oTN-2008-08 openlab Technical Note

The SNARL Service: Standards-based Naming for Accessing Resources in an LFC

By Karolina Sarnowska
Supervisors: Erwin Laure, Andrew Grimshaw, and Akos Frohner

16 August 2008
Version 2
Distribution:: **Public**

1 Introduction

This report presents the SNARL (**S**tandards-based **N**aming for **A**ccessing **R**esources in an **L**FC) service. SNARL is a web services implementation of one of the Open Grid Forum (OGF) [1] specifications. Specifically, SNARL is an implementation of the first version of the Resource Naming Service (RNS) [2] specification for the LCG File Catalog (LFC) [3]. The creation of the SNARL service occurred as a part of the 2008 Openlab Summer Student Program at CERN.

This report is divided as follows. First, we present background information describing the grid software interoperability problem, the Open Grid Forum, the Resource Naming Service specification, and the SNARL service. Next, in the main section of this report, we describe the design and development of the SNARL service. We end with a discussion of future work and conclusions.

In this section, we present the background concepts that the SNARL service is based upon. To put the relevance of the SNARL Service into context, we begin with a discussion of the grid software interoperability problem. We then discuss how the mission of the OGF has been to strive to find solutions to this problem. Finally, we discuss the specific interoperability area that is addressed by the RNS specification and implemented by the SNARL service.

1.1 Grid Software Interoperability

The high-level problem area that has motivated the creation of the SNARL service is the problem area of grid software interoperability. It is difficult if not impossible for users in different scientific communities using different grid infrastructures to collaborate directly using their own grid software. Instead users must download and install specialized clients to communicate with other grids or collaborate entirely outside their grid environment. Both of these alternatives are much less efficient for basic collaborations involving grid data than simply utilizing the native grid environment. The root of the problem is that grid software is not generally interoperable.

1.2 Open Grid Forum

There is no direct way for different grid infrastructures to communicate and understand each other unless the grids share some common interfaces. Over the past years, the Open Grid Forum (OGF) has been attempting to help address this issue. The OGF is the organization leading the global standardization effort for grid computing. During OGF events, users, developers, and vendors gather together to discuss their experiences with grids and distributed systems. In their discussions, they attempt to consolidate their best practices into grid standards that specify sets of interfaces for accomplishing common grid tasks. Through these efforts, the OGF strives to address the grid software interoperability issue. If different grids follow the same standards, then it will be possible for grids to directly interoperate. In this way, the OGF hopes to help drive the evolution and adoption of applied distributed computing environments.

1.3 Resource Naming Service Specification

The ways in which grid software could interoperate is as varied as the complexity of a grid's infrastructure. The goal of creating grid standards is not to embody every possibility but instead to describe a standard means of achieving interoperability for basic functionality. Over the past years, many standards have been developing via the OGF to address different types of interoperability. Since these standards are meant to be implementation independent, they are described in terms of operations provided by web services. One such standard is described by the Resource Naming Service (RNS) specification. This specification deals with the access to resources and their naming.

Accessing resources in a grid is a fundamental task. To be able to access resource, users require some means of referring to the resources. Often resources are named in a human-readable fashion that is then used by users to refer to them. In turn, this name maps to some meaningful address that uniquely identifies the resource. The RNS specification is concerned with the handling of such mappings. It specifies 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 is a basic interoperation that lays the foundation for further collaboration such as manipulating the data that may be represented by named resources.

The RNS specification is based on other OGF standards. To make interoperability possible, the specification needs to be built from standards compliant pieces. One really important standard utilized by the RNS specification is concerned with how grid resources are addressed. This standard, WS-Addressing [4], describes XML elements needed to identify web service endpoints. The resulting endpoint reference (EPR) is basically a handle to any referenceable grid resource, processor, or other entity to which a web service message can be addressed. The RNS specification describes a standard means for mapping human readable names to these EPRs.

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. The add operation adds a new entry into the namespace. The remove operation deletes an existing entry from the namespace. The list operation lists entries associated with a given name in the namespace. The move operation provides a means of renaming an existing entry. The query operation is used to get the properties associated with a particular name. In this way, the RNS specification provides a standard mechanism for performing basic interactions on named entities with the namespace of a grid.

1.4 LCG File Catalog

SNARL is a web service that implements the RNS specification for the LCG File Catalog. We now provide some background about the LFC. The LFC is one means of tracking and facilitating access to data in EGEE grids. As the name indicates, the LFC is a file catalog keeping track of the locations of files in the grid. Files that are distributed amongst various storage elements in a grid can be registered in an LFC. Each unique file is given a unique logical file name. For each logical file name, a mapping is created to the actual address of the file. If multiple replicas of one logical file exist, these replicas are registered under one logical name. Thus, for each logical file name entry registered in the LFC, there can be one or more physical files associated with that logical name. The LFC stores a list of all logically different file names along with the mapping between logical file names and the addresses needed to access the data.

