

# Using Standards-based Interfaces to Share Data across Grid Infrastructures

K. Sarnowska\*, A. S. Grimshaw\* and E. Laure\*\*

\* Department of Computer Science, University of Virginia, Charlottesville, USA

\*\* PDC, Royal Institute of Technology, Stockholm, Sweden  
{sarnowska, grimshaw}@virginia.edu, erwinl@pdc.kth.se

**Abstract**—Data grids, such as the ones used by the high energy physics community, are used to share vast amounts of data across geographic locations. However, interactions with grid data are generally limited by the interfaces provided by the corresponding grid’s infrastructure. The standardization of grid interfaces is one way to expand the reach of grid data seamlessly for users as well as to broaden the set of exploitable grid tools. This in turn enables new collaboration possibilities. The Open Grid Forum has created standards related to accessing grid data. In our work, we explore the usability of two OGF standards, RNS and ByteIO, to enable access to data resources residing in EGEE grids. Grids that implement the RNS specification can address named entities in other grids while grids that implement the ByteIO specification can manipulate data associated with named resources in other grids. Data management functionality in EGEE grids was developed before these specifications were created. As such, implementing RNS and ByteIO for EGEE grids is a test of whether these standards can be applied to an existing grid infrastructure. Through the development of the SNARL and SABLE web services, we demonstrate that such an implementation is possible and measure its performance ramifications. These services expand the access to EGEE grid data by enabling interoperability with other standard compliant grid infrastructures.

## I. INTRODUCTION

INTEROPERATION rarely happens by accident – it must be engineered. Contemporary grid infrastructures do not interoperate. Thus, it is impossible for users of different grid infrastructures to collaborate directly using their own grid software. Instead, users can learn to use new clients to communicate with other grids or they can choose to collaborate outside of the grid environment. Both of these alternatives are much less efficient and convenient for basic collaborations involving grid data than simply utilizing the native grid environment.

One method of enabling direct communication between grid infrastructures is to have the grid software stacks interoperate transparently to the client. In general there are two techniques used to make two software stacks interoperate. One approach is to create a set (or layer) of “glue” functions that emulate one stack’s functionality by calling the other stack’s functions and vice versa. If there are  $K$  stacks, this can require up to  $O(K^2)$  interoperation layers to be built. The result is usually a fragile, lashed together mess. An alternative approach is to capture the

essence of the desired functionality in a set of standard interfaces and write front-ends and back-ends to these standard specifications. This is similar to using an intermediate representation in compilers with front-ends for each language and back-ends for each target architecture.

Over the past decade, the Open Grid Forum (OGF) [1] has been taking the second approach to developing standard interfaces for basic grid functionality. The OGF is the organization leading the global standardization effort for grid computing. The OGF community of users, developers, and vendors of grids and distributed systems has consolidated best practices into standards that describe interfaces for accomplishing common grid tasks. As different grid software stacks present interfaces to these standards, interoperability will flourish.

In order to access data in a grid there must first be a way to identify or name the data, ideally in a human friendly way. Then, once the particular file or data resource has been identified, there must be a mechanism to access the data. The OGF has developed a standardized interface for each of these steps. The ByteIO specification [2] describes a standard way of accessing the bytes of data associated with flat-file grid resources while the Resource Namespace Service (RNS) specification [3] describes a standard way of naming grid resources. Together, these two specifications can be used to enable basic data sharing functionality between standard compliant grids.

In our work, we validate the RNS and ByteIO specifications. As standards develop, it is important to also develop implementations that attempt to follow these standards. Such implementations help to ensure that the purpose of a specification does not inadvertently get lost in the often multiyear development effort. Implementations also test the ability of standards to be applied in different contexts. Incorporating standards into emerging grid systems is only one piece of the validation process. An equally important, if not more challenging, piece is determining whether a specification can be implemented and provide the desired functionality for an existing grid infrastructure. As many grid systems have developed without standards, this ability for standards to wrap legacy functionality is vital if the standards are to become widely adopted and actually unite the grid community.

