Genesis II: Motivation, Architecture, and Experiences using Emerging Web and OGF Standards[1]

Mark Morgan, University of Virginia

7 September 2006

Abstract

In the past year, hype over web services and their use in emerging software applications has prompted the creation of many standards and proto-standards. On top of this, the OGF (Open Grid Forum [3]) has seen a number of standards making their way through various design and edit pipelines. In parallel with the continued growth of these specifications, it is important that implementations develop along-side the documents which validate the efforts of the document writers. Well meaning authors of a specification document might inadvertently loose site of the purposes to which those specifications must eventually answer. Further, in the context of the OGF and its work, no specification exists in isolation. OGF specifications inevitably compose to form a higher order product then the individual components themselves. Together, these specifications will be used to form the grid infrastructure of the future and an evaluation of this emerging work, not only in and of itself, but also taken together in the context of an emerging grid becomes increasingly relevant.

This document describes the pieces of, motivation behind, and experiences composing the Genesis II system at the University of Virginia. Genesis II is a production level compute and data grid, based on existing and emerging web and grid standards, designed not only to provide a framework for research going forward, but also to validate the components from which it is derived. In doing so we hope to provide useful feedback to the OGF, and more specifically, to the working groups represented by this effort.

[1] Partially funded by NSF-SCI-0426972 and the Office of the UVa Vice Provost for Research.

1 Table of Contents

Abstract 1
1 Table of Contents 2
2 Acknowledgements 3
3 Introduction 3
4 Background 4
4.1 WS-Addressing 4
4.2 WS-BaseFaults 4
4.3 WS-Security 4
4.4 JSDL 4
4.5 WS-Naming 5
4.6 RNS 5
4.7 OGSA-ByteIO 6
4.8 OGSA-BES 6
5 Genesis II Architecture and Use 6
5.1 Basic RNS Directories 7
5.2 Basic ByteIO Resources 8
5.3 Exported Directories 9
5.4 Basic BES Containers and BES Activities 9
5.5 BES Multiplexer 11
5.6 Genesis II Container Service 12
6 Experiences and Future Work 12
6.1 Experiences 13
6.1.1 Naming as Core Infrastructure 14
6.1.2 BES Overly Generic 15
6.2 Future Work 16
References 18

2 Acknowledgements

Lest I forget the individuals that have allowed me to write this document, I wish to acknowledge the individuals who have helped transform Genesis II from an "over coffee" discussion a few months ago into the functional, production system available now. In particular, I wish to thank Andrew Grimshaw and Howie Huang for their help designing the system, and John Karpovich, Duane Merrill, Woochul Kang, and Janis Shultz for their continued efforts to design and implement various components of Genesis II.

3 Introduction

In the past year, hype over web services and their use in emerging software applications has prompted the creation of many standards and proto-standards. On top of this, the OGF (Open Grid Forum [3]) has seen a number of standards making their way through various design and edit pipelines. In parallel with the continued growth of these specifications, it is important that implementations develop along-side the documents which validate the efforts of the document writers. Well meaning authors of a specification document might inadvertently loose site of the purposes to which those specifications must eventually answer. Further, in the context of the OGF and its work, no specification exists in isolation. OGF specifications inevitably compose to form a higher order product then the individual components themselves. Together, these specifications will be used to form the grid infrastructure of the future and an evaluation of this emerging work, not only in and of itself, but also taken together in the context of an emerging grid becomes increasingly relevant.

This document describes the pieces of, motivation behind, and experiences composing the Genesis II system at the University of Virginia. Genesis II is a production level compute and data grid, based on existing and emerging web and grid standards, designed not only to provide a framework for research going forward, but also to validate the components from which it is derived. In particular, Genesis II is based heavily on the following specifications and proto-specifications[2]:

- WS-Addressing [4]
- WS-BaseFaults [6]
- WS-Security [7]
- JSDL [8]
- WS-Naming [9]
- RNS (Resource Naming Service) [10]
- OGSA-ByteIO [11]
- OGSA-BES (Basic Execution Service) [12]

[2] Many specifications lead to the design of Genesis II including, but not limited to SOAP 1.2, WSDL 1.1, HTTP 1.1, etc. It would be impossible to do justice to all specifications which are part of this work so I limit discussion herein to those specifications which are directly relevant to the work coming out of the OGF.

This document begins with a brief, high-level description of each of these specifications and proto-specifications. Following the introductory material, we describe the Genesis II system and its architecture. Finally, we conclude with a section summarizing our experiences implementing and using these specifications, both in isolation and in coordination with one another.