1.5 The SNARL Service

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 effort involved in standards development. In our work, we strive to validate the RNS specification by creating an implementation for LFC's in EGEE grids. The Standards-based Naming for Accessing Resources in an LFC (SNARL) service is the first implementation of RNS for a pre-existing grid system. The University of Virginia's Genesis II [5] grid and a grid at the University of Tsukuba are the only other existing implementations of the RNS specification. We hope to provide useful feedback to the OGF and the RNS working group through this work.

2 Development

In this section, we describe the design and development of the SNARL service. We begin with a general overview of the SNARL service and then proceed with a detailed discussion of the development of the service. This process can be divided into distinct phases. First, we setup a web services framework to provide the infrastructure for a web service. Next, we used this framework to generate a skeleton web service compliant with the RNS specification. To test the operation of the SNARL service, we created a local Axis2 client. We then implemented each RNS operation by filling in the skeleton service with the appropriate business logic. Finally, once everything was working, we performed a set of interoperability tests with a different grid.

2.1 SNARL Service Overview

The SNARL service is a web service that implements the RNS specification for LFCs. The SNARL service basically serves as a translator between LFC and RNS calls. This enables other grid implementing the RNS specification to contact the LFCs in EGEE grids and communicate via the SNARL service. An illustration of this interaction is provided in Figure 1.

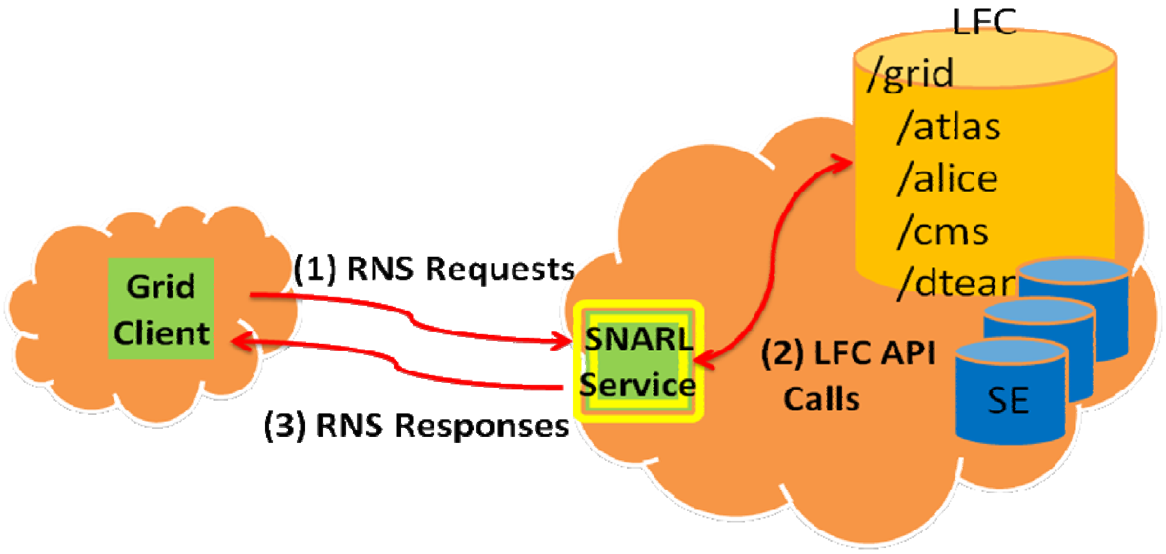

Figure 1. A depiction of the SNARL service in relation to other grid components. The SNARL service understands RNS calls and translates them into LFC API calls to answer RNS operation requests. Here, the SEs represent different grid storage elements hooked into a grid. The orange bubbles represent different grid infrastructures. The two grids are directly communicating with RNS calls.

2.2 Axis2/C, A Web Services Framework

We have used Apache Axis2/C [6] to create the SNARL web service. Apache Axis2/C is a web services engine used to provide and consume web services. It supports both the SOAP and REST style of web services.

We decided to use Axis2/C for two main reasons. Firstly, Axis2/C provides a web services implementation in C and the LFC is written in C. Secondly, Axis2/C supports a variety of WS-* specification implementations including WS-Addressing and WS-Security [7]. Since the RNS specification is based on WS-Addressing, it was very useful to have this support and not have to implement it separately. Additionally, we hope to use the WS-Security support of Axis2/C in future work that will incorporate security into the SNARL service. In this section we briefly summarise our experiences with setting up and running an Axis2/C server.