In this paper, we explore the usability of the standardized interfaces described by the RNS and ByteIO specifications to enable access to data resources residing in EGEE grids [4]. We validate the ability of these specifications to be implemented for a grid infrastructure that developed before these specifications were created. We present SNARL and SABLE, two web services that provide standards-based access to EGEE grid data registered in LCG File Catalogs [5] (LFCs). The SNARL (Standards-based Naming for Accessing Resources in LFCs) service implements the RNS specification while the SABLE (Standards-based Access to Bytes of LFC Entries) service implements the ByteIO specification. We demonstrate that with some overhead these services expand the access to EGEE grid data by enabling interoperability with other standard compliant grid infrastructures.

The remainder of this paper is organized as follows. In Section 2 we present background information describing EGEE grids and the RNS and ByteIO specifications. Next in Section 3 we describe the design and development of the SNARL and SABLE web services. In section 4 we discuss interoperability testing that was performed between the EGEE grid and another grid infrastructure. Then in Section 5 we present an evaluation of the performance of our web services. Before concluding in Section 8, we discuss related work in Section 6 and future work in Section 7.

## II. BACKGROUND

By implementing the RNS and ByteIO specifications, the SNARL and SABLE services provide standards-based access to data resources in EGEE grids. Specifically, SNARL and SABLE provide access to data resources registered in EGEE grid LFCs. In this section we discuss which EGEE grid data SNARL and SABLE provide access to and present more details about the two specifications that these web services implement.

### A. Data Management in EGEE Grids

The SNARL and SABLE web services provide standard-based access to data resources registered in EGEE grid LFCs. LFCs are the file catalog technology provided by EGEE grids to facilitate access to grid data. LFCs keep track of the locations of files distributed amongst various storage elements in a grid. These files are assumed to be read-only. A unique file is registered in an LFC with a logical file name. For each logical file name, an LFC stores a mapping to the physical location of the file. If multiple replicas of one logical file exist, the locations of these replicas are registered under one logical name. Thus, LFCs can be consulted to acquire the physical locations of a given logical file. The SNARL and SABLE web services provide access to EGEE grid files that have been registered in LFCs.

### B. The RNS Specification

Accessing resources in a grid is a fundamental task. To be able to access resources, users require some means of referring to the resources. Often resources are named in a human-readable fashion that is more easily applied by users to refer to

the resources. In turn, these name maps to addresses that uniquely identify the resources. The RNS specification is concerned with the handling of such mappings. Developed by the OGF, this specification describes a standard way for mapping names to entities within a grid. If different grids follow the RNS specification, then they can understand what named entities exist in each other's grid space. This basic interoperation lays the foundation for further collaboration such as manipulating the data that may be associated with named resources.

The RNS specification is based on standards compliant pieces. One critical standard utilized by the RNS specification is concerned with how grid resources are addressed. This standard, WS-Addressing [6], describes XML elements needed to identify web service endpoints. The resulting endpoint reference (EPR) is a handle to any referenceable grid resource, processor, or other entity to which a web service message can be addressed. The RNS specification effectively describes a standard means for mapping human readable names to these EPRs, thus enabling named entities to be uniquely addressable.

In specifying a means of handling name-to-resource mappings, the RNS specification describes a set of five basic operations associated with such mappings: *add*, *remove*, *list*, *query*, and *move*. An *add* operation can be used to create a new entry in the namespace. A *remove* operation can be used to delete an existing entry from the namespace. A *list* operation can be used to list the subentries associated with a directory in the namespace. A *move* operation can be used to rename an existing entry. A *query* operation can be used to acquire additional properties associated with an entry. The five operations defined by the RNS specification makeup the standardized interface for performing basic interactions on named grid entities.

### C. The ByteIO Specification

The ByteIO specification describes a set of operations that enable POSIX-like access to grid data resources i.e. read and write. Two different interfaces for interacting with data are presented. The RandomByteIO port type describes operations for dealing with data resources that are accessible directly in a random, session-less manner. Alternatively, the other port type, StreamableByteIO, describes operations for dealing with data resources in a stream-like, session-based manner.

