

# An Open Grid Services Architecture Primer

**Andrew Grimshaw, Mark Morgan, and Duane Merrill, *University of Virginia***  
**Hiro Kishimoto, Andreas Savva, and David Snelling, *Fujitsu***  
**Chris Smith, *Platform Computing***  
**Dave Berry, *Edinburgh National e-Science Center***

To expand the use of distributed computer infrastructures as well as facilitate grid interoperability, OGSA has developed standards and specifications that address a range of scenarios, including high-throughput computing, federated data management, and service mobility.

**G**rid computing<sup>1-3</sup> involves the virtualization, integration, and management of services and resources in a distributed, heterogeneous environment that includes collections of users and resources across traditional administrative and organizational domains. To date, developers have built grids using either ad hoc open source software components and protocols or proprietary technologies. While various open source and commercial solutions are successful in their niche areas, interoperation is limited. The result is islands of grid computing that do not interact, preventing grids from reaching their full potential.

Grid scenarios present several significant challenges to end users, application developers, and IT managers. These challenges revolve around issues such as security (authentication, authorization, trust, and data integrity), quality of service (meeting service-level agreements, availability, and so forth), data management, scheduling, and resource management. If neither the application developer nor the grid middleware addresses these issues, the result can be missed deadlines, cost overruns, and less-than-robust software.

The Open Grid Services Architecture addresses these complex challenges by defining a set of standard inter-

faces, service interaction protocols, and profiles on existing standards that provide the foundation on which to build robust grid applications and grid management systems. Thus, OGSA defines the services, their interactions, and the design philosophy.

OGSA is an open service-oriented architecture based on Web services (WS).<sup>4</sup> OGSA is an open architecture in two ways: First, the development of specifications and profiles occurs in a completely open process where no one group or company can define the outcomes, port types, schemata, and interaction protocols. Instead, specifications and profiles are the result of consensus and are nonproprietary. Second, different implementations are available, some of which are open source reference implementations.

OGSA services and specifications have three potential applications: high-throughput computing, providing access to federated data, and facilitating service mobility.

## DEEPER DIVE

OGSA services fall into six broad areas, defined in terms of capabilities frequently required in grid scenarios. While interdependencies can exist between services, not all services need to be used at any given time—different use cases call for different subsets of services.

The six areas are

- *Infrastructure services.* These services include common functionalities, such as naming, typically required by higher-level services.
- *Execution management services.* Ranging from simple jobs to complex workflows and composite services, these services deal with issues related to starting and managing tasks, including placement, provisioning, and life-cycle management.
- *Data management services.* These services handle issues such as data consistency, persistence, and integrity by providing functionality to move data to where it is needed, maintaining replicated copies, running queries and updates, and transforming data into new formats.

**In OGSA, a resource is a physical or logical entity that supports the use or operation of a computing application or environment.**

- *Security services.* Providing authentication, authorization, and integrity assurance, these services facilitate the enforcement of security-related policy within an organization and support safe resource sharing.
- *Resource management services.* These services provide management capabilities for grid resources: the resources themselves, resources as grid components, and the OGSA infrastructure.
- *Information services.* These services provide efficient production of, and access to, information about the grid and its constituent resources. Information refers to dynamic data or events for status monitoring, relatively static data for discovery, and any logged data.

### Infrastructure services

Infrastructure services are a set of common functionalities required by higher-level services. As OGSA builds on Web services technologies, the Web Services Description Language defines service interfaces. Infrastructure includes standards such as Web Services Resource Framework (WS-RF), WS-Management, and WS-Addressing.

Infrastructure services are concerned with resource naming, communication, and reflection. In OGSA, a resource is a physical or logical entity that supports the use

or operation of a computing application or environment. Resources are often stateful and provide a capability or capacity such as servers, networks, memory, applications, and databases. Resources might also handle dynamic entities, such as processes, print jobs, database query results, and virtual organizations.

OGSA builds heavily on existing WS standards and augments them with two specifications that address high-level service naming and binding.

