An Evolutionary Approach to Realizing the Grid Vision

Marvin Theimer, Savas Parastatidis, Tony Hey
Microsoft Corporation

Marty Humphrey
University of Virginia

Geoffrey Fox
Indiana University

Abstract

The vision of a network of seamlessly integrated distributed services that provide access to computation, data, and other resources is known simply as the 'Grid'. This paper proposes an evolutionary process to standardizing services and protocols necessary to support applications on the Grid. An architecture with a minimal set of services to support interoperable High-Performance Computing in a Grid environment is presented as a first step towards realizing the Grid platform in its entirety.

1. Introduction

The Open Grid Services Architecture (OGSA) [OGSA] provides the vision and abstract design of a flexible and robust Grid middleware infrastructure. The OGSA working group in the Global Grid Forum (GGF) [GGF] draws requirements from a set of use cases and abstractly describes the potential functional aspects of Grid systems. The blueprints of the architecture are based on the 'service' abstraction and governed by the principles of service composability and interoperability.

The OGSA working group has a mandate to define an overarching intellectual context in which an interrelated, focused set of design efforts can take place. This note proposes that such design efforts should try to define a Grid architecture in incremental steps by concentrating on pragmatic, small and application domain-specific functional subsets of the overall vision. We observe that successful standardization efforts almost always address such small, well-defined, modular subject areas while taking into account broader goals and context. In general, services and protocols that are widely viewed in the community as having open research issues should be avoided in standardization efforts until a time when they attain wide industry acceptance, mature and interoperable implementations, and design stability.

We believe that the key to building a widely-accepted Grid platform is to start with minimalist designs to which small, composable, extensible services and their associated interaction protocols are added in an incremental fashion. Simultaneous design efforts should be encouraged to publish their proposals early and often so that they can be examined and debated by the wider Grid community in the overall context defined by the OGSA working group. The result should hopefully be that common services and interaction protocols will be identified early enough to avoid most of the dangers of reinvention. Furthermore, Grid application domains can selectively implement only those services and interaction protocols that are relevant to them. Figure 1 is a graphical depiction of the approach we are proposing in this paper.