4 Background

4.1 WS-Addressing

WS-Addressing is a specification coming out of the W3C [4] which describes, among other things, a schema type called an EndpointReferenceType (EPR) that clients and services use to identify a target web service and the means by which clients transform that information into the message headers for a SOAP message targeting the indicated web service. This schema includes fields for identifying the target address (URI) of the desired service, opaque referencing information which target services may use to further identify session data, and metadata information which can be used by clients as hints describing various aspects of the target web service or web service resource[3].

4.2 WS-BaseFaults

WS-BaseFaults is a standard emerging from OASIS and part of the WSRF [5] framework which extends on the basic concept of SOAP Faults[14]. Specifically, WS-BaseFaults gives a base schema type for fault information "thrown" by web service operations that includes information about the source of the exceptions, details message information, timestamp information, and other relevant pieces of information.

4.3 WS-Security

Security is fundamental to the success and use of web services and distributed systems. It is perhaps the most common denominator between successful grid technologies. Because Grids exist to bring together resources from distributed, mutually distrustful organizations and entities with widely varying security requirements, concerns, and policies, the adoption of any given grid technology is inevitably tied to the usability and functionality of its security layers. WS-Security is a specification that describes how web services communicate securely to protect individual message integrity, confidentiality, and client authentication. In and of itself, WS-Security is not a complete security story, but in Genesis II it serves to facilitate a model which protects the interactions between users and web services.

4.4 JSDL

The act of launching a job or activity on a compute grid roughly breaks down into four distinct phases; describing the job to execute, locating the resource or host on which to execute said job, preparing or deploying the selected application for its execution, and

[3] For an excellent discussion of both the evolution of the WS-Addressing EndpointReferenceType and the uses of its various fields, please refer to Steve Vinoski's paper, "WS-Addressing Metadata" [1]

running and maintaining the indicated job on the selected resource. JSDL addresses the first of these requirements.

JSDL is far too large to describe in total in this document, but at a high-level, it is a schema for describing in XML an application, how to stage the data for that job, and what restrictions apply to its execution. Further, rather than attempt to describe specifications for all possible application types (legacy binary[4], web service instance, grid-aware application, etc.), the JSDL authors instead have left numerous extensibility points (open ended xsd:any elements) throughout the schema which allow for extension profiles that further describe or specify the various application types, data types, etc. In particular, the JSDL document itself includes one such profile for what the authors call POSIX applications (essentially legacy applications).

4.5 WS-Naming

While most OGF specifications described in this document are intended to address specific grid concerns, WS-Naming is somewhat unique in its applicability to non-grid application domains. This specification specifically deals with two issues that are common to almost any distributed system; unique identification or naming of target endpoints, and rebinding of or re-resolution to those endpoints from abstract identifiers to communicable addresses[5].

WS-Naming describes two extensibility profiles on the standard WS-Addressing specification whereby target service endpoints add additional elements to their WS-Addressing EndpointReferenceType's metadata element; namely an endpoint identifier element which serves as a globally unique (both in space and time) abstract name for that resource, and a list of zero or more resolver endpoints. The endpoint identifier element gives clients a way of identifying and comparing addressing endpoints without requiring them to communicate with those endpoints while the resolver list indicates a number of services that clients can use to bind those endpoint identifiers to new WS-Addressing EPRs (for example when a service endpoint migrates from one address to another).

4.6 RNS

As Tannebaum and van Steen describe in their distributed systems book [2], naming in a distributed system often involves a three-tiered approach whereby human-readable names or paths are resolved to abstract names or identifiers, which in turn are mapped into location-dependent addresses. WS-Naming addresses the latter mapping in this hierarchy. RNS addresses the former – that of translating from human-readable names into location independent identifiers.

At its most basic, RNS is a distributed means of mapping human readable names into WS-Addressing EPRs. Because these EPRs can refer to any relevant grid (or web)

[4] By "legacy", I mean to identify application or job types which are existing binaries, unaware of the nature of the grid or its involvement, which a client intends to execute on the grid by staging required data to and from that job as needed.

[5] Andrew Tannenbaum and Martin van Steen given an excellent description of naming in distributed systems in there book "Distributed Systems: Principles and Paradigms" [2].

service, one achieves a hierarchical (or filesystem-like) organization by allowing a named mapping inside an RNS resource to indicate another RNS resource (in much the same way as an entry inside a filesystem's directory may be another filesystem directory).