The ByteIO specification additionally aims at enabling more efficient bulk data transfer. Complying with the specification does not require supporting a more complicated transfer mechanism than SOAP where the requested data is included as a text element in the SOAP message. However, the specification also provides the means to use a more efficient transfer mechanism like DIME or MTOM.

Like the RNS specification, the ByteIO specification is based on other standards. In particular, ByteIO also uses WS-Addressing. This web service standard for addressing is employed to specify the resource a ByteIO request is aimed at.

The ByteIO specification describes six operations for POSIX-like access to grid data resources. Four operations,

*read*, *write*, *append*, and *truncAppend*, describe the random data access provided by the RandomByteIO port type. For these operations, the client specifies the starting offset and size of the blocks of data to be read or written. A *read* operation can be used to request to read a chunk of data from a resource. A *write* operation can be used to specify a chunk of data to be written in a resource. An *append* operation can be used to specify a chunk of data to be appended to the end of a resource. Finally, a *truncAppend* operation can be used to first truncate a portion of a resource and then also optionally append a chunk of data to the end of that resource. The remaining two ByteIO operations, *seekRead* and *seekWrite*, embody the session-like semantics of the StreamableByteIO port type. Both of these operations assume a session has been opened on a data resource. A *seekRead* operation can then be used to read a chunk of data from the resource while a *seekWrite* operation can be used to write a chunk of data to the resource. The six operations defined by the ByteIO specification provide a standardized interface for performing basic manipulations on bytes of data associated with flat-file grid resources.

### III. DEVELOPMENT

The development process of the SNARL and SABLE web services consisted of four distinct phases. First, we set up a web services framework to provide the infrastructure for the services. Next, we used this framework to generate skeleton web services compliant with each specification. We then created clients to test the operation of our services. Lastly, we implemented the functionality associated with each specification by using the appropriate API calls. We discuss each of these phases in this section.

#### A. Axis2/C, A Web Services Framework

We have used Apache Axis2/C [7] to create the SNARL and SABLE web services. Axis2/C is a web services engine used to provide and consume web services. We decided to use this framework for two main reasons. Firstly, Axis2/C provides a web services implementation in C and the LFC API is also written in C. Secondly, Axis2/C includes a variety of WS-\* specification implementations such as WS-Addressing and WS-Security. Since both the RNS and ByteIO specifications utilize WS-Addressing, it was very useful to have this support and not have to implement it separately. Additionally, the WS-Security support of Axis2/C could be used in future work to help incorporate security into our services.

#### B. Contract-First Web Service Creation

To implement SNARL and SABLE, we have used a contract-first approach. In this approach, first a contract in the form of WSDL and XML schema files was created specifying what each web service was to offer. Then the business logic was implemented to provide what was promised in the contract. This approach was well suited for our development as WSDL files describing the desired interfaces are included in the RNS and ByteIO specifications. We tailored these

WSDL files for the functionality we wanted SNARL and SABLE to provide and then used the WSDL files to create skeleton services.

We created the skeleton services for SNARL and SABLE by running an Axis2/C code generation tool, WSDL2C, on the WSDL files. This command line tool processes a given WSDL file and produces a skeleton web service written in C based on that WSDL. The tool maps schema types in the given WSDL to native C types and structures. These resulting services are skeletons as the specification related functionality of each service must be filled in to create a fully operational service. The SNARL skeleton service generated by the WSDL2C tool consisted of 89k LOC while the SABLE skeleton service consisted of 77k LOC.

#### C. Client Testing

To test the SNARL and SABLE web services, we created Axis2/C clients to consume each service. The basic job of any client is to prepare a payload, send it to a service, receive a response, and process it. Each of our test clients prepares a payload consisting of one of the standardized RNS or ByteIO operation requests. This request is then sent to the appropriate service. Upon receipt of the response message, the client displays the response so that its format can be analyzed for conformation to specification.