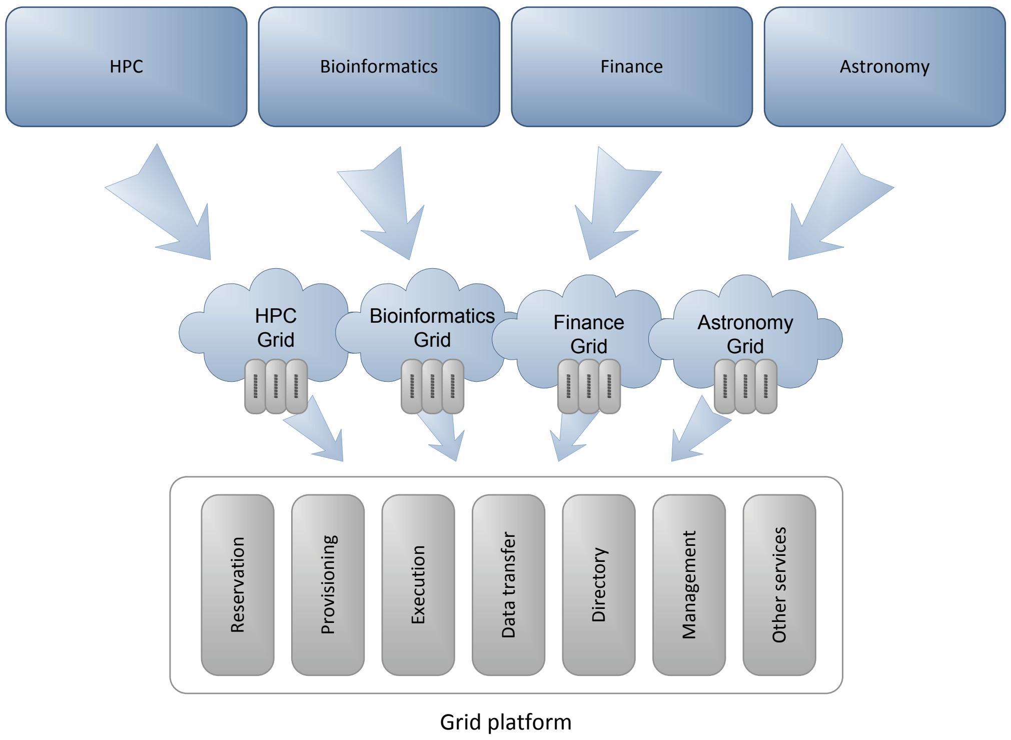

Figure 1: An evolutionary approach to Grid services

As a first step of the proposed design approach this paper describes an architecture that provides a minimal set of services specifically to support applications in the High Performance Computing (HPC) domain. The goal is to enable the production of interoperable HPC Grid environments with adequate core functionality as a starting point, which over time can evolve in complexity and power through the introduction of additional, composable services as identified by experience.

We argue that there is a *core* use-case that drives the requirements for our HPC Grid: the use of computational resources on the Internet to run programs. By standardizing a simple architecture for this use-case alone, we create an interoperable, cross-platform, vendor-neutral Grid foundation sufficient for a significant majority of more specialized HPC use-cases. This paper describes a simple architecture that leverages existing work in the Web Services (WS) community and proposes simple, factored job scheduling and data transfer related services as the foundation for an HPC Grid. The primary purpose of the paper is to advocate a particular approach to designing the Grid architecture. It is not our aim to present our architecture for the HPC Grid in detail; instead, we discuss only those aspects of our design which help us illustrate the principles we are advocating.

In addition to describing the proposed architecture in Section 2, Section 3 of the paper discusses the way in which these services of this architecture could be implemented using widely accepted Web Services specifications. It assumes that standardization work on WS protocols dealing with non-HPC-specific architectural requirements, such as security, policy descriptions, metadata, transactions, reliable messaging, system management, etc., is taking place in the appropriate organizations [W3C, OASIS, IETF, and DMTF] with the broad support of the IT industry. Section 4 concludes the paper.

1.1. Key Objectives

We end this introductory section with a summary of the key objectives of our design approach:

- Emphasize the importance of an evolvable service-oriented Grid architecture. While the eagerness of the Grid community to define all the details of a future Grid platform is understandable, our goal is more modest and is to propose a set of initial services as the basis for a single Grid application domain, namely High Performance Computing. We believe it would be very helpful for the entire Grid community to engage in similar efforts for a variety of Grid application domains. Ongoing work would then incrementally result in more services and protocols, some being specialized versions of the existing ones, while others would add new functionality. Early publication of incremental designs, public debate, and cross-fertilization between design efforts would result in the identification of services and protocols that warrant broader standardization.
- Promote interaction protocols with narrow scope. Interaction protocols between Grid Services are easier to define, implement, combine, and evolve if they are kept simple and address very specific requirements.
- Align the efforts in the Grid domain with existing infrastructure and the wider industry initiatives that are already underway and are beyond the Grid community's control. For example, in the Web services space, the industry is defining all the necessary protocols for secure, reliable, transacted messaging. The industry and open source communities are delivering middleware support and user-education material for these protocols. We argue that Grid specifications should be built on top of only those Web services technologies that are stable, widely-accepted, and have good tooling support.
- Highlight the importance of boundaries in administrative domains and virtual organizations when defining interaction protocols. Standardization work on protocols and services is necessary when it is clear that functionality must be exposed beyond the explicit boundaries of a domain. Services and interaction protocols whose scope can be constrained to stay within the bounds of a domain do not require the definition of a globally applied standard. For example, management of individual system resources is something that arguably should occur only within a bounded (virtual) organization. Consequently a standardized approach to system resource management need not be part of the initial requirements for a simple Grid architecture.