4.7 OGSA-ByteIO

ByteIO is a service specification consisting of two separate port types (one for random access and one for stream-able access) which serves to facilitate a data story for grid technology. The ByteIO authors specifically designed the specification with the belief that clients and users of grids are best served when presented familiar compute pieces (such as data and process execution) via a familiar interface. As such, ByteIO delivers data (or receives data) via an interface which is heavily based on the traditional POSIX file interface. This not only simplifies the use of ByteIO service endpoints by giving clients an interface familiar to them or their authors, but also further facilitates the development of common data sharing protocols which have traditionally presented such an interface to their own clients (such as NFS, CIFS, FTP, etc.).

Further, similar to named pipes, device drivers, and the /proc filesystem in UNIX-like operating systems, ByteIO is not limited to presenting file information, despite its intentional design as a file-like resource. Implementers may choose to use ByteIO to present a file-like interface to any source or sink of data in the grid.

4.8 OGSA-BES

BES is an emerging specification originally designed to fulfill the most basic core of a larger architecture under development in the OGSA; The Execution Management Specification (EMS) [13]. As described above in the section on JSDL, the compute story can be decomposed into four phases leading from intent to action in a compute scenario. BES addresses the final stage of this story by understanding fully reified JSDL and executing/managing the activity described therein.

In addition, similar to ByteIO, BES describes an interface for translating JSDL into running activities, but does not specify that the BES container need be directly responsible for those activities. An implementation may directly control started activities, but could also instead submit the request for an activity to another, backend BES container (which may in turn delegate responsibility to yet another entity).

5 Genesis II Architecture and Use

The VCGR [15] group at the University of Virginia originally created Genesis II (using Java, Axis, WSS4J, and other related products) to address a number of needs and answer various questions about emerging grid technology. These included

- the need for a production grid system with which to provide compute and data grid capabilities to various partner groups and research projects,
- the desire to have a fully functional grid framework on which further grid research could be performed,

- and the desire to "test drive" the various specifications making their way through various standardization organizations to both vet and better understand those specifications, both in isolation, and together as a whole.

Today, users can interact with Genesis II via both a familiar command-line interface (based largely on common *NIX commands such as ls, cat, cp, etc.), and through a grid aware FTP daemon. As we continue to work on Genesis II and as research leads us down various paths, we anticipate the creation of NFS and CIFS servers, web portals for job submission, and other graphical tools for using, observing, and managing the grid.

Further, because Genesis II is based on grid standards both existing and under development, we hope to extend Genesis II grids by "linking" other, non-Genesis II based implementations, into existing Genesis II grids.

This section describes in detail the various components that make up a Genesis II grid. Because of its importance to Genesis II and user interaction, we then detail the uses of RNS and ByteIO that facilitate bringing grid functionality to the average Genesis II user. It concludes with detailed descriptions of two scenarios that show common user interactions with the Genesis II system (a data interaction, and a compute interaction).

5.1 Basic RNS Directories

All human-software interaction boils down to humans identifying intent and the sources and targets of that intent. The most basic instance of this is the filesystem. From the beginning, operating systems have interpreted directories and paths as ways of identifying the objects of its users' intentions. Whether that interaction was to indicate an executable to launch, a file to read, a location to store an output, or a named pipe into another process, the filesystem has provided the ubiquitous bridge between the human element, and the software element.

Genesis II is designed largely around the belief that as grid technology matures and becomes available to the lay user, the adoption of that technology will be inextricably bound to the ability of the grid system to translate age-old interaction paradigms into modern grid intentions. To this end, the ability for RNS resources to provide a familiar filesystem-like abstraction via which users can indicate and talk about target grid resources is one of the most important concepts in the Genesis II software. Nearly every grid resource available to the user exists as a named path in a grid-wide RNS space (this includes everything from file and directories to execution containers, queues, running applications and even non-grid web sites).

```
$ vcgr ls /
/:
VCGR Homepage
bes-containers
collections
containers
factories
queues
$ vcgr cd bes-containers
$ vcgr pwd
/bes-containers
$ vcgr ls -la
bes-containers:
[BES]           BootstrapBES
[BES]           centurion021
[BES]           centurion022
```

Example 1: Example of common, filesystem-like interactions with Genesis II RNS space.

The most basic instance of RNS in Genesis II is in a simple, core RNS implementation whose sole purpose is to represent and operate on a mapping of human readable names to WS-Addressing EndpointReferenceTypes or EPRs. By composing these mappings together into a hierarchical namespace, a distributed filesystem emerges which, along with human organizational conventions, provides a complete and familiar picture to the end user. Example 1 shows an example session where a user is interacting (via a command line interface) with a running Genesis II grid using familiar "filesystem-like" tools. Despite the fact that the directory structure implied exists in a distributed grid which could span computers and countries, the familiarity of the filesystem view makes it easier for the end user to learn the new environment. Additionally, RNS gives us the ability to name any grid object imaginable (not just files and directories). By giving the command line shown in Example 2, the user identifies via RNS paths the name his running job should take on as well as the BES container that will manage that job.

```
$ vcgr run --name=MyJob \
>     --in=input.txt/http://www.tempuri/inputs/input.txt \
>     --out=output.txt/mailto:mmm2a@cs.virginia.edu \
>     /bes-containers/MyContainer
```

Example 2: An example command line that shows non-file RNS interactions.