2.2.1 Running an Axis2 Server

Setting up the Apache Axis2/C web service environment was the first step in creating the SNARL service. A free copy of binary release version 1.4.0 was downloaded from the Apache Axis2/C website. The online installation instructions for setting up Axis2/C under Linux were used to get an Axis2/C server running. Initial test runs with sample client and service code were successful. In these tests and throughout the development of the SNARL service, we have used the simple axis server provided with the Axis2/C framework. This server is started with the following command.

```
AXIS2_HOME/bin> ./axis2_http_server
```

2.3 Service Generation with the WSDL2C Tool

The WSDL2C tool is a code generation tools provided by the Axis2/C framework. This is a command line code generation tool used to create a skeleton web service in C. The tool processes a given WSDL file and produces a skeleton web service based on that WSDL. Once a skeleton web service is created, the business logic for the service must be filled in to create fully functioning service. In this section, we describe how the SNARL service skeleton was created using the Axis2/C WSDL2C tool.

2.3.1 WSDL Files for SNARL

There are two different approaches to implementing a web service: code-first or contract-first. In the code-first approach, a developer first codes the business logic and then exposes the implementation as a web service. In the contract-first approach, a contract in the form of WSDL and XML schema files is created specifying what the web service is to offer and then the business logic is implemented to provide what was promised in the contract. We used the contract-first approach in creating the SNARL service as the RNS specification precisely outlines using WSDL what an RNS compliant service is to offer.

The main WSDL file for the SNARL service was created using the RNS specification and the Genesis II RNS WSDL. We started off with a copy of the Genesis II RNS WSDL. This file was then modified to specify all the needed elements to be compatible with the Axis2/C WSDL2C tool. This included adding appropriate <service> and <binding> tags to the WSDL file. The resulting WSDL file actually specifies a set of WSDL and XML schema files needed to describe an RNS compliant service.

2.3.2 Axis Data Binding

We chose to generate the SNARL web service skeleton with Axis Data Binding (ADB) support. A data binding framework such as ADB maps XML schema constructs to programming language objects. The WSDL2C ADB support maps schema types in the given WSDL to native C types and structures. Thus, in the SNARL service code we are able to program with simple C types instead of dealing with XML.

2.3.3 Running the WSDL2C Tool

Once the WSDL and XML schema files were appropriately setup, we were able to automatically generate a web service skeleton for the SNARL service using the WSDL2C tool. The following command-line options were used.

```
  AXIS2_HOME/bin/tools/WSDL2C> WSDL2C.sh -uri RNS.wsdl -u -ss -sd -d adb -o
SNARL_Service_Code
```

We describe the function of each command line option that was utilized. The '-u' option specifies that each structure should be put in separate files. The '-ss' specifies that server side code is to be generated. The '-sd' option specifies that the service descriptor file which contains information about the service name, operations and other metadata required for service deployment in Axis2/C should be created. The '-d' option with parameter *adb* specifies that Axis data binding should be used. The '-o' option specifies the output directory.

2.3.4 WSDL2C Tool Output

The WSDL2C tool generates many files that describe the skeleton for the web service specified by the given WSDL file. However, there is only one file that needs to be edited to add the actual business logic for the service. This is the `axis2_skel_<service_name>.c` file. Inside this file, the locations where the business logic is missing are clearly marked with the following comment.

/ TODO fill this with the necessary business logic */*

As expected, for the SNARL service skeleton that was generated by the WSDL2C tool, this comment appeared in five locations. Each of these locations corresponded to where the business logic for each of the five RNS operation should be implemented.

2.3.5 Compiling the Service Skeleton

After creating the SNARL service skeleton, our next goal was to run the empty service. The first step in this process was to compile the code automatically generated by the WSDL2C tool. For this purpose, a build script is utilized. This script is also automatically generated by the WSDL2C tool along with all the service code. However, executing the build script produced several compilation errors. The bugs in the generated code concerned undeclared identifiers. We were able to determine the source of each problem and modify the code such that the service skeleton compiled. These bugs and solutions were reported to the Axis2/C development group. As a result, the bugs have been fixed in the current version of the tool.

2.3.6 Deploying an Axis2 Service

Once the SNARL service code was compiling, we were able to successfully deploy the service in Axis2. To make any Axis2 service available, it must first be deployed. To deploy a service, a new folder named after the service is created under the `AXIS2_HOME/services` directory. Into this new folder, a description of the service (`service.xml`) and the shared library file (`*.so`) are placed. To verify that a service has been properly deployed, one can start-up an Axis2 server and browse the list of deployed services using a web browser. The service list is located at `http://localhost:<port_number>/axis2/services`. The default port number for the Axis2 simple server is 9090.

2.4 Creating an Axis2 Client

To test the SNARL service locally, we needed to create a client to consume the web service. We created an Axis2 client to test the operation of the SNARL service throughout development. This client was used to send a variety of RNS requests as per the RNS specification. In this section, we describe the creation of a client in Axis2.