2. A Basic Grid Architecture for HPC

One of the core parts of the Grid vision is the seamless use of computational resources on the Internet to run programs. In a service-oriented architecture, such resources become available through services. To achieve interoperability, services offering identical functionality support the same interaction protocols. The decision on which of the available services on the Internet to use becomes an issue of price, policy, Quality of Service (QoS) requirements, or other metrics, but implementations of the consuming applications need not change.

Beyond “standard” system requirements, such as support for distributed communications, security, and traditional system management, we assert that the following are the ‘core’ requirements for being able to run arbitrary programs in a Grid environment:

- The ability to run programs on designated computational resources, referred to in this paper as ‘job scheduling’. There are three aspects to consider:
    - The computational, data, network, and other resources necessary for the execution of a program must be specified, selected, and reserved for use;
    - One or more programs and their associated libraries and other resources must be (potentially installed and) made accessible for execution on the reserved computational resources; and
    - Programs must actually be executed on the resources that have been reserved and provisioned for use.
- The ability to ship potentially large amounts of data to and from places where executing programs can access and produce the data.
- The ability to dynamically discover and use HPC Grid related services.

The rest of this section first recaps the design principles we are espousing and then describes how these design principles inform the individual pieces of our HPC Grid architecture. Our design principles can be summarized as follows:

- Start with a minimal base and design for incremental evolution.
- Align with and exploit major industry initiatives and existing infrastructure when possible.
- Avoid employing controversial solutions where possible.
- Employ abstraction and boundaries to minimize the scope over which things must be visible or can be affected.

An additional design principle is to “design simple, composable interaction protocols with narrow scope”. This is arguably a corollary of the “design for evolution” principle since it is much easier to evolve systems based on such interaction protocols than on complex, monolithic ones.

2.1. Leveraging Existing Standards

Many things are not specific to Grid computing, but are rather generic aspects of distributed systems. Basic communications between distributed services falls in this category and we see no reason not to leverage the solutions being deployed within the wider IT industry.

Another prime example is security, where the solutions proposed by the IT industry are sufficient for realizing a basic Grid architecture. We also anticipate that broader security approaches (e.g. standards for cross-domain authentication infrastructures) can be leveraged.

There are many other examples, which we will not enumerate in this paper. Instead, we will highlight the more challenging issue of identity, naming, and addressing of artifacts (e.g.

services, resources, devices, processes).[1] Despite many discussions within the broad distributed computing community over the years, this issue has not yet led to a common understanding of the terms and a globally accepted solution.

Interestingly, a widely accepted base infrastructure for naming does exist in the form of the Domain Name System (DNS) [DNS]. This infrastructure has successfully served the Internet for many years. However, various application domains above DNS have differing requirements from each other and have adopted different solutions. As points of reference, the general Web architecture makes use of Uniform Resource Locators (URLs) [URL] to address resources while specific application domains make use of non-transport-specific Uniform Resource Identifiers (URIs) [URI] for their identity/naming requirements (e.g. Amazon Standard Identification Number [ASIN], Life Science Identifier [LSID], etc.)

Based on the principle of avoiding the unnecessary use of controversial or poorly understood concepts and standards, we conclude that a basic HPC Grid architecture should impose as little in the way of identity, naming, and addressing infrastructure as possible. Thus we do not impose a particular identity or naming scheme for resources and services and advocate the use of only an addressing mechanism that guarantees interoperable communications.