5.2 Basic ByteIO Resources

The ByteIO specification, along with RNS, completes the picture of a distributed filesystem for the user. In Genesis II, a basic ByteIO service implements the specification by storing its contents on the local disk where the JVM is running. As users perform operations on the ByteIO service, the ByteIO service translates those operations into equivalent local filesystem commands, acting as a proxy between the grid world, and the local machine.

```
$ vcgr cat hello.txt
Hello World!
$
```

Example 3: User "cat-ing" a simple ByteIO resource.

5.3 Exported Directories

While the simple RNS and ByteIO service implementations provide the most basic versions of their respective specifications, the ExportDir capabilities of Genesis II combine these specifications together to offer a more advanced data-sharing technology to its users. With ExportDir, users select a directory structure on their local machine to share into the grid, the ExportDir service "wraps" that directory and each subdirectory inside with an RNS interface, and each contained file with a ByteIO interface. All operations that take place through the grid interface are reflected into the actual filesystem that the user shared out, and all changes made to the local filesystem on that machine are in turn reflected into the grid. ExportDir is a perfect example of using simple specifications to build a product whose value is greater than the sum of its parts.

5.4 Basic BES Containers and BES Activities

The Basic BES container is at the heart of the Genesis II compute technology. This container is a simple implementation of the BES specification that creates legacy activities by launching a new process on the containing host machine.

When a Basic BES service implementation receives a request from a client to create a new activity, it parses the JSDL to verify that all requirements indicated by that are acceptable. It then creates a new activity instance which will take on the responsibility of staging the data in (currently via http, ftp, or rns), executing or launching the legacy job, and finally staging the data out (via ftp, rns, or mailto).

The BES container in Genesis II makes heavy use of the RNS interface to realize the belief that common, familiar interfaces make for a more enjoyable and more productive user experience. This shows up in two separate facets of the BES interaction; data staging[6], and activity identification.

In Example 4**Error! Reference source not found.**, we include a snippit from an example JSDL file that indicates a user staging data to and from an RNS path (one absolute, one relative). Much as would be the case with command line arguments in a conventional operating system like Windows or Linux, the user is indicating that he wishes to load and/or store files at specific paths (which are assumed to refer to ByteIO capable resources) during the execution of his activity. The BES container is capable of working

[6] Data staging to and from RNS is achieved by creating a new URI type whose protocol is the string "rns". This protocol type has not been registered yet with the proper URI registries and as such is informal at this time.

with both relative and absolute RNS path names (realizing that an absolute RNS path name is only a relative path to a given root) by retrieving the "calling-context" of the user who submitted the job. This calling context is passed in SOAP headers with every call and includes information about what is the EPR of the root of the user's RNS space, what is his current location in that space, and what security information he wishes to pass along with the call. Using this context information, and the path given in the JSDL, the BES activity can stage data to and from RNS space just as it would copy the data to and from other URIs.

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition xmlns="http://www.example.org/"
      xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

      ...

      <jsdl:DataStaging>
            <jsdl:FileName>file1.dat</jsdl:FileName>
            <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
            <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
            <jsdl:Source>
                  <jsdl:URI>rns:/home/mmm2a/file1.dat</jsdl:URI>
            </jsdl:Source>
      </jsdl:DataStaging>

      <jsdl:DataStaging>
            <jsdl:FileName>output.dat</jsdl:FileName>
            <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
            <jsdl:DeleteOnTermination>true</jsdl:DeleteOnTermination>
            <jsdl:Target>
                  <jsdl:URI>rns:output.dat</jsdl:URI>
            </jsdl:Target>

      </jsdl:DataStaging>

      ...