**Naming.** Suppose there are two resources, A and B, and that A wishes to interact with B. How does A refer to B? OGSA defines a multilayer naming scheme of addresses, location-transparent identities, and pathnames.

OGSA uses the WS-Addressing specification for addressing. This specification defines an XML data structure called the *end point reference* that encapsulates the information the client needs to message a service. The EPR includes data such as a network protocol and address, an extensible metadata section to convey arbitrary suggestions such as security policies, and an opaque section for session/resource identifiers.

The Open Grid Forum's (OGF's) WS-Naming specification profiles WS-Addressing to provide location-transparent identities and name rebinding. An optional EPR metadata element called *end point identifier* supports location-transparent names. An EPI is a uniform resource identifier that is unique in space and time. Clients can compare the EPIs contained in two or more EPRs. If the EPIs are the same, the EPRs are said to point to the same entity. The semantics of the underlying service determine *sameness*. If the EPIs are different, nothing can be inferred.

The rebinding aspects of WS-Naming EPRs facilitate the implementation of the traditional distributed system transparencies: migration, failure, replication, and so forth. Embedded in an EPR's metadata element is an optional resolver EPR. A client can call the resolver EPR to acquire a new EPR for the service. For example, suppose there is a client *C* and service EPR *S* that contains a resolver EPR *R*. Suppose *S* migrates. Subsequent invocations of *S* by *C* will fault because *S* has moved. *C* can then invoke the resolution function to acquire a new EPR for *S*, *S'*, that is,  $S' = S.R.resolve()$ .



While EPRs are convenient for applications to manipulate, they can easily exceed hundreds of characters in length, making them awkward for humans to use. Further, the EPR namespace usually does not represent relationships between EPRs. To address these shortcomings and make grids more human friendly, the resource namespace service provides a hierarchical directory structure that maps string paths to EPRs much like a Unix directory maps string paths to inodes. For example, suppose there is an RNS path `/biology/Sequences/pir21.a`. RNS can map the human-readable path to a WS-Addressing EPR, and the EPR can directly access the resource.

### EMSs address problems with executing units of work, such as placement, “provisioning,” and lifetime management.

**Reflection and metadata.** Reflection is a critical infrastructure capability. Reflection here refers to the ability to discover properties or attributes of grid resources or services, for example, the port types, security mechanisms, and the provenance of data. Examples in use include WS-RF and WS-Metadata Exchange. The OGSA WS-RF Base Profile addresses selected WS-RF specifications including the operation `getResourceProperties`, which returns an XML document with the metadata associated with a resource.

#### Execution management services

Execution management services are concerned with the problems of instantiating and managing, to completion, units of work, including OGSA services or legacy applications. More formally, EMSs address problems with executing units of work, such as placement, “provisioning,” and lifetime management. These problems include, but are not limited to

- *Finding execution candidate locations.* What are the locations at which a unit of work can execute?
- *Selecting execution location.* Once it is known where a unit of work *can* execute, the question is where it *should* execute.
- *Preparing for execution.* This includes deployment and configuration of binaries and libraries, staging data, or other operations to prepare the local execution environment.
- *Initiating the execution.* Once preparation is complete, this includes starting the execution and carrying out related actions such as registering it in the appropriate places.

- *Managing the execution.* Once the execution is started, this includes managing and monitoring to completion. If it fails, should it be restarted in another location? Should there be periodic checkpoints to enable restarts?

These major issues that EMS must address cover the gamut of tasks and involve interactions with many other OGSA services—for example, provisioning, logging, registries, and security—that other OGSA capabilities are expected to define. To date, there are four OGF specifications related to execution management: Job Submission Description Language (JSDL), OGSA Basic Execution Services (BES), OGSA Resource Selection Services (RSS), and the High-Performance Computing (HPC) Basic Profile.

JSDL is an XML-based schema for describing applications, resources required for the application (for instance, memory, number and speed of CPUs), files to stage in before the application executes, files to stage out on completion, the command line string to execute, and so forth. JSDL delineates terms in the job description space and encourages definition of new terms.