#### D. Implementing SNARL's Functionality

To be compliant with the RNS specification, the SNARL service provides the interfaces described by the specification. The SNARL service translates each RNS operation request into the appropriate LFC API calls needed to carry out the request. The result is then packaged according to the specification and a response message is returned to the requesting party. Table I shows the main LFC API calls corresponding to each RNS operation request.

TABLE I  
API CALLS FOR RNS OPERATIONS

RNS Operation	LFC API Call
Add	lfc_creat
List	lfc_readdir
Remove	lfc_unlink
Query	lfc_stat
Move	lfc_unlink + lfc_creat

The SNARL service translates each RNS operation request into the LFC API call(s) needed to carry out the requested operation.

In creating a correspondence between LFC entries and RNS resources, the question arose of whether to depict replicas as named entities in the RNS namespace. Providing users with a view of just logical files is sufficient to represent what unique data exists in LFCs. However, it is the physical files associated with each logical file entry that have more detailed information associated with them such as creation time, size, and last access time. We decided that for our initial SNARL service implementation we would depict replicas as named entities in the namespace. Thinking about how to handle replicas brought out a question not directly addressed by the interface provided by the RNS specification.

### E. Implementing SABLE's Functionality

To be compliant with the ByteIO specification, the SABLE service provides the interfaces described by the specification. However, the SABLE service does not implement all of the ByteIO operations. The specification describes two port types that address different use cases. We decided that the sessionless, random file mode of access of the RandomByteIO port type would be well suited for accessing files registered in LFCs. Thus, the current SABLE service only implements the ByteIO operations related to the RandomByteIO port type. In the future, the SABLE service may be extended with implementations of the two remaining operations related to the StreamableByteIO port type.

TABLE II  
API CALLS FOR BYTEIO OPERATIONS

ByteIO Operation	GFAL API Call
All	gfal_open
Read	gfal_lseek + gfal_read
Write/Append	gfal_lseek + gfal_write
TruncAppend	gfal_creat + Append

The SABLE service translates each ByteIO operation request into the corresponding GFAL API call(s) needed to carry out the requested operation.

Table II shows the main API calls used to carry out each ByteIO operation request. The SABLE service translates ByteIO requests into Grid File Access Library (GFAL) API calls. GFAL is one mechanism supplied by the EGEE grid middleware for accessing files on storage elements. GFAL provides a POSIX interface for I/O operations. As the ByteIO specification also aims at enabling POSIX-like access to grid data resources, the GFAL library was a natural match for implementing the ByteIO operations.

Since LFCs catalogue read-only data, we had to decide whether the *write*, *append*, and *truncAppend* operations should be valid requests on LFC resources. We decided that for the *append* and *truncAppend* operations, the SABLE service should throw faults indicating that a write is not permitted on LFC data. Users accessing data associated with LFC entries should not be allowed to append or truncate read-only resources. The decision was not as clear cut for the *write* operation. LFC resources can be characterized as being write-once, read-many. Thus, an initial write to a resource is semantically correct. However, for our initial SABLE implementation, we decided that the *write* operation should also throw a fault as the use case we are currently addressing deals with the sharing of already existing data and not the creation of new data.

## IV. INTEROPERABILITY TESTING

To test the ability of the SNARL and SABLE web services to provide access across grid infrastructures to EGEE grid data registered in LFCs, we conducted interoperability testing with a Genesis II grid [8]. The Genesis II grid platform provides an infrastructure for compute and data grids. Genesis II was created at the University of Virginia and has developed around standards. It is one of three grid infrastructures that contain implementations of both the RNS and ByteIO specifications.

In this section, we describe the setup and results for interoperability tests performed with Genesis II.

### A. Setup

Two machines were used for the interoperability testing. Both machines resided inside the CERN internal network to avoid dealing with security as at this point the SNARL and SABLE services do not support security. On one machine, an LFC was set up along with an instance of each of the SNARL and SABLE services. On the second machine, a Genesis II client was set up with all of its security mechanisms also turned off. RNS and ByteIO requests were sent between the two machines for the interoperability testing. Figure 1 illustrates this setup.