2.4.1 SNARL Client Overview

The basic job of a client is to prepare a payload, send it to a service, receive a response, and process it. For testing the SNARL service, the SNARL client prepares a payload consisting of one of the RNS operation requests. This request is sent to the SNARL service. The response message from the service is then displayed. The request messages sent by our client were formed as per the RNS specification while the response messages were tested for conformation to the RNS specification.

2.4.2 Axis2 Client Components

The Axis2 documentation describes five general steps that need to be fulfilled in creating an Axis2 client. We list and briefly describe each of these steps below.

1. *Create the environment to be used by the client:* The environment instance encapsulates the memory allocator, error handler, and logging and threading mechanisms. Each function in Axis2/C takes a pointer to this instance.
2. *Create and set an options instance:* An options instance is used to set options such as the endpoint address of the service to be consumed by the client.
3. *Create and set options to service client instance*: The service client instance is set with the options to be used by the service client.
4. *Send request and receive response*: The axis2_svc_client_send_receive method can be used to invoke the send receive operation on the service client instance. The send receive operation takes the request payload as an AXIOM node and returns the response payload as an AXIOM node. This node can then be processed accordingly.
5. *Process request*

2.4.3 Axis Object Model

AXIOM stands from **AXI**s **O**bject **M**odel. It refers to the XML infoset model used by Apache Axis2. The XML infoset is the information encoded inside an XML file. The goal behind AXIOM is similar to the goal behind XML models such as DOM and JDOM that represent the XML infoset in a language specific manner that makes it more convenient to manipulate programmatically. Two versions of AXIOM exist: AXIOM/C and AXIOM/Java. The payload for the client messages is created using AXIOM. Similarly, the response payload is represented as an AXIOM node. To create a client for consuming the SNARL service, we had to become familiar with AXIOM to create requests and process the responses correctly.

2.4.4 Engaging WS-Addressing

Axis2/C provides an implementation of WS-Addressing in a built-in module. This module can be globally or programmatically engaged. The module is globally engaged by adding the following line to the axis2.xml file located under the `AXIS2_HOME` directory.

```
  <module ref="addressing"/>
```

To programmatically engage the WS-Addressing module, the following line client API call is made.

```
  axis2_svc_client_engage_module(svc_client, env, AXIS2_MODULE_ADDRESSING);
```

In our implementation, we have chosen to globally engage WS-Addressing. With the module engaged, no special options need to be set server side. WS-Addressing is automatically employed if incoming messages have WS-Addressing headers. There is one mandatory client side requirement for using WS-Addressing. The WS-Addressing action for the operation that is to be invoked must be set accordingly. This action then appears in the SOAP header of an outgoing message. The action is set using the following client side API call.

```
 axis2_options_set_action(options,env,<action_name>);
```

2.4.5 Compiling an Axis2 Client

Once the Axis2 client has been created, it must be compiled before it can be used. Here we list the basic required dependencies to compile an Axis2 client. These dependencies arise from utilizing the Axis2 client APIs to setup the client as described in the sections above.

```
gcc -o <client_output_name>
  -I$AXIS2C_HOME/include/axis2-1.4.0/
  -L$AXIS2C_HOME/lib
  -laxutil
  -laxis2_axiom
  -laxis2_parser
  -laxis2_engine
  -lpthread
  -laxis2_http_sender
  -laxis2_http_receiver <client_filename>.c
  -ldl -Wl,--rpath -Wl,$AXIS2C_HOME/lib
```

2.5 Implementing the RNS Operations

To be compliant with the RNS specification, the SNARL service implements each of the five RNS operations: add, remove, list, query, and move. To implement each operation, the SNARL service must make the appropriate LFC API calls. In this section we describe in detail the specifics relating to each RNS operation and the LFC API calls that were chosen to carry out these operations. In making these implementation decisions, a dilemma arose regarding the representation of replicas in an RNS namespace. We begin with a discussion of this dilemma.

2.5.1 Naming Replicas

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. At one level, LFCs are registries of logical file names. However, each logical file name corresponds to one or more physical files (aka replicas). It is about these physical files that LFCs contain more detailed information such as creation time, size, and last access time.

In choosing whether to depict replicas in the RNS namespace, we considered the consequences of our choice. If we did not reveal replicas in the RNS namespace, a separate method would be needed to convey this information. One could image using a separate service to deal with the replicas. However, such a service would no longer be under the RNS specification and thus our hope for interoperating would be lost.