The OGSA BES specification defines interfaces for creating, monitoring, and controlling computational entities such as Unix or Windows processes or parallel programs—known as activities. Clients identify activities using JSDL, and a BES implementation executes each activity that it accepts. A BES resource can represent a single computer; a cluster handled by a resource manager such as Load Leveler, Sun Grid Engine (SGE), Portable Batch System (PBS), Condor, or Platform’s Load Sharing Facility (LSF); or even another BES implementation.

To execute an activity, a client sends a JSDL document describing the activity to a BES resource. The BES resource either faults if it cannot execute the activity or returns an EPR that refers to the new activity. Subsequently, the client can either terminate or check the activity’s status.

The OGSA RSS specification defines an abstract interface for performing queries used to select resources for any purpose. The specification also narrows the abstract interface for the specific purpose of selecting a BES on which to instantiate an activity based on a JSDL document and classifies renderings for such services according to both the WS-RF and the WS-Transfer draft.

The HPC Basic Profile on OGSA BES and JSDL supports a minimal set of capabilities to satisfy a particular use case. The minimal set of operations does not include data staging, delegation, or application deployment. HPC Basic Profile File Staging, a subsequent profile, addresses file staging. Because there was not an agreed-upon security model at the time specification authors developed the document, the profile includes username-token and X.509 token credential profiles from the Web Services Interoperability (WS-I) Basic Security Profile (BSP) for authentication.

## Data management services

The yin to the execution management services' yang, data management services determine the process of describing, finding, storing, accessing, transferring, and managing data resources. The OGSA data architecture presents a toolkit of data services and interfaces that address multiple scenarios. These services and interfaces include data access, data transfer, storage management, data replication, data caching, and data federation.

Data management services address data access via Web services for both unstructured and structured data. OGSA ByteIO provides access to unstructured data, that is, sequences of bytes without type or interpretation. ByteIO provides Posix-like read and write operations on sequences of bytes. There are two variations of this interface. In the RandomByteIO interface, operations handle the offset, number of bytes, and data. In the StreamableByteIO interface, the operations do not take the offset.

In addition to providing access to structured data, for example, relational or XML databases, the WS Data Access and Integration family of specifications offers the ability to submit queries against structured data. One specification in the family covers each resource class—for example, RDBMSs, XML databases, flat files, and RDF triple stores. The client sets up a query and invokes it against a WS-DAI resource. The WS-DAI resource then parses and executes the query, creates a new WS-DAI resource, and returns the EPR of the new resource. That returned resource implements a suitable interface for accessing data, typically WS-DAI. For example, in WS Data Access and Integration Relational, the result of a SQL query is another WS-DAIR resource that contains the resulting table.

Note that a resource can implement more than one port type. In the above WS-DAIR example, the returned resource can support both the WS-DAIR port type and the ByteIO port type. Thus, the client can both query the resource and read it as a text file. OGSA ByteIO and WS-DAI resources are addressed using WS-Addressing EPRs. As such, clients can easily place both resource types into RNS namespaces and access them using RNS pathnames rather than EPRs.

Along with computing and networking resources, data storage resources are one of the basic building blocks of a distributed computing infrastructure. Often, for performance reasons, storage resources need to be directly accessible using non-WS means, for example, using direct operating system read/write calls. There are many kinds of storage resources, ranging from a memory stick to a multipetabyte tape silo. The OGSA data architecture documents describe using storage resource management. SRM services provide space to either store files or function as a formatted and mounted raw device or block device.

## Security services

Security is a system's ability to protect the assets of its users and those of its resource providers. At the most fundamental level, the assets within a service-oriented grid architecture are the resources exposed by service clients and end points. Such resources can exist in the form of information and data, communication and data processing services, controls for equipment and facilities, and so forth.

Following the Open Systems Interconnect threat model, the types of security threat to which grid resources are vulnerable include disclosure or theft of resources, modification (including destruction) of resources, and resource

### The OGSA data architecture documents describe using storage resource management.

service interruption. Because of these and other threats, an effective security model is paramount to the adoption of the OGSA. Without a commitment to meaningful security, many potential adopters would be unable to participate because of undue risk or legal restrictions.