2.2. Job Scheduling

In order to enable the execution of programs on remote computational resources, we require a job scheduling interaction protocol. As mentioned earlier, we argue that there are three distinct areas of functionality to consider: resource reservation, provisioning, and execution. The principle of factorization implies that these three aspects of job scheduling should be split out into separate, composable pieces rather than convolved together into a single monolithic job scheduling protocol. Such a split enables a variety of benefits:

- Quality-of-service guarantees and service level agreements can be satisfied through resource reservation spanning multiple executions of the same or different programs (either serially, in parallel, or both).
- Different kinds of provisioning scenarios can be supported independent of the other aspects of job scheduling. For example, in some systems provisioning may be unnecessary because the relevant software is already pre-installed or available through a distributed file system. In other systems provisioning may require explicitly copying binary executable files to the relevant compute resources (and then deleting them upon completion of the program execution). Yet other systems may require the execution of installation and uninstallation scripts to implement provisioning. Separating provisioning from program execution also enables optimizations such as installing/uninstalling a program only once for several executions.
- Interactive and debugging sessions with remotely executing programs can be supported independently of the resource reservation and provisioning steps.

[1] No attempt is made in this document to discuss the similarities/differences between identities, names, and addresses.

Another design consideration to take into account is that there already exist various job schedulers, with varying levels of functionality and specialization, which are in wide use already. The principle of alignment implies that a job scheduling protocol should try to foster interoperability among existing product deployments. However, it is unreasonable to expect scheduling vendors and developers to implement the superset of all features found in the existing job scheduling products.

The principle of minimal, evolutionary design favours an approach in which one should seek to identify those functional denominators—captured as interaction protocols—necessary to support common job scheduling scenarios. The richer functionality found in may of today's job scheduling products can be modelled as extensions to the defined set of core protocols. For example, not all job schedulers support the concept of program suspension. Hence such functionality could be provided by means of an "extension profile" to the basic job execution protocol.

In summary, we advocate that support for job scheduling should be designed as a set of independently evolvable services for reservation, provisioning, and execution. We further advocate that each service should be designed to support a minimalist base functionality that can be selectively enhanced and extended by means of composable protocol extension profiles. As a result, all clients and schedulers will be able to interoperate with each other at a very basic level, while richer functionality of various sorts will be available to those clients and schedulers that understand the associated protocol extension profiles.

2.3. Data Transfer

Jobs submitted for execution on remote computational resources may require access to data not available locally. Hence in HPC applications it is often necessary to transfer such data close to where a job executes. Data transfer requires:

- A means of transferring bytes between distributed services; and
- A means of specifying how those bytes should be interpreted by the services. It is necessary to understand how to interpret the format of the data being transferred (e.g. are these bytes a single file, a directory of files, a row-set, or a relational table) and where the transferred data should be placed within the receiving party's storage name space.

The principle of factorization implies that the issues related to data transfer should be treated separately from the data format and name space ones. This allows a variety of different protocols to be used for transferring data, whose format could adhere to one of several standard formats, and whose final name space destination could be specified by means of one of a variety of different conventions.

While a data transfer protocol for explicitly transferring files and directories of files between independent services is clearly useful and would enable the basic HPC Grid to which we aspire, it is not clear that a global namespace for files—with its attendant issues of global consistency—is necessary for most HPC applications. Hence we argue that such functionality should not be part of a base HPC Grid design. This is not intended to mean that work on a global filesystem with a federated namespace should be discouraged or that the Grid community will not eventually decide on its incorporation as a core service to the Grid platform. However, the principles of incrementally building from a minimal base design and of avoiding solutions that are not yet widely accepted or have open research issues point towards the adoption of a less ambitious data transfer protocol for the HPC Grid

2.4. Directory Services

In a distributed environment with many interchangeable services (i.e. services offering the same functionality by supporting the same interaction protocols), it is useful to be able to discover and dynamically choose one based on QoS, policy, security, inter-organization agreements, and other criteria.

We observe that people already successfully use a variety of means to discover computing resources that may be available to them:

- Their solutions 'know' the name/address of available HPC Grid services;
- Their deployments use static "config" files with a list of endpoints to the services with which they interact: and
- The applications use standard directory service solutions such as UDDI and LDAP.