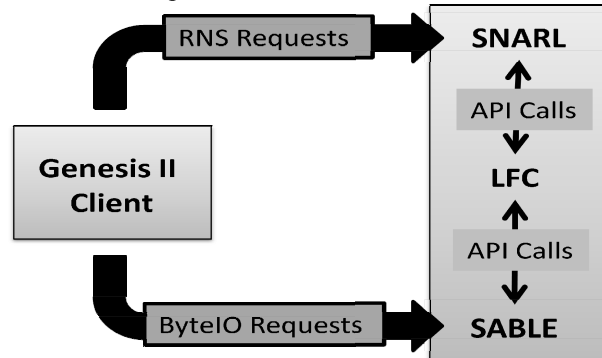


Fig. 1. For the interoperability tests, a Genesis II grid client sent RNS and ByteIO requests to the SNARL and SABLE web services. The web services in turn made the appropriate API calls to the LFC to process the requests. The results were then sent back to the Genesis II client.

### B. Results

The interoperability tests were carried out successfully. Three of the five RNS operations were tested. The *query* and *move* RNS operations could not be tested as the Genesis II implementation of these operations deviates from the RNS 1.0 specification that SNARL implements. RNS *list*, *add*, and *remove* requests were sent by a Genesis II client to the SNARL service. Also, Genesis II sent a ByteIO *read* request to the SABLE service. SNARL and SABLE successfully processed each request, made the appropriate API calls, and returned properly formatted responses messages that were understood by the Genesis II client. In parallel to the interoperability testing, command-line tools were used to confirm the correctness of the responses and that each operation had occurred as expected.

These basic interoperability tests confirmed that the SNARL and SABLE services are able to provide access across grid infrastructures to data registered in LFCs. The tests also underlined the orthogonal issue of security that needed to be addressed to make interoperability between different grid systems possible. In the next section, we discuss the performance ramifications of using web services to access LFCs as compared to traditional modes of access.

## V. PERFORMANCE

A series of performance tests were conducted to determine the overhead associated with using the SNARL and SABLE services to access EGEE grid data registered in LFCs. These

tests focused on measuring the performance impact due to the web services framework. Accessing an LFC via API calls and command-line tools was compared against accessing the LFC via the standardized interface provided by the SNARL and SABLE services. The SNARL and SABLE services were invoked using calls from an Axis2/C client and a Genesis II client. The performance results are summarized for each service in Tables III and IV.

#### A. Setup

The performance tests were carried out on a machine running Scientific Linux CERN 4 with an Intel Pentium 4 2.8 GHz processor and 1 GB of memory. The machine was set up with an LFC as well as an Axis2/C server running an instance of the SNARL and SABLE services. For the cross grid infrastructure tests, a Genesis II client was also set up on the machine. Since all the tests were ran on just one machine, any overhead associated with location or network performance was minimized. In this way, we concentrated on testing the performance impact associated with the different modes of access.

Three series of performance tests were run. First, we measured how long it took to directly access an LFC using API calls and command-line tools. To make the API calls, standalone C programs were created that consisted of the same API calls as used in our web services. Next, we made calls using our Axis2/C test clients to time how long it took to access an LFC via SNARL and SABLE. Lastly, we made the request calls using a Genesis II client to compare how long it took to access an LFC via SNARL and SABLE from a different grid infrastructure.

TABLE III  
TIME TO LIST CONTENTS OF LFC DIRECTORIES

Directory Size:	LFC API	Command-line	Axis2/C Client	Genesis II Client
<b>Small</b> (10 entries)	1.49s	0.42s	1.94s	1.98s
<b>Medium</b> (100 entries)	11.73s	0.47s	15.92s	17.16s
<b>Large</b> (1000 entries)	2m 0.37s	0.71s	2m 1.11s	2m 5.05s

The performance of using the SNARL service to perform RNS *lists* to list the contents of an LFC directory is compared against other listing mechanisms that do not involve web services. SNARL is engaged by using an Axis2/C client as well as from a different grid infrastructure by using a Genesis II client. The table presents average times to list the contents of an LFC directory using the different listing mechanisms. Lists are performed on three different directory sizes.