The biggest challenges of architecting a practical grid security model arise from the fact that often the users and resources participating within a grid are equipped with an existing security policy and mechanism for authorized behavior. OGSA's site autonomy theme posits that organizational domains will retain control over these security policies, even as they might need adjustment to accommodate new vulnerabilities arising from exposure through grid service interfaces.

The WS-Security family of specifications defines a general-purpose mechanism for associating security credentials with message content, then constructing a set of specific profiles for encoding popular token types—for example, X.509, Kerberos, Security Assertion Markup Language (SAML), and username-token credentials. The WS-Security core specification also defines the application of XML encryption and XML digital signature to provide end-to-end messaging integrity and confidentiality without the support of the underlying communication protocol. To achieve real-world interoperability, the WS-I BSP provides guidance on the use of WS-Security and its associated security token formats to resolve nuances and ambiguities between communicating implementations intending to leverage common security mechanisms.

OGSA envisions the highly dynamic coupling of grid resources, which makes interoperability further dependent on the ability to normatively describe and advertise individual secure communication requirements that affect

message format. OGF has developed two supplemental security profiles that provide guidance for the expression and conveyance of secure communication requirements.

The OGF Secure Communication Profile (SecComm) is a profile of WS-Security Policy, a flexible, extensible approach for specifying the security tokens, cryptographic algorithms, and protocol mechanisms needed for secure communication. More specifically, the SecComm profile describes how to convey cryptographic key material (for example, digital certificates) and versioning information within policy assertions, clarifies the semantics for several WS-Security Policy assertions, and profiles normative “well-known” policy documents that identify commonly used security mechanisms.

**OGF has developed two supplemental security profiles that provide guidance for the expression and conveyance of secure communication requirements.**

To facilitate the distribution of communication requirements, the OGF Secure Addressing Profile (SecAddr) describes the attachment of such security policy within WS-Addressing end point references. The WS-Addressing end point reference data structure is a useful construct because it provides the “invocation context” for a service end point—the necessary information that a client requires to establish meaningful communication. In addition, the SecAddr profile describes the digital signature of EPR documents to provide guarantees of trust regarding the identity of the minter and the integrity of the EPR—useful properties given the OGSA models of storing and exchanging EPRs within intermediary services.

### Resource management and information services

Because they are dependent on infrastructure, execution management, data management, and security services, resource management and information services are now receiving attention.

In concert with the Distributed Management Task Force, the OGSA working group (WG) has developed a computer-integrated manufacturing-based resource model for OGSA resources. The result is OGF Grid Final Document (GFD; [www.ggf.org/gf/docs](http://www.ggf.org/gf/docs)) 119, “Execution Environment and Basic Execution Service Model in OGSA Grids.” The document provides Unified Modeling Language and Microsoft Operations Framework definitions of OGSA BES concepts mapped to CIM. GFD 45 contains a survey of issues in resource management in grids.

In terms of information services, a significant challenge in grids is understanding the system’s state. This state can include metadata or attributes resources, their interfaces, current status, event logs, and policies. Not only is the sheer scale of the number of resources (tens of billions in a large system) daunting, making collection, organization, and querying of the state difficult, but the problem of data currency is present. The basic problem in grids is determining how to collect and manage resource metadata to support queries against the data for resource discovery, system management, or other tasks. There have been several solutions for this in grids, but no consensus has emerged for a standard.

### PUTTING THE PIECES TOGETHER

The specifications described here are sufficient to realize several grid use cases, including high-throughput computing, a federated data environment, and service mobility.

#### High-throughput computing

High-throughput computing is one of the most common uses cases in grids today. The challenge is distributing a large number of jobs onto a set of computational resources that can span multiple administrative domains and file systems. For example, a researcher wants to screen a potential new cancer drug against a large number of targets. An application runs for each potential target, which can amount to tens of thousands of application instances.