Since 'simply knowing' the name/address of available services is a viable alternative, the principle of defining a minimal base plus incremental extensions implies that a minimal HPC Grid design should not include a discovery protocol. Since there is clearly significant utility in having a more general means of discovery, we advocate an incremental extension to our base HPC Grid design that encompasses a simple means of service discovery. Employing the principle of alignment with existing infrastructure and industry initiatives, we argue that this extension should target an existing directory service infrastructure rather than inventing a new one.

Unfortunately, whereas there are several existing directory service candidates, such as UDDI and LDAP, there is not a single clear candidate that is unequivocally better than the others. As a result, the HPC Grid architecture does not propose a specific solution. Experience, evolution, and potential convergence would dictate which directory service solution could become an integral part of the Grid platform.

2.5. System Management

System management is a fundamental part of any distributed system and hence of any Grid infrastructure. However, we argue that a global standard for system management is not necessary for the provision of a basic HPC Grid infrastructure.

System management almost always spans the resources residing *inside* an organization. Organizational boundaries are typically set up precisely to prevent resources from being directly accessible by external parties. This implies that each organization could use its own choice of system management infrastructure without affecting the choices and operation of other organizations. The basic HPC Grid design does not depend on a common solution for systems management.

We recognize, however, the potential importance of standard systems management protocols in supporting virtual organization scenarios. As the availability of standards in this space matures and widely accepted solutions emerge they could become part of the larger Grid platform.

3. Web Services

The Web Services Interoperability (WS-I) organization [WS-I] was formed by the industry to create profiles of WS standards that can ensure interoperable implementations. As a result, the published WS-I profiles can be used as a safe basis for any service implementation. Indeed, today the exchange of SOAP [SOAP] messages between different middleware platforms and the use of WSDL [WSDL] to describe message-based interactions is, in most cases, trouble-free. Work in the area of secure and reliable messaging is also being performed based on the relevant standards (i.e. WS-Security [WS-SEC] and related specifications, WS-ReliableMessaging [WS-RM]). Finally, although the work on WS-Addressing [WS-ADDR] has yet to be finalized in W3C, it is widely accepted that it provides a needed functionality for SOAP-based messaging.

One of our design principles is that standards should be based on non-controversial technologies whenever possible. Consequently we advocate that a base HPC Grid architecture use only those WS technologies for which agreement has been reached by the key stakeholders in the WS space. At the moment, such an agreement has been reached only on a subset of the entire WS suite of technologies. This is not to say that new solutions and ideas should be ignored. Instead, the Grid community should continue to experiment with them, work on their standardization and wide acceptance, and demonstrate interoperable tooling. Only then should they be considered for integration in production Grid standards.

Figure 2 presents some of the specifications in the WS technology space that seem relevant to the implementation of our basic HPC Grid architecture. This is by no means a definitive survey of all relevant WS-related work being undertaken. The diagram aims to illustrate that while the industry is significantly investing on standards and interoperable middleware and tooling, there is still a long way to go until the landscape is completely clear of uncertainty and these technologies are stable and standardized.

