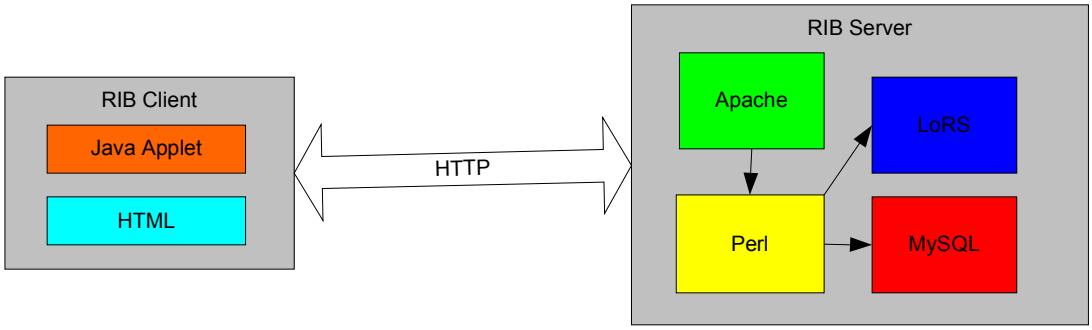

Figure 6: RIB interaction

The new IBP functionality has been exposed in the RIB administrative interface as a context-sensitive menu (Figures 7 and 8). This menu appears when right clicking on an object in the object browser. If the object is an element and has a file attached, the user will see the IBP submenu. If the file is an exNode, then the IBP submenu will contain the following options: *download, augment, refresh, and list*. If the file is not an exNode, the IBP submenu will provide

an option to *upload* the file into IBP. When the *download* option is chosen, the RIB server will download the file from IBP and place it on the RIB file system. The database record associated with the *Element* will now point to the new file instead of the exNode. Conversely, when the *upload* option is selected, the file will be uploaded into IBP space, an exNode will be placed on the RIB server's file system, and the database record associated with the *Element* will point to the new exNode.

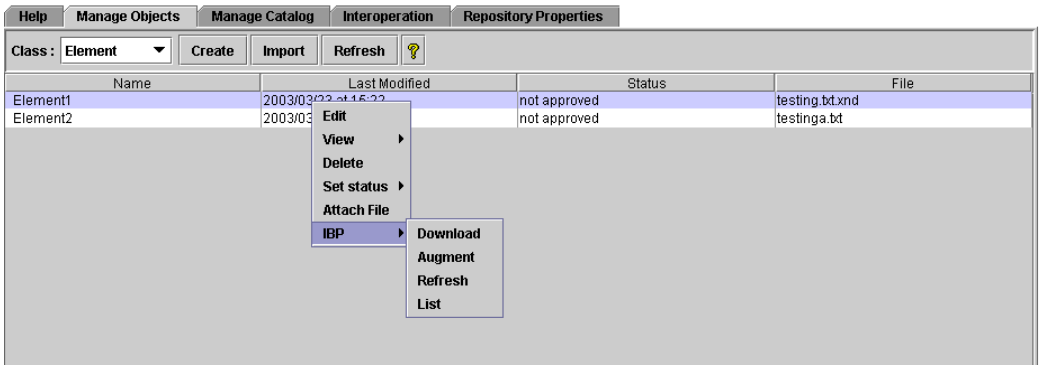

Figure 7: IBP submenu for an Element with exNode attached.

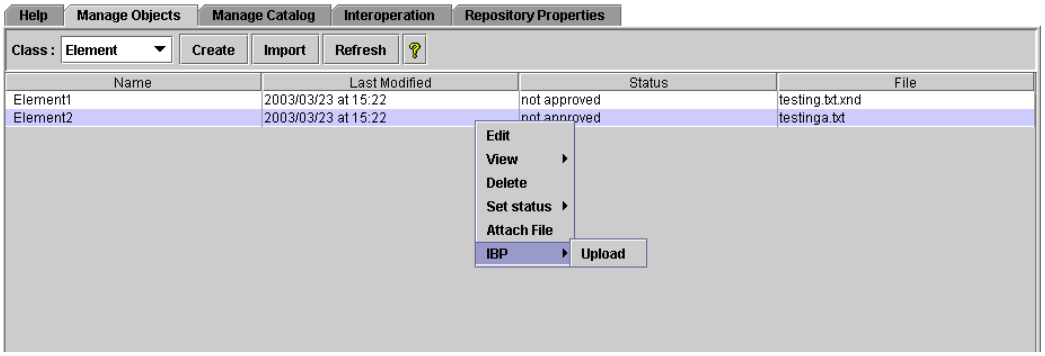

Figure 8: IBP submenu for Element with "normal" file attached.