The most basic specifications for this use case are JSDL and OGSA BES. Compute resources are represented by BES resources that can run the applications. A simple run command generates an appropriate JSDL document for the needed execution and sends the JSDL document in a round-robin fashion to one of a number of preconfigured BES resources. The BES resources can receive RNS pathnames to simplify administration. The BES resources might wrap a Globus Gram; gLite Compute Element; Unicore gateway; or a PBS, SGE, or LSF queue. Note that while run is out of the scope of OGSA specifications, implementations of run tools exist.<sup>5</sup> The JSDL document can contain data staging elements specifying where to fetch input files and where to place the results.

Round-robin job placement on a preselected set of BES resources is rigid and might not reflect different application requirements or organizational policies. Another scenario would be implementing a BES resource that, rather than directly executing JSDL documents itself, proxies for a large number of other BES resources and schedules jobs on those containers in first-in, first-out order. The run command now simply sends the JSDL document to the queue, similar to BES resources. Alternatively, a metascheduling BES resource might match the job requirements in the JSDL document, such as CPU architecture, against the resource properties of the BES resources.

This use case also requires security management. Users log in to their local domain's identity provider, as per WS-Trust, to obtain credential tokens. Alternatively, they might already have their own tokens locally, such as an X.509 certificate and corresponding private key on a smart card device.

When users select a BES resource and obtain an EPR for it, the EPR contains a WS-Security Policy section (as profiled by the WS-Secure Addressing) within its metadata that contains secure communication information such as cryptographic identity (for example, an X.509 certificate), the types of security tokens required (for example, X.509, SAML, or username-token), and the secure communication actions (for example, transport- and message-level cryptographic algorithms).

With credential types suitable for delegation, the meta-scheduling BES can participate in the further brokering of identity in the event that its resources are located within yet another administrative domain.

### Access to federated data

Data storage and management can take place in several different locations. Users might not be able to directly access the data using standard operating environments such as Unix or Windows due to disjoint administrative domains or unwillingness to provide accounts between suborganizations. Further, copying data to a single data warehouse might not be practical. Whatever the reason, a mechanism is required to access data directly.

To realize this use case, we start by assuming the data exists as unstructured flat files located in a rooted directory tree stored in a Windows or Unix file system. The first step is to create a set of ByteIO EPRs that correspond to the files and RNS EPRs that correspond to the directories in the rooted tree. This set of ByteIO and RNS end points represents an RNS namespace in which the leaves are ByteIO files. We call this step "exporting" a directory structure. The exported directory structure then links to an existing RNS namespace in a manner similar to a Unix link. Client applications access data regardless of physical location by issuing the appropriate ByteIO read and write calls.

Developers can add structured relational data by including WS-DAI EPRs in the RNS namespace. Each WS-DAI EPR can refer to a different table or set of tables. SQL queries can be executed against each WS-DAI end point. The result is another WS-DAI EPR that represents the result of query execution. Thus, RNS can find an EPR for a particular dataset and issue a WS-DAI query; the resulting EPR can reside in yet another RNS directory for subsequent use. If the returned WS-DAI resource also implements the OGSA ByteIO port type, clients could then read the result set as a file.

The security aspects for this use case are analogous to those of the high-throughput computing use case. EPRs

to data sources will advertise the security token requirements, cryptographic actions, claims, and references to interdomain security token services for credential brokering. Authorization policies in place at the services' end points determine which actions are allowed to proceed.

So far we've made no assumptions about how the client accesses the data—whether through a C library call, user-defined Web services call, or some other mechanism. While we could use a special grid API such as OGF's Simple API for Grid Applications, the problem is that many users can't change their applications to access the data grid resources via a new API. Instead, developers must hide the grid from users and applications so they can access data without application modification.

**Developers must hide the grid from users and applications so they can access data without application modification.**

One solution is to use one or more grid-aware mountable file systems. The basic idea is simple: Create a grid-aware Windows Installable File System, Network File System, Common Internet File System, or Filesystem in User Space file service and map the grid in the local file system. Reads and writes progress through standard operating system interfaces and pass through the operating system to the grid-aware file system, which then interacts with the grid using Web services.