TABLE IV  
TIME TO READ CONTENTS OF FILES REGISTERED IN LFCs

File Size:	GFAL API	Command-line	Axis2/C Client	Genesis II Client
<b>Small</b> (1 KB)	2.54s	2.60s	2.55s	3.49s
<b>Medium</b> (1 MB)	2.83s	3.61s	2.93s	3.64s

The performance of using the SABLE service to perform ByteIO *reads* to read files is compared against other file access mechanisms that do not involve web services. SABLE is engaged by using an Axis2/C client as well as from a different grid infrastructure by using a Genesis II client. The table presents average times to read the contents of file registered in an LFC using the different file access mechanisms. Files with two different sizes are read.

#### B. SNARL Performance

To test the performance of the SNARL service, we measured the length of time to list the contents of LFC directories. We created test directories with 10, 100, and 1000 entries. Due to connection timeouts, larger directory sizes could not be listed until an optimization like caching is utilized by the SNARL service.

The amount of time it took to make a RNS *list* request on each of these directories was measured using a local Axis2/C client as well as a Genesis II client. We compared the performance of making RNS *list* requests to the SNARL service against two non-web service modes of access: via LFC API calls and command-line tools. The *lfc-ls* command-line tool was chosen for this comparison as it performs a list on an LFC directory. These performance tests were repeated multiple times. The average timing results are presented in Table III.

#### C. SABLE Performance

To test the performance of the SABLE service, we measured the time to read the contents of files registered in LFCs. 1 KB and 1 MB files were used in testing. Since at this time SABLE includes the data being read as a text element in the SOAP message, we did not test with larger files. Such files should be sent as DIME, MIME, and MTOM attachments to avoid SOAP message size restrictions and enable more efficient bulk data transfer. The ByteIO specification supports the use of these transfer mechanisms.

The amount of time it took to read a file via ByteIO *read* requests was calculated when using an Axis2/C client as well as a Genesis II client. We compared the performance of reading files via ByteIO *read* requests against two non-web service modes of file access: via GFAL API calls and command-line tools. The *lfc-cp* command-line tool was chosen as it uses GFAL. The tool was used to copy a grid file to a local directory. These performance tests were repeated multiple times. The average timing results are presented in Table IV.

#### D. Timing Analysis

We timed the performance of using API calls and command-line tools to list directories and read files contents to get a baseline for comparing the performance of the SNARL and SABLE services. For reading files, we expected to find that using API calls would be slightly faster than using the command-line tools as command-line tools generally require extra load time. Indeed, we found that using a command-line tool for reading took about 20% longer. However, this was not the case for listing files. The command-line tool took one order of magnitude less time than the API calls. This is because the tool simply lists the entries in a directory and does not stat them to determine if they are files or directories. The SNARL service stats all entries being listed so that it may create an appropriate EPR for each entry. Our timing tests showed that this has tremendous performance ramifications. Caching would be one way to avoid these costs. A temporary EPR could be created and returned to the user before the stat call is completed. Once cached, the stat information could be

used to quickly generate the appropriate EPR when a list call for a cached entry is repeated. This optimization would enable the listing of larger directories in a matter of seconds instead of minutes and solve the issue of connection timeouts that typically occur after several minutes.

We compared the performance of using API calls and command-line tools to request calls made via the Axis2/C client to determine how much longer it takes to use a web service interface to accomplish the same functionality. Web services are notoriously slow. We measured that for small and medium directory listings, using web services is around 35% slower. For listings of larger directories as well as reading files, using web services is less than 10% slower. As long as the list operation only takes on the order of seconds, the performance loss associated with using web services might be tolerated by users. This will be the case if caching is implemented for the SABLE service.