</jsdl:JobDefinition>
```

Example 4: Example JSDL Snippit Showing Absolute and Relative RNS Data Staging

Additionally, the basic BES service also itself implements the RNS port type. In this way, it acts as a pseudo-directory (similar to directories in the /proc filesystem) whose entries are determined not from contents added by outside clients, but rather by internal state – in this case, the activities currently running in the container. As the BES container manages various activities, it associates a name (usually available from the JSDL) with each activity's EPR and offers this mapping to external clients via the RNS interface. Using this interface, users can browse into the BES container as if it were another directory and see each running activity, referring to them later by RNS path in other command line tools or grid applications (see Figure 1 for an example).

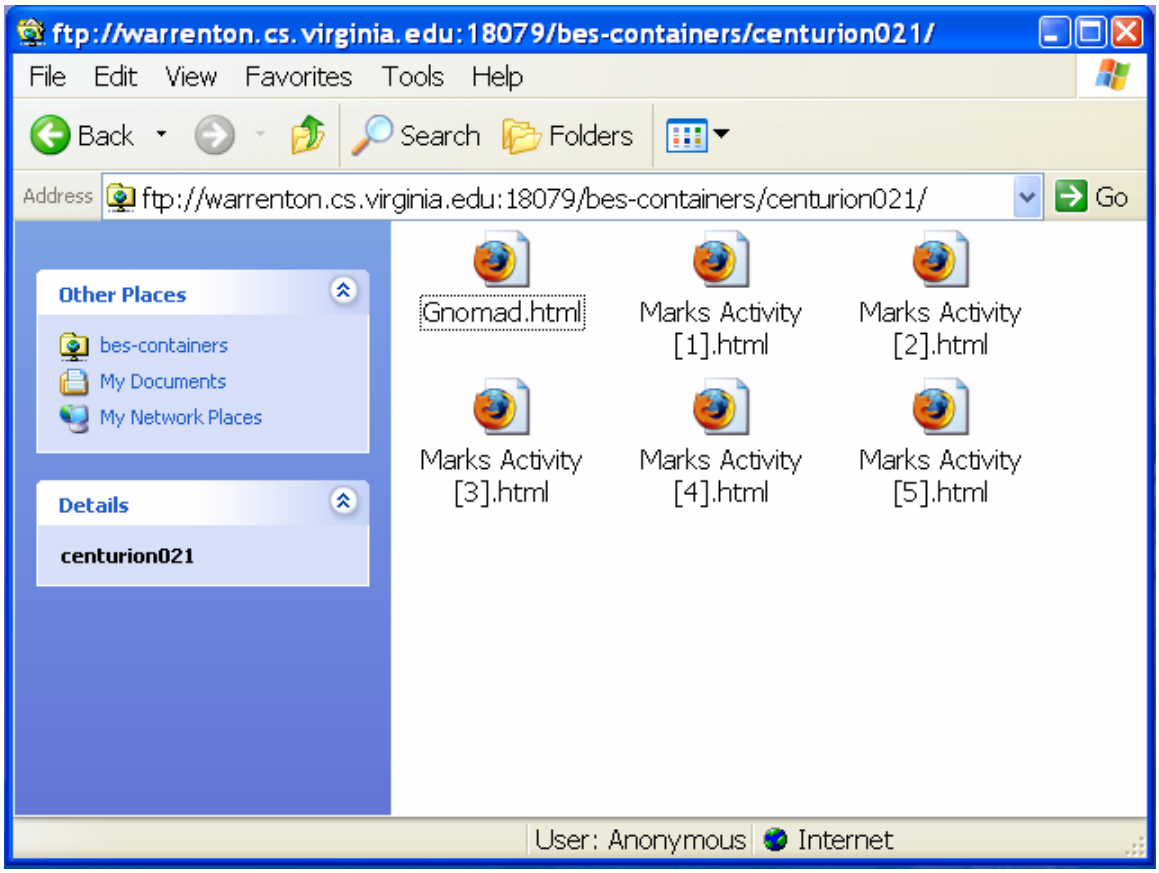

Figure 1: Windows FTP client showing activities in a BES container by browsing into that container through the Genesis II FTP daemon.

5.5 BES Multiplexer

The basic BES container assumes that when it receives a request to create an activity, the intention is for that activity to run in the hosting environment where the BES itself exists. However, this is merely an implementation detail, and not required or even indicated by the BES specification. The BES multiplexer service is a different Genesis II implementation of the BES specification, but in this case, instead of launching activities on a local resource, the multiplexer passes activity requests on to other BES containers (which may themselves be multiplexers, basic BES resources, or some other implementer of the BES protocols). In this way, administrators can form hierarchies of BES containers, thus distributing load and acting as a simple scheduler.

As with the basic BES container, the BES multiplexer also implements the RNS interfaces. However, instead of acting as a mapping of names to activities running in that BES container, instead the multiplexer contains lists of other BES containers. In fact, it is through this interface that administrator manage the BES multiplexer. By linking an existing BES resource into the multiplexer's RNS map, the administrator is indicating that he wishes the multiplexer to consider that new BES container when making scheduling decisions. If the administrator removes a BES resource from a multiplexer's RNS space, then he is effectively reconfiguring the multiplexer to ignore that resource.

5.6 Genesis II Container Service

The final Genesis II service that we will talk about in this paper is the Genesis II container service. This service is not based off an existing or emerging web or grid standard, but rather is a complete homegrown service that Genesis II uses to manage the web-hosting environment (in our case, Jetty, Axis, and a Java JVM). In particular, the container service exports an operation that administrators can use to remotely shutdown a hosting environment and all of its contained resources.

However, while the Genesis II container service is not based off standard specification, it does make heavy use of RNS for facilitating user interactions. In particular, the container service implements a pseudo-RNS space that serves to name services implemented by that container. In other words, if a client lists the contents of the container's Services directory, that user will receive as output a listing of all services currently running in that container's hosting environment (see Example 5). Many tools which need to create new resources for existing services (for example, creating a new exported directory) can use RNS paths indicating these services to indicate which service will host the new exported directory.