The advantage of this approach is that user applications and shell scripts can execute without any modifications. A challenge is properly associating local operating system identities with grid identities. In the simplest case, each grid-aware file system plug-in instance corresponds to a particular single user and has a properly delegated credential for that user.

### Resource mobility

While existing Web services best practices support heterogeneity, concurrency, and behavioral transparency, the use of name rebinding mechanisms as defined in WS-Naming can provide a framework for realizing additional important transparencies—in particular, migration, location, replication, and failure transparency.

Four real-world use cases highlight the importance of supporting these additional transparencies:

- *Migrating closer to active users.* Suppose a client application is intensely using a resource that is located far away—for example, an application in California that

**Table 1. Specification adoption.**

	WS-Naming	RNS	OGSA WS-RF BP	OGSA ByteIO	WS-DAI	JSDL	OGSA BES	HPC Basic Profile
Crown	No	No	No	No	No	Yes	No	Yes
Fujitsu USMT	Will	No	Yes	Yes	No	Yes	Yes	Yes
Genesis II	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Gfarm	No	Yes	No	Will	No	No	No	No
gLite3	No	Yes	No	Yes	No	Yes	Yes	Yes
Globus	Yes	No	Yes	No	Yes	Yes	Yes	Yes
GridSAM	No	No	No	No	No	Yes	Yes	Yes
Microsoft CCS	No	No	No	No	No	Yes	Yes	Yes
NAREGI	No	No	No	No	Yes	Yes	Yes	No
OGSA DAI	No	No	Will	Yes	Yes	No	No	No
Platform	No	No	No	No	No	Yes	Yes	Yes
Unicore 6	No	No	Yes	Yes	Will	Yes	Yes	Yes

reads and modifies a shared file (ByteIO) resource currently residing in New York. In this configuration, the application might be suffering unnecessarily from poor performance due to high network latency. Ideally, the file resource should migrate from New York to California without any service interruption to other users that need access to the shared file.

- *Migrating away from failing or overloaded systems.* Consider a service or resource executing on a host that is heavily loaded in some way—perhaps the host CPU is overloaded or the network into the host is flooded. There might be problems with the physical environment such as a power shortage or air conditioning failure, or a host might be planning downtime because of maintenance. The service should migrate to another host without interrupting the service and without disrupting ongoing interactions with this and other services and resources.
- *Recovery from a failed resource.* Consider a stateful resource that has failed (for example, due to a hardware or software failure) and needs to be restarted, possibly on a different physical resource. The resource instance should migrate to a different location or machine while minimizing interruptions for accessing the resource.
- *Replica management and usage.* A resource might have multiple back-end end points that can each perform its services with an option to dynamically select which replica to use. For example, one replica

might be closer (in network terms) to the end user than another or one replica might offer better quality of service in some dimension such as performance.

When combined with WS-Naming-aware stubs, the WS-Naming rebinding services can realize each of these use cases. The EPR of resource *A* includes the EPR of a service *R*, which can acquire a new EPR for *A* when *A* migrates or fails.

**ADOPTION**

OGF and OGSA specification and profile authors represent a variety of organizations from around the world. Projects and products that incorporate OGSA and OGF specifications come from equally diverse representatives from industry, government, and academia. Table 1 captures the state of adoption in late 2007. Each cell has one of three values: Yes, Will, or No. Yes means the project has implemented or is in the process of implementing the specification. Will means the project is planning an implementation but has not started. No means that the project did not plan to implement the specification as of late 2007.

Several interoperation events have taken place, including at Supercomputing 2006 and 2007. These events tend to show interoperation on a particular specification or profile or a set of related specifications and profiles, for example, JSDL, OGSA BES, and HPC Basic Profile.

**A**fter several years of development, consensus building, and experimentation, the Open Grid Services Architecture has reached the state where there are sufficient specifications and profiles to enable the construction of interoperable grids. The specifications cover a range of use cases and requirements from basic infrastructure services—such as naming and binding—to execution management, data management, and security. Developers can use the specifications to construct secure, interoperable compute and data grids that meet a variety of real-world high-throughput computing and data-sharing use cases in the pharmaceutical, financial services, electronic design automation manufacturing, and aerospace industries as well as several use cases in the sciences and engineering.