Lastly, we measured the performance when making requests to SNARL and SABLE from a different grid system client. We expected that calling the web services using our Axis2/C client would take less time than calling the web services using a Genesis II client due to extra processing performed by the Genesis II client. The Axis2/C client is very basic and entirely tailored to testing SNARL and SABLE while the Genesis II client is much more sophisticated as the main interface to the Genesis II grid system. We found that RNS list requests are on average 10% slower when made via the Genesis II client than when made using the Axis2/C client. ByteIO read requests are on average 30% slower. This overhead is associated with processing the Genesis II client performs on the responses to the RNS and ByteIO requests.

## VI. RELATED WORK

Enabling data sharing directly between grid infrastructures can be classified as one solution to the problem of remote data access. This general problem has been around as long as networks have existed. In this section, we take a closer look at some of the more popular solutions and present their advantages and disadvantages.

NFS [9] is the standard Unix solution for accessing files on remote machines within a LAN. With NFS, a disk on a remote machine can be made part of the local machine's file system. Accessing data from the remote system now becomes a matter of accessing a particular part of the file system in the usual manner. While NFS is easy to understand and does not require applications to be modified to access files that have been NFS mounted, it does not scale to WAN environments nor between organizational units. This is precisely what we enable in our work by facilitating interoperability between two different grid infrastructures.

FTP [10] has been the tool of choice for transferring files between computers since the 1970s. FTP is a command-line tool whose commands resemble Unix. As such, FTP is relatively easy to use and comes installed virtually everywhere. However, FTP is a copy-based file sharing mechanism that requires copying an entire file. SNARL and SABLE provide the ability to access entire as well as chosen

portions of files. Additionally, using FTP requires at least knowing the machine name and part of the directory structure where files are to be copied. Users must also have an account on the machine where files are being copied which gives them more privileges than just file access. While anonymous FTP can be used to alleviate the need for accounts, this option provides anyone access to the anonymous ftp directory.

SCP and SFTP belong to the SSH [11] family of tools. SCP is basically a secure version of the Unix RCP command that can copy files to and from remote sites, whereas SFTP is a secure version of FTP. Both are command-line tools. The syntax for SCP resembles the standard Unix CP command with a provision for naming a remote machine and a user on it. Likewise, the syntax and usage for SFTP resembles FTP. The benefits of using SCP or SFTP are that their usage is similar to existing tools. SCP and SFTP also have the advantage of providing encrypted password and data transfer. Data encryption is something the current versions of the SNARL and SABLE web services do not provide. However, these are not inherent limitations. The future plan is to incorporate encryption and security features into the services.

GridFTP [12] is a tool for transferring files built on top of the Globus toolkit. GridFTP can be used to transfer files from one machine to another, similar to the manner in which SCP and SFTP are used. GridFTP also encrypts passwords and data. However, it is more likely than with FTP or the SSH copy tools that one of the parties will have to install the Globus toolkit in order to use GridFTP. GridFTP does have the advantage of offering higher performance by providing concurrent access to data by design. This is something not currently possible with the ByteIO specified interfaces for transferring data. There is no reason, however, that future versions of the specification will not be expanded to also provide better performance via a standardized interface for concurrent data access.

## VII. FUTURE WORK

The current SNARL and SABLE web services provide basic implementations of the RNS and ByteIO specifications. There are many enhancements that could be made to improve these services. Security is perhaps the most important addition that would make the services more useful. We discuss this and other possible improvements in this section.

### A. Incorporating Security

To make SNARL and SABLE usable in production-level environments, security mechanisms need to be incorporated into the services. Currently, the services do not provide any security measures. This is because there is no agreement on one security mechanism to use for interoperability communications across grid infrastructures. As such security contexts are decided, the SNARL and SABLE web services can be augmented appropriately. Conveniently, the Axis2/C framework provides built in support for the WS-Security standard that could be utilized to include security information in our web services.