```
$ ./vcgr ls /containers/BootstrapContainer/Services
Services:
CounterPortTypePort
BrokerPortType
AdminService
CounterFactoryTypePort
RNSPortType
ExportedRootPortType
ExportedFilePortType
BESPortType
Version
BESActivityPortType
RandomByteIOPortType
ExportedDirPortType
SimpleBESQueuePortType
VCGRContainerPortType

$
```

Example 5: Example of a client listing the contents of the Genesis II Container's services directory.

6 Experiences and Future Work

In the final section of this paper, we discuss some of the experiences and lessons learned from implementing various web service and grid service standards and proto-standards and then we offer some suggestions for possible changes or future directions for those

specifications. Finally, we conclude with a brief description of some possible future directions that Genesis II may take and some ideas for further improving usability.

6.1 Experiences

Technology has come a long way since the days of rolling your own communication protocol and serialization protocols. Instead, today software developers have a wide array of pre-existing tools and environments that perform large portions of this low-level work for them. This is certainly true in the case of Web Services. Web Service developers already have a communication protocol laid out before them – namely SOAP over HTTP(S)[7]. Once you have chosen to use SOAP for your communication, XML is the only realistic choice for serialization and fortunately, the web service hosting environment that your average developer uses probably already handles the serialization for you.

That said, improvement opportunities exist for modern specifications like WSDL, SOAP, and Web Services in general. WSDL is unnecessarily complex and as a result, no two tools parse WSDL the same[8]. WSDL generated by one platform is not guaranteed to be understood by a different platform. Considering that the whole point is interoperability, this seems unfortunate. Likewise, SOAP (and particularly XML serialization), is incredibly inefficient for a communication protocol. Especially in the realm of data grids, specification designers and implementers find themselves forced to implement out-of-band communication protocols for transmitting large quantities of data to make up for the inadequate performance of the existing communication medium. Finally, while viewing a web service in the guise of a pure, service-oriented architecture has some nice properties, from an implementation point of view, it complicates life greatly. The resultant code, which now has to separate data from code, is often large and unwieldy and less robust then it could have been.

In the OGF, the difficulties arise not only due to the relative immaturity of the specifications (which are for the most part still under development), but also from conflicting goals within the organization between its constituents and sometimes even within individual working groups. Further, there is an unfortunate tendency to achieve (or strive for interoperability) by specification dilution[9] rather than through actual compromise. Also, in the attempt to achieve interoperability, there is a tendency to over-generisize interfaces. This makes it easy to interoperate with services at a purely communicational level, but much more difficult at an actual understanding or comprehension level (for example, operations which take an arbitrary XML element describing an arbitrary set of operations and parameters is very easy to communicate with, but nearly impossible to interact with). Finally, vast numbers of loosely coordinated working groups leads to multiple, incompatible solutions for similar

[7] Other options exist for web services, but SOAP over HTTP(S) is so ubiquitous as to be the de-facto standard in web service communication.
[8] In fact, this author has yet to use a tool which in and of itself parses valid WSDL 100% correctly.
[9] By dilution here, I mean that specifications often loose operations and functionality over time in order to resolve disagreements on content.

problems whereas a single, universally adopted specification could have solved the problem for all interested parties. For the remainder of the experiences section of this document, we will focus on two examples that were particularly relevant to the development of GenesisII; WS-Naming and RNS adoption, and the specification of the BES port types.

6.1.1 Naming as Core Infrastructure

Naming comes in three layers in Genesis II (and as described by Tannebaum [2]); human-readable, abstract or location independent, and address or location dependent. The last of these, the address of a resource, is generally not under dispute in the OGF – WS-Addressing EPRs solve the problem of addressing web service resources elegantly and efficiently. The first two components of naming however are often the source of much confusion and much debate. The RNS specification allows for the naming of any grid or web resource, but frequently seems relegated to the role of simple filesystem provider. The second, abstract or location independent names, are often ignored due to the misconception that they are an unnecessary addition to the already sufficient WS-Addressing EndpointReferenceType.