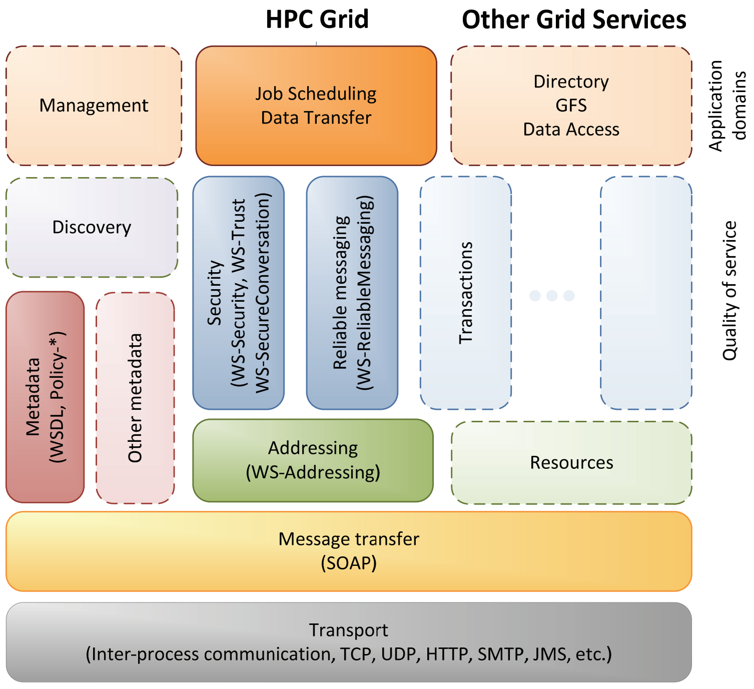
HPC Grid
Other Grid Services
Management
Job Scheduling
Data Transfer
Directory
GFS
Data Access
Application domains
Discovery
Security (WS-Security, WS-Trust WS-SecureConversation)
Reliable messaging (WS-ReliableMessaging)
Transactions
Quality of service
Metadata (WSDL, Policy-*)
Other metadata
Addressing (WS-Addressing)
Resources
Message transfer (SOAP)
Transport (Inter-process communication, TCP, UDP, HTTP, SMTP, JMS, etc.)

Figure 2: A snapshot of the Web Services technology space. Fainted boxes represent domains for which unstable, non-standardized yet, or even competing specifications exist.

There are still areas where specifications are either incomplete or not even submitted to a standards organization. Furthermore, in some domains industry leaders are supporting competing specifications that provide equivalent functionality. As a result, such specifications cannot be considered stable and should be avoided in the implementation of Grid related services until the industry has either converged on a single solution or until natural selection by market forces has declared a clear favorite.

4. Conclusions

While it is tempting to try to define the entire Grid architecture as a whole, there are still open issues to resolve and we believe it is thus premature to try to define a single, monolithic architecture. Instead, our approach advocates defining small, functional subsets of the Grid vision and employing an incremental, evolutionary approach to combining these subsets to achieve an eventual overall Grid architecture.

A key design principle of our approach is reliance on composable, service-oriented, and narrow-scoped interactions protocols. These allow additional services and functionality to be selectively introduced over time (Figure 3). We believe that this evolutionary, composable approach to defining Grid standards will enable rapid and focused progress, encourage early adoption of well-defined, useful Grid subsets, and provide important experience and feedback for an eventually defined overall Grid architecture.

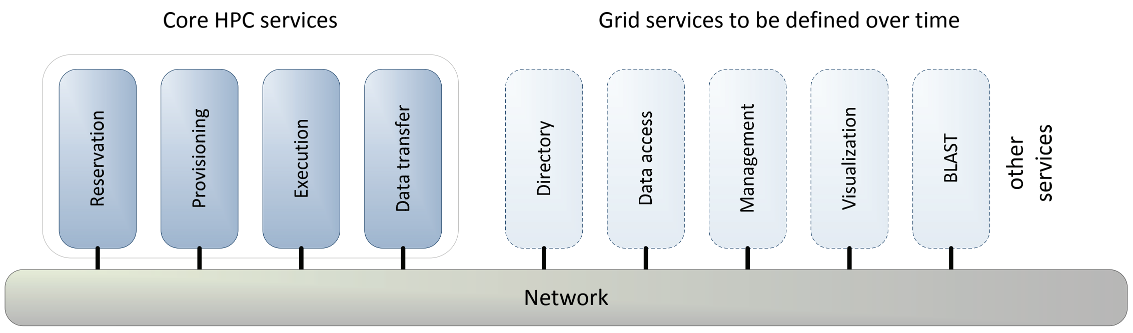

Figure 3: The evolvable Grid architecture as a growing set of services

The evolutionary approach to the overall Grid architecture can also be applied within particular application domains. Those services and interaction protocols which are core to providing the basic functionality for the most common use cases should be defined first. Then, more advanced functionality can be considered, which should be built on the existing core functionality without replacing it and be introduced into the architecture following the principles of composability. Applications should be allowed to integrate only the necessary core functionality and make use of advanced offerings only if desired and appropriate.