### B. DIME, MIME, and MTOM Support

Currently, the SABLE web service utilizes a simple transfer mechanism that is not very efficient. Requested data that is being read is included as a text element in the SOAP message. To enable more efficient bulk data transfer, DIME, MIME, and MTOM support could be implemented.

### C. RNS 1.1

The SNARL service implements the first version of the RNS specification. Revisions to the RNS specification are already in progress. A new version of the specification is scheduled to appear later this year. We hope that our experiences with implementing the SNARL service may help shape the development of this update to the RNS specification. To remain useful from the interoperability perspective, the SNARL service will need to be updated to comply with the next version of the specification.

### D. StreamableByteIO

A natural extension to the SABLE service would be to implement the operations related to the StreamableByteIO port type of the ByteIO specification. This port type, consisting of two ByteIO operations, assumes a stream-like, session-based interaction with data resources. To implement this port type, the SABLE service would need to be extended with the two operations, *seekRead* and *seekWrite*.

### E. Scalability

Currently, the performance of SNARL and SABLE to process one client request at a time has been evaluated. To make the services usable in a production-level environment, the services must efficiently handle multiple requests. This is an orthogonal issue to ensuring the web services provide the specified functionality but an essential feature for servicing any often accessed LFCs.

## VIII. CONCLUSIONS

It is impossible for users of different grid infrastructures to collaborate directly using their own grid software unless the grids share common interfaces or special “glue” software is written. To address this interoperability problem, the grid community has developed various standards that enable basic interactions. The RNS and ByteIO specifications developed by the OGF provide interfaces that can be used for standardized data sharing. In our work, we have demonstrated that these standards can be implemented and successfully utilized to share the data of an existing grid system, the EGEE grid, whose data management functionality developed before these specifications were created. We have shown that our implementations, the SNARL and SABLE web services, expand the access to EGEE grid data by enabling interoperability with other standard compliant grid infrastructures. Our work illustrates that it is possible to utilize standards to unite old and new grid infrastructures and enable data sharing. The cost to users for this more direct and seamless data sharing is the performance overhead related to using a framework based on web services.

## ACKNOWLEDGMENT

We would like to acknowledge Mark Morgan for his help with implementing the specification functionality and working with Genesis II. We would like to acknowledge Akos Frohner for his help with using the EGEE grid infrastructure.

## REFERENCES

- [1] "Open Grid Forum," <http://www.ogf.org>.
- [2] M. M. Morgan, "ByteIO specification," Global Grid Forum. GFD-86, 2006.
- [3] M. Pereira, O. Tatebe, L. Luan, T. Anderson, and J. Xu, "Resource Namespace Service specification," Global Grid Forum, 2006.
- [4] E. Laure, et al. "Middleware for the next generation Grid infrastructure," *Computing in High Energy Physics and Nuclear Physics*, 2004.
- [5] L. Abadie, P. Badino, J. P. Baud, J. Casey, A. Frohner, G. Grosdidier, S. Lemaitre, G. Mccance, R. Mollon, K. Nienartowicz, D. Smith, P. Tedesco, "Grid-enabled standards-based data management". *IEEE Mass Storage Conference*, 2007.
- [6] M. Gudgin, M. Hadley, and T. Rogers, "Web services addressing 1.0 – Core," World Wide Web Consortium, 2006.
- [7] "Apache Axis2/C," <http://ws.apache.org/axis2c>.
- [8] M. M. Morgan and A. S. Grimshaw, "Genesis II – standards based grid computing," *Seventh IEEE International Symposium on Cluster Computing and the Grid*, 2007.
- [9] "Network File System (NFS) version 4 protocol," Internet Engineer Task Force, RPC #3530, 2003.
- [10] File Transfer Protocol (FTP), Internet Engineer Task Force, RPC #959, 1985.
- [11] T. Ylnen, "SSH---secure login connections over the internet," *Proceedings of the Sixth USENIX Security Symposium*, 1996, pp. 37–42.
- [12] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, "GridFTP: protocol extensions to FTP for the Grid," Global Grid Forum GFD-RP, 2003.