As Genesis II shows, human-readable naming is a vastly important concept in distributed systems and in grids. At the highest level, they provide a means for identifying abstract entities which otherwise would be difficult to talk about during human-software interactions. Things like identifying which container a user wants to run on, which application to launch, which activity to kill, etc., cannot be relegated to low level implementation detail but need to be considered as a fundamental architectural component of all grids as these specifications develop. Human-readable naming is the key to grid adoptability in the future and needs to rest on equal footing with such concepts as serialization, and communication protocols.

The mapping from location independent or abstract names to location-dependent addresses is an easy one to over look. After all, if you have an address for something, you have identified which something you wish to talk to right? Unfortunately, this is not the end of the story when it comes to naming. The location-independent name to address binding is the facilitator for a number of classic Distributed Systems transparencies, not the least of which is migration transparency. If you bind the concept of where something is with the identify of that something, you prohibit that something from every migrating away from its location[10]. Further, identity itself is of prime importance simply as a coding tool. For caching reasons, and cycle-checking reasons, software will often need to ask the question, "Have I talked with this entity before or not?" This phenomenon occurs repeatedly in Genesis II. Indeed, throughout the code, algorithm after algorithm checks to make sure that the EPR to which it is target represents a WS-Name with a valid Endpoint Identifier field. Sometimes failure to do so results in decreased performance and usability by the end user, and sometimes concessions cannot be made and the EPR

[10] It is important to note that migration sometimes happens without a component ever leaving its hosting environment. Consider the case where a web service container such as Jetty or Tomcat runs on a laptop that frequently moves about physically with its owner. Such a laptop will almost certainly acquire new IP addresses as it moves from one network to another.

must be rejected outright. As a result, interoperability becomes an issue when one starts attempting to link together grids that do not offer a uniform, well-specified way of identifying unique resources.

Genesis II has served as an excellent vehicle for showing (at least internally to the VCGR group at UVa) that WS-Naming and RNS are key elements for the future of grids. In particular, as specifications mature within the OGF, an agreement within the community to specify WS-Names as the most appropriate form of EPR would do a lot to help the prospect of grid implementation interoperability.

6.1.2 BES Overly Generic

For this final section on experiences implementing Genesis II, we focus on one specification in isolation; that of BES. The specification represents the work of a number of devoted individuals with a variety of important experiences in the distributed computing and grid computing fields. Representatives from UVa, IBM, Microsoft, Fujitsu, HP, Platform, Argonne, and others have all contributed to producing a specification that is applicable to a wide host of compute grid applications. Unfortunately, in doing so the resultant specification is just generic enough to be difficult to implement and use. In particular, two general problems arise when one tries to implement the BES port types; lack of knowledge about activity location, and lack of specific, detailed information about activity state and operation results.

In the end, most of the difficulties with implementing and using BES result from the strict adherence to a design decision whereby all specified interactions with BES activities must occur through the container itself. While this has some nice properties like easily allowing for bulk operations and giving a second level of defense against non-responsive jobs, the results of this architecture make more mundane and common uses of the technology problematic. Difficulties in defining exceptions (or BaseFaults) for operations which take vectors of activities on which to operate has lead to a specification where these operations instead return vectors of results which little information beyond simple true or false values indicating success or failure. As an example, in Genesis II, while its very easy to determine whether or not an activity completed successfully on a BES container, it is impossible according to BES specification to determine, once an activity has failed, what the failure mode was. This problem can and will be solved in Genesis II with additional port types to the BES container or activity, but such a solution nearly guarantees reduced functionality when interacting with non-Genesis II implementations.

Finally, difficulties arise for software and users trying to control activities started through the BES multiplexer. With the simple BES container, the control point is obvious – it is simply the container to which the job was submitted. Users can asynchronously return to that BES container in the future and, using RNS, identify and check on the status of their running activities. Unfortunately, when the users submits the same activity to the BES multiplexer, he is no longer aware of where the activity finally ended up executing. He has no way of knowing from this point forward how to identify his job without checking the contents of every available BES container delegated to by the multiplexer. This lack