Another option would be to use other WS-* standards to handle this situation. The WS-Naming [8] standard describes how resolvers can be used to handle resolution between multiple endpoints such as in the case of replicas. This option also requires the creation of a separate resolution service to handle the replicas. However, the service falls under the WS-Naming specification that extends WS-Addressing and thus being standards compliant would work with the RNS specification and not hurt interoperability. However, even when using a resolver service, information about only one replica is presented at one time. The additional concern of how to choose one replica arises. We decided that for our first SNARL service implementation we would depict replicas as any other named entity in the namespace. Thinking about these issues brought out the simplicity of the RNS interface.

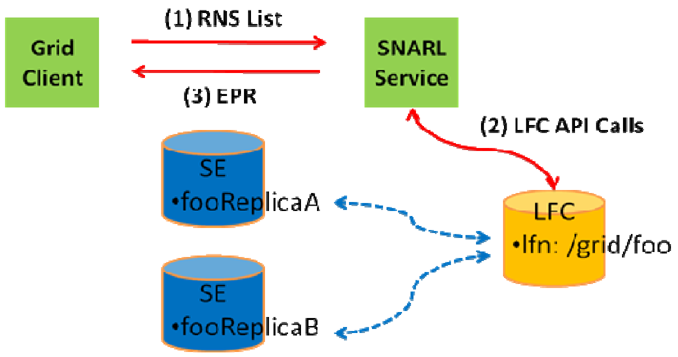

Figure 2. A diagram of the distinction between logical file names in an LFC and physical file locations on different storage elements. This distinction brought up the question of how replicas should be represented in an RNS namespace.

2.5.2 RNS Add Operation

The add operation defines a new name-to-resource mapping that is to be entered into the namespace. The provided name is to be mapped to an optionally specified endpoint reference (if not specified, one is created). In relation to an LFC, an RNS add operation translates to the creation of a new logical file entry in the LFC. Additionally, since replicas appear as named entities in the RNS namespace, the RNS add request may be requesting the addition of a new replica mapping into the LFC. We distinguish between these two cases as they require different LFC API calls.

Figure 3 below provides an example of an add request and response message using SOAP 1.1 as taken from the RNS specification document. This example illustrates the exchange for a request to register an entry named “foo.txt”.

```
<s11:Envelope
      xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://www.w3.org/2005/03/addressing"
      xmlns:rns="http://schemas.ggf.org/rns/2006/05/rns"
    <s11:Header>
      <wsa:Action>
        http://schemas.ggf.org/rns/2006/03/rns/add
      </wsa:Action>
      <wsa:To s11:mustUnderstand="1">
        http://abc.com/rns/A
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:add>
        <rns:entry_name> foo.txt </rns:entry_name>
        <rns:entry_reference>
            <wsa:Address> http://xyz.com/misc/foo.txt </wsa:Address>
        </rns:entry_reference>
      </rns:add>
    </s11:Body>
</s11:Envelope>
```

Figure 3a. An example of an RNS add request message using SOAP 1.1. The request is to register a new entry "foo.txt" into the namespace that maps to an EPR with address "http://xyz.com/misc/foo.txt".

```
<s11:Envelope
      xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://www.w3.org/2005/03/addressing"
      xmlns:rns="http://schemas.ggf.org/rns/2006/05/rns"
    <s11:Header>
      <wsa:Action>
        http://schemas.ggf.org/rns/2006/03/rns/addResponse
      </wsa:Action>
      <wsa:To>
        http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:addResponse>
        <rns:entry reference>
            <wsa:Address> http://abc.com/rns/A/foo.txt </wsa:Address>
        </rns:entry reference>
      </rns:addResponse>
    </s11:Body>
</s11:Envelope>
```

Figure 3b. An example of an RNS add response message using SOAP 1.1. The response indicates that a new mapping has been entered into the namespace for an EPR with address "http://xyz.com/misc/foo.txt".

2.5.3 RNS Remove Operation

The remove operation is used to delete an existing name-to-resource mapping from the namespace. In relation to an LFC, an RNS remove operation corresponds to the deletion of an existing logical file entry from the LFC. Additionally, since replicas appear as named entities in the RNS namespace, the RNS remove request may be requesting the deletion of an existing replica mapping from the LFC. We distinguish between these two cases as they require different LFC API calls.