When the *list* option is selected, a popup window will appear showing the exNode information obtained from the LoRS list command (Figure 9).

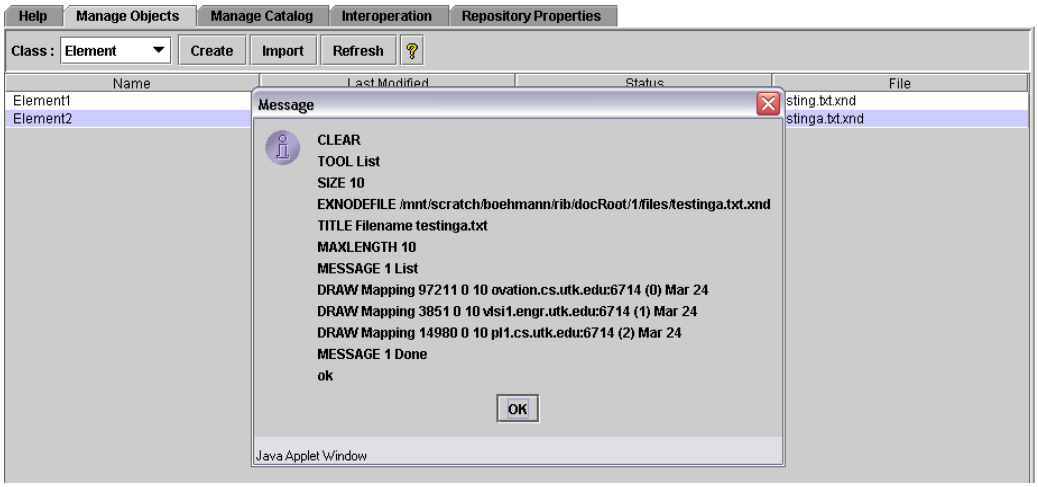

Figure 9: Popup window as a result of choosing "List" option.

5. CONCLUSION AND FUTURE WORK

The ability to utilize distributed storage via IBP is a very useful addition to a software package whose main concern is to store metadata. Such functionality allows for the conservation of space on RIB servers while bringing the RIB package to the edge of the latest in storage technologies. There are several aspects of this work that could be expanded upon to provide more and better functionality.

The LoRS tools which are wrapped by the RIB server provide many options for customization. Currently all operations exposed are using the tools under their default operation modes. At present the only way to take advantage of any of these options is by using the .xndrc file which makes for global options. It would be nice if a more feature-rich interface could be provided at the user level so that the user can choose options for LoRS operations. A trim operation from the RIB interface could be useful, but this is a difficult operation for which to provide defaults.

Right now, using the upload and download options may be difficult for files of large size. The reasons for this are two-fold. One, the User Interface blocks during these IBP operations. Two, the communication between the applet and the RIB server is facilitated via http. HTTP is not a very reliable communication mechanism. In order to make this more robust, an alternate protocol to http could be used for a more reliable communication mechanism. If these operations were asynchronous, the user would be able to perform other operations while waiting for the other operations to complete. Issues such as error handling would have to be addressed in an asynchronous architecture.

ACKNOWLEDGEMENT

I would like to give special thanks to my advisor, Dr. Shirley Moore, for helping me find a PILOT project and helping me along the process. I would also like to thank Dr. Jack Dongarra and Dr. Micah Beck for serving on my graduate committee. Thanks to Yuanlei Zhang for

answering all my questions about RIB and shipping me the latest source code. Thanks to Jeremy Millar for being so helpful while I was trying to understand IBP and the LoRS tools.

REFERENCES

[1] Atchley, S., Soltesz, S., Plank, J. S., Beck, M., and Moore T. Fault-Tolerance in the Network Storage Stack. *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, Ft. Lauderdale, FL*, April, 2002.

[2] *IEEE Standard for Information Technology - Software Reuse - Data Model for Reuse Library Interoperability: Basic Interoperability Data Model (BIDM)*. IEEE Std 1420.1, 1995.

[3] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall: Englewood Cliffs, New Jersey, 1991.

[4] Logistical and Internetworking Lab Website, *http://loci.cs.utk.edu*.

[5] Repository in a Box Homepage, *http://www.nhse.org/RIB*.