Key to the success of this evolutionary approach to defining Grid standards is the use of a stable, widely accepted, and well-supported infrastructure. We make—what we believe is—a non-controversial proposal to build an initial basic Grid architecture on top of only that subset of WS specifications which benefit from wide industry acceptance, quality tooling, and in a standards organization.

To illustrate our proposed design approach, we have introduced a basic HPC Grid architecture that consists of a set of core services to enable interoperable High-Performance Computing on the Grid. The proposed HPC Grid architecture leverages existing work in the Web services community, as well as existing infrastructure for things like security and directory services. It proposes just a few interaction protocols related to job scheduling and data transfer. The base versions of each protocol are intended to be small and simple enough so that universal acceptance and implementation should hopefully be non-controversial. Finally, we observe that the thorny issue of systems management need not be addressed in a basic HPC Grid design since it should arguably be contained within the boundaries of organizations.

6. References

[ASIN]
Amazon Standard Identification Number.
http://www.amazon.co.uk/exec/obidos/tg/browse/-/898182/203-5509542-1342339

[DMTF]
Distributed Management Task Force, Inc. http://www.dmtf.org

[DNS]
P. Mockapetris, "Domain Names - Implementation and Specification" RFC 1035, November 1987.

[FTP]
J. Postel, J. Reynolds, "File Transfer Protocol", RFC 959, October 1985.

[GGF]
Global Grid Forum, http://www.ggf.org

[IETF]
Internet Engineering task Force, http://www.ietf.org

[LSID]
OMG Life Sciences Identifiers Specification dtc/04-05-01. Available at:
http://www.omg.org/docs/dtc/04-05-01.pdf

[OASIS]
Organization for the Advancement of Structured Information Standards,
http://www.oasis-open.org

[OGSA]
I. Foster, H. Kishimoto, A. Savva, eds. " The Open Grid Services Architecture, Version 1.0", Global Grid Forum Document GFD-I.030, 29 January 2005. Available at:
http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf

[OGSA-BES]
OGSA Basic Execution Services Working Group,
https://forge.gridforum.org/projects/ogsa-bes-wg/

[SOAP]
M. Gudgin, et al, "SOAP Version 1.2 Part 1: Messaging Framework", W3C Recommendation 24 June 2003.

[UDDI]
Universal Description, Discovery, and Integration protocol., http://www.uddi.org

[URI]
T. Berners-Lee, et al, "Uniform Resource Identifiers (URI): Generic Syntax," RFC 2396, August 1998.

[W3C]
World Wide Web Consortium, http://www.w3.org

[WS-ADDR]

D. Box, et al, "Web Services Addressing (WS-Addressing)", W3C Member Submission 10 August 2004"

[WS-DM]

OASIS Web Services Distributed Management (WSDM) TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm

[WS-I]

Web Services Interoperability Organization, http://www.ws-i.org

[WS-MAN]

A. Arora et al, "Web Services for Management (WS-Management)", June 2005. Available at: http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-management.pdf

[WS-RM]

R. Bilorusets, et al, "Web Services reliable Messaging Protocol (WS-ReliableMessaging), February 2005. Available at: http://msdn.microsoft.com/library/en-us/dnglobspec/html/WS-ReliableMessaging.pdf

[WS-SEC]

OASIS Web Services Security (WSS) TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

[WSDL]

E. Christensen, et al, "Web Services Description Language (WSDL) 1.1", W3C Note 15 march 2001.