Figure 4 below provides an example of a remove request and response message using SOAP 1.1 as taken from the RNS specification document. This example illustrates the exchange for a request to remove an entry named "foo.txt".

```
<s11:Envelope
      xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://www.w3.org/2005/03/addressing"
      xmlns:rns="http://schemas.ggf.org/rns/2006/05/rns"
    <s11:Header>
      <wsa:Action>
        http://schemas.ggf.org/rns/2006/03/rns/remove
      </wsa:Action>
      <wsa:To s11:mustUnderstand="1">
        http://abc.com/rns/A
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:remove>
        <rns:entry_name> foo.txt </rns:entry_name>
      </rns:remove>
    </s11:Body>
</s11:Envelope>
```

Figure 4a. An example of an RNS remove request message using SOAP 1.1. The request is to remove an entry named "foo.txt" from the namespace.

```
<s11:Envelope
      xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://www.w3.org/2005/03/addressing"
      xmlns:rns="http://schemas.ggf.org/rns/2006/05/rns"
    <s11:Header>
      <wsa:Action>
        http://schemas.ggf.org/rns/2006/03/rns/removeResponse
      </wsa:Action>
      <wsa:To>
        http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:removeResponse>
        <rns:entry_name> foo.txt </rns:entry_name>
      </rns:removeResponse>
    </s11:Body>
</s11:Envelope>
```

Figure 4b. An example of an RNS remove response message using SOAP 1.1. The response indicates that the entry named "foo.txt" was removed from the namespace.

2.5.4 RNS List Operation

The list operation is used to request a list of subentries associated with a given name in the namespace. In the RNS 1.0 specification, this request can be made with the exact directory name or using a regular expression. In relation to an LFC, an RNS list operation corresponds to a directory listing. Additionally, since replicas appear as named entities in the RNS namespace, the RNS list request may be requesting the listing of the replicas associated with a logical file entry. We distinguish between these two cases as they require different LFC API calls.

It has been noted in experiences with other RNS implementations that accepting regular expressions makes the implementation of this operation very difficult. As was done in the Genesis II RNS implementation, we have chosen to accept the exact name of the RNS directory to be listed. If no name is specified, the root directory listing is returned.

Figure 5 below provides an example of a list request and response message using SOAP 1.1 as taken from the RNS specification document. This example illustrates the exchange for a request to list the current operating directory.

```
<s11:Envelope
      xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://www.w3.org/2005/03/addressing"
      xmlns:rns="http://schemas.ggf.org/rns/2006/05/rns"
    <s11:Header>
      <wsa:Action>
        http://schemas.ggf.org/rns/2006/03/rns/remove
      </wsa:Action>
      <wsa:To s11:mustUnderstand="1">
        http://abc.com/rns/A
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:list>
        <rns:entry_name_regexp> </rns:entry_name_regexp>
      </rns:list>
    </s11:Body>
</s11:Envelope>
```

Figure 5a. An example of an RNS list request message using SOAP 1.1. The request is to list the subentries of the current operating directory.

```
<s11:Envelope
      xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://www.w3.org/2005/03/addressing"
      xmlns:rns="http://schemas.ggf.org/rns/2006/05/rns"
    <s11:Header>
      <wsa:Action>
        http://schemas.ggf.org/rns/2006/03/rns/listResponse
      </wsa:Action>
      <wsa:To>
        http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:listResponse>
        <rns:EntryList>
          <rns:entry>
            <rns:entry_name> foo.txt </rns:entry_name>
            <rns:entry_reference>
              <wsa:Address> http://xyz.com/misc/foo.txt </wsa:Address>
            </rns:entry_reference>
          </rns:entry>
          <rns:entry>
            <rns:entry_name> bar.txt </rns:entry_name>
            <rns:entry reference>
              <wsa:Address> http://123.com/bar.txt </wsa:Address>
            </rns:entry_reference>
          </rns:entry>
        </rns:EntryList>
      </rns:listResponse>
    </s11:Body>
</s11:Envelope>
```

Figure 5b. An example of an RNS list response message using SOAP 1.1. The response lists the names and EPRS of each of the named subentries in the current operating directory.

2.5.5 RNS Move Operation

The move operation provides a means of moving or renaming an existing entry in the namespace. In relation to an LFC, an RNS move operation corresponds to the deletion of an existing logical file entry from the LFC and the creation of a new logical file entry with the specified name. Additionally, since replicas appear as named entities in the RNS namespace, the RNS move request may be requesting the deletion of an existing replica mapping from the LFC and the creation of a new mapping with a different name. We distinguish between these two cases as they require different LFC API calls.

Figure 6 below provides an example of move request and response messages using SOAP 1.1 as taken from the RNS specification document. This example illustrates the exchange for a request to rename an entry named “foo.txt” to “bar.txt”.

2.5.6 RNS Query Operation

The query operation is used to get the properties associated with an existing name-to-resource mapping in the namespace. In relation to an LFC, an RNS query operation corresponds to a stat operation for an existing logical file entry in the LFC. Additionally, since replicas appear as named entities in the RNS namespace, the RNS query request may be querying a replica. In this case the appropriate replica stat LFC API call is made.