of information leads to serious problems which once again can only be solved by taking on port types not considered part of the standard, thus limiting interoperability with other grids. In fact, the BES specification discusses such a port type, the BES Activity Port Type, but leaves this as an optional, suggested port type rather than a required one. The reasons for this are clear – it is easier to achieve consensus on material that is not mandatory – but to paraphrase a sentiment often overheard at OGF meetings, a specification is the sum of its required elements. Components not required by a specification never interoperate with other implementations. Genesis II shows that this BES Activity Port type is an essential part of a functional, usable compute grid.

6.2 Future Work

Genesis II has proven both a successful proof-of-concept work with regards to specifications making their way through the OGF process, and a successful implementation of basic grid functionality. By adopting a design model where user-interaction becomes a driving force behind system architecture, and with liberal use familiar interaction abstractions, Genesis II will provide an easy-to-learn and easy-to-use platform for the lay user wishing to reap the benefits of grid technology in a virtually seamless manner.

Moving forward, many exciting additions are planned for the Genesis II system. RNS has proven to be an ideal means of conveying grouping information to end users, but many resource types do not exhibit this mapping paradigm. ByteIO could easily become relevant here just as file abstractions in the /proc filesystem have. One could imagine a future where a new ByteIO file is created and filled in with a JSDL document inside a BES resource, thus indicating to the BES service that a new activity was to be created. Such an addition would make it possible for users to start compute activities simply by dragging and dropping JSDL files from their desktop into the grid (or from other locations in the grid). A faux-ByteIO interface on BES activities would allow for users to check on the status of their jobs simply by "cat-ing", or double-clicking those ByteIO files.

Other potential additions include more robust application or binary management and identification (imagine a resource, linked into RNS with a human-readable name, which represents the data for deploying and configuring a binary), or better scheduling algorithsm. Already, work in underway to include a BES broker service that schedules BES activities using modern market and economic principles of use and resource auctioning. Further, a queue-like grid object (which held batches of jobs until resources to execute them became available) which supports the BES interface has already been discussed.

In addition, the VCGR group is looking into ways of doing cutting edge research with QoS policies for both normal, day-to-day use of grid components, as well as QoS policies for handling extreme situations for emergency infrastructure survival. In particular, we plan to give users the ability to specify policies like:

- "This ByteIO resource must have high availability,"
- "I need a BES container which will complete my job before this deadline,"

- “I want a BES container which gives me the lowest cost for usage of its compute cycles,”
- etc.

As the specifications in the OGF continue to develop and mature, Genesis II will continue to track these documents in the hopes of providing a robust and production quality environment that brings grid technology to the lay grid user while attempting to facilitate interoperability with future grid products.

References

[1] Vinoski, Steve. "WS-Addressing Metadata," www.iona.com/hyplan/vinoski/pdfs/IEEE-WS-Addressing_Metadata.pdf. IEEE Internet Computing. May-June 2005. p. 90-93.

[2] Tannenbaum, A. and van Steen, M., 2002, "Distributed Systems: Principles and Paradigms," Prentice Hall, 2002. p. 184-210.

[3] Open Grid Forum. http://www.ogf.org.

[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] Snelling, D., Robinson, I., and Banks, T., 2006, "WSRF," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[6] Liu, L., and Meder, S., 1 April 2006, "Web Services Base Faults 1.2 (WS-BaseFaults)," http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf.

[7] Lawrence, K., Kaler, C., and Flinn, D., 2006, "WS-Security," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

[8] Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., McGough, S., Pulsipher, D., and Savva, A., 7 November 2005, "Job Submission Description Language (JSDL) Specification," https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.jsdl-wg/docman.root.published_docs.jsdl_1_0/doc12582/28.

[9] 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.

[10] Pereira, M., Tatebe, O., Luan, L., and Anderson, T., June 2006, "Resource Namespace Service Specification," https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.gfs-wg/docman.root.working_drafts/doc8272/5.

[11] 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.

[12] Grimshaw, A., Newhouse, S., Pulsipher, D., Morgan, M., Theimer, M., and Robinson, I., 2006, "OGSA Basic Execution Service," https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.ogsa-bes-wg/docman.root.current_drafts/doc13740/1.

[13] Grimshaw, A., Smith, C., and Subramaniam, R., 2005, "Executoin Management Services – OGSA," https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.ogsa-wg/docman.root.design_teams.execution_management_services/doc12774/1.

[14] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D., 8 May 2000, "Simple Object Access Protocol (SOAP) 1.1," http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[15] "Virginia Center for Grid Research (VCGR)," http://vcgr.cs.virginia.edu.