Figure 7 below provides an example of query request and response messages using SOAP 1.1 as taken from the RNS specification document. This example illustrates the exchange for a query of an entry named “foo.txt”.

```
<s11:Envelope
      xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://www.w3.org/2005/03/addressing"
      xmlns:rns="http://schemas.ggf.org/rns/2006/05/rns"
    <s11:Header>
      <wsa:Action>
        http://schemas.ggf.org/rns/2006/03/rns/move
      </wsa:Action>
      <wsa:To s11:mustUnderstand="1">
        http://abc.com/rns/A/foo.txt
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:move>
        <rns:entry_name> bar.txt </rns:entry_name>
      </rns:move>
    </s11:Body>
</s11:Envelope>
```

Figure 6a. An example of an RNS move request message using SOAP 1.1. The request is to rename an entry from “foo.txt” to “bar.txt”.

```
<s11:Envelope
      xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://www.w3.org/2005/03/addressing"
      xmlns:rns="http://schemas.ggf.org/rns/2006/05/rns"
    <s11:Header>
      <wsa:Action>
        http://schemas.ggf.org/rns/2006/03/rns/moveResponse
      </wsa:Action>
      <wsa:To>
        http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:moveResponse>
        <rns:entry_reference>
            <wsa:Address> http://abc.com/rns/A/bar.txt </wsa:Address>
        </rns:entry_reference>
      </rns:moveResponse>
    </s11:Body>
</s11:Envelope>
```

Figure 6b. An example of an RNS move response message using SOAP 1.1. The response indicates that an entry has been mapped into the namespace for an EPR with address “http://abc.com/rns/A/bar.txt”.

```
<s11:Envelope
      xmlns:s11="http://www.w3.org/2003/05/soap-envelope"
      xmlns:wsa="http://www.w3.org/2005/03/addressing"
      xmlns:rns="http://schemas.ggf.org/rns/2006/05/rns"
    <s11:Header>
      <wsa:Action>
        http://schemas.ggf.org/rns/2006/03/rns/query
      </wsa:Action>
      <wsa:To s11:mustUnderstand="1">
        http://abc.com/rns/A/foo.txt
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:query />
    </s11:Body>
</s11:Envelope>
```

Figure 7a. An example of an RNS query request message using SOAP 1.1. The request is to query a namespace entry named “foo.txt”.

```
<s11:Envelope
      xmlns:s11=”http://www.w3.org/2003/05/soap-envelope”
      xmlns:wsa=”http://www.w3.org/2005/03/addressing”
      xmlns:rns=”http://schemas.ggf.org/rns/2006/05/rns”
    <s11:Header>
      <wsa:Action>
        http://schemas.ggf.org/rns/2006/03/rns/queryResponse
      </wsa:Action>
      <wsa:To>
        http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
      </wsa:To>
    </s11:Header>

    <s11:Body>
      <rns:queryResponse>
        <rns:EntryProperties>
          <rns:entry_parent>
            <wsa:Address> http://abc.com/rns/A </wsa:Address>
          </rns:entry parent>
          <rns:entry_name> foo.txt </rns:entry_name>
          <rns:entry_reference>
            <wsa:Address> http://xyz.com/misc/foo.txt </wsa:Address>
          </rns:entry_reference>
        </rns:EntryProperties>
      </rns:queryResponse>
    </s11:Body>
</s11:Envelope>
```

Figure 7b. An example of an RNS query response message using SOAP 1.1. The response indicates that queried entry’s name, the entry EPR, and the entry’s parent EPR.

3 Interoperability Testing

Once the development of the SNARL service was completed, we conducted RNS interoperability tests with a Genesis II grid. The Genesis II grid system developed at the University of Virginia contains one of two existing implementations of the RNS specification. The other RNS implementation was developed at the University of Tsukuba. However, the SNARL service is the first RNS implementation to have been done for a pre-existing grid system. The interoperability tests with the Genesis II grid were the big trail of the success of the implementation of the SNARL service. In this section, we describe the testing setup and the interoperability results.

3.1 The Setup

A Genesis II RNS test client was created for the interoperability testing. This Genesis II client was setup to send request messages to the SNARL service for three (add, list, and remove) of the RNS operations. All Genesis II security features were turned off for the interoperability testing. Specifically, the Genesis II security configurations were set to the WARN level that allows a Genesis II client to communicate with resources that cannot be authenticated.

Two separate machines were used for the interoperability testing. Both machines resided inside the CERN internal network. On one machine, an LFC was setup along with an instance of the SNARL service. On another machine, a Genesis II grid was setup. Messages were sent between these two machines for the interoperability testing.

3.2 Interoperability 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 does not comply with the RNS specification. RNS list, add, and remove requests were sent by a Genesis II client to the SNARL service. The service processed each request, made the corresponding LFC API calls, and returned the appropriate response messages to the Genesis II client. In parallel to the interoperability testing, LFC command line calls were made to confirm the correctness of each response and that each operation had indeed occurred as expected. We describe this procedure in more detail for each tested operation.

3.2.1 RNS Add Operation Testing

The RNS add operation creates a new entry in the namespace. Before the start of the RNS add request to the LFC, an `lfc-ls` call was made to check what entries existed in the LFC. This command-line call directly queries the status of the LFC. After the completion of the RNS add operation, an `lfc-ls` call was again made. This call confirmed that the new entry had been created in the LFC via the SNARL service as requested by the Genesis II client request.

3.2.2 RNS List Operation Testing

The RNS list operation requests the listing of subentries associated with a given entry. The list of subentries returned in the RNS list response message was compared against the list of subentries existing in the LFC for the specified directory. The LFC was contacted using the command-line `lfc-ls` call to make this comparison. This call confirmed that the correct entry listing was returned to the Genesis II client via the SNARL service in the RNS response message.

3.2.3 RNS Remove Operation Testing

The RNS remove operation requests the deletion of an entry from the namespace. Before the Genesis II client sent the remove request to the SNARL service, the LFC was directly contacted using the command-line `lfc-ls` call to check what entries existed in the LFC. After the completion of the RNS remove operation, the `lfc-ls` call was once again performed. It was confirmed that the SNARL service had successfully deleted the requested entry from the LFC as per the RNS remove request issued by the Genesis II client.

4 Future Work

The current SNARL service provides a very basic implementation of the RNS specification. There are many enhancements that could be made to make the service more useful and well-rounded. In this section, we mention some of these possible enhancements to the SNARL service.

4.1 Adding Security

To make the SNARL service actually usable in a production-level environment, security mechanisms need to be incorporated into the service. Currently, the service does not facilitate any security measures. This can be changed by utilizing one of the OGF security specifications. Conveniently, the Axis2/C framework provides built in support for the WS-Security standard that could be utilized to include security information with RNS operations.

4.2 Implementing ByteIO

The logical next step after implementing the RNS specification is implementing the ByteIO [9] specification. This OGF specification describes a standard way of handling the transfer of data associated with grid endpoints. The RNS specification reveals named entities in a namespace while the ByteIO specification enables manipulation of any bytes of data associated with theses named entities. Providing a standard specified way to access this data is the next logical step in expanding the interoperability between EGEE LFCs and other standards compliant grids.

4.3 RNS 2.0

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

4.4 SNARL in the Middle

If an implementation of the ByteIO specification and security mechanisms are incorporated into SNARL service, use opportunities for the SNARL service outside the grid interoperability space would be possible. The service could provide basic access capabilities for LFCs not just between grid infrastructures but also programming platforms. Since the LFC is implemented in C, access to LFC APIs for Java developers has not been easily possible. Perhaps the SNARL service could bridge this gap and serve as the interface for such interactions. The performance ramifications of using the SNARL service over other alternatives would have to be investigated.

5 Conclusions

This project was an implementation exercise of the RNS specification for EGEE LFCs. It was the first undertaking to implement this OGF specification for an existing grid system. The resulting SNARL service successfully enabled direct interoperation between an LFC in an EGEE grid and a Genesis II grid using the RNS standard specified operations. We hope our work will provide the RNS OGF working group with useful insights into the difficulties of trying to make a production-level system utilize this OGF specification.

6 Acknowledgement

I would like to acknowledge the individuals who have made the creation of the SNARL service possible. In particular, I want thank my advisors Erwin Laure and Andrew Grimshaw for the project idea, and my daily mentors Akos Frohner and Mark Morgan for their help with the implementation details.

References

1. **"Open Grid Forum,"** *http://www.ogf.org*.
2. M. Pereira, O. Tatebe, L. Luan, and T. Anderson, **"Resource Namespace Service Specification,"** *https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.gfswg/docman.root.working_drafts/doc8272/5,* World Wide Web Consortium, 2006.
3. 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.
4. Gudgin, M., Hadley M., and Rogers T., 9 May 2006, **"Web Services Addressing 1.0 – Core,"** World Wide Web Consortium, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509.
5. M. M. Morgan and A. S. Grimshaw, **"Genesis II – Standards Based Grid Computing,"** *Seventh IEEE International Symposium on Cluster Computing and the Grid,* 2007.
6. **"Apache Axis2/C,"** *http://ws.apache.org/axis2c*.
7. Lawrence, K., Kaler, C., and Flinn, D., 2006, **"WS-Security,"** http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.
8. WS-Naming Working Group at OGF, 2006, **"WS-Naming Specification,"** https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.ogsa-naming-wg/docman.root.current_drafts/doc6861/1.
9. Morgan, M., Chue-Hong, N., and Drescher, M., 2006, **"ByteIO Specification 1.0,"** https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.byteio-wg/docman.root.current_documents/doc13719/1.