In addition, several organizations are adopting the specifications. These organizations, from the commercial, academic, government, and open source communities, have developed and continue to develop OGF and OGSA specifications.

OGSA is by no means complete. While it addresses key core services, it does not currently address many critical services such as auditing, transactions, service-level agreements, resource discovery, and information services workflows. It is not the intent of the OGSA WG to define all of these but rather to adopt and profile existing specifications as appropriate and, when necessary, develop new specifications and schemata. To learn more about OGSA, visit [www.ogf.org/OGSA\\_primer.php](http://www.ogf.org/OGSA_primer.php). To download and experiment with an open source implementation of the OGSA specifications, visit [www.cs.virginia.edu/~vcgr](http://www.cs.virginia.edu/~vcgr). 

## Acknowledgments

The entire OGSA regular team: Fred Maciel, Jay Unger, Ellen Stokes, Jem Tredwell, Steve Newhouse, Allen Luniewski, Stephen McGough, Chris Kantarjiev, Tom McGuire, Ian Foster, Ravi Subramanian, Donal Fellows, Michel Drescher, Mike Behrens, Chris Jordan, Frank Siebenlist.

This material is based on work partially supported by the National Science Foundation under grant no. 0426972. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## References

1. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufman, 1999.
2. I. Foster et al., “Grid Services for Distributed System Integration,” *Computer*, June 2002, pp. 37-46.
3. A. Grimshaw and A. Natrajan, “Legion: Lessons Learned Building a Grid Operating System,” *Proc. IEEE*, 2005, pp. 589-603.

4. I. Foster et al., “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration,” white paper, Globus Project, Jan. 2002.
5. M. Morgan and A. Grimshaw, “Genesis II—Standards Based Grid Computing,” *Proc. 7th IEEE Int’l Symp. Cluster Computing and the Grid*, IEEE Press, 2007, pp. 611-618.

**Andrew Grimshaw** is a professor in the Department of Computer Science at the University of Virginia. He received a PhD in computer science from the University of Illinois at Urbana-Champaign. Grimshaw is a member of the OGF. Contact him at [grimshaw@virginia.edu](mailto:grimshaw@virginia.edu).

**Mark Morgan** is a member of the research faculty for the Department of Computer Science at the University of Virginia. He received an MS in computer science from the University of Virginia. Morgan is a member of the OGF. Contact him at [mmm2a@virginia.edu](mailto:mmm2a@virginia.edu).

**Duane Merrill** is a graduate student in computer science at the University of Virginia. He received a master’s degree in computer science from the University of Virginia. Merrill is a member of the OGF. Contact him at [dgm4d@virginia.edu](mailto:dgm4d@virginia.edu).

**Hiro Kishimoto** is a senior research fellow at Fujitsu Laboratories and visiting professor at the National Institute of Informatics. He received a PhD in computer science from Tohoku University. He is a member of the IEEE Computer Society and Information Processing Society of Japan. Contact him at [hiro.kishimoto@jp.fujitsu.com](mailto:hiro.kishimoto@jp.fujitsu.com).

**Andreas Savva** is a researcher at Fujitsu Laboratories. He received a Dr. Eng. in computer science from the Tokyo Institute of Technology. Savva is a member of the ACM and IEEE. Contact him at [andreas.savva@jp.fujitsu.com](mailto:andreas.savva@jp.fujitsu.com).

**David Snelling** is the assistant division manager at Fujitsu Laboratories of Europe. He received a PhD in computer architecture from the University of Manchester. Snelling is a member of the IEEE Computer Society. Contact him at [david.snelling@uk.fujitsu.com](mailto:david.snelling@uk.fujitsu.com).

**Chris Smith** is principal product architect at Platform Computing. He received BSc in computer science from the University of British Columbia. Smith is a member of the ACM. Contact him at [csmith@platform.com](mailto:csmith@platform.com).

**Dave Berry** is the deputy director for research and e-infrastructure development at the UK National e-Science Centre. He received a PhD in computer science from the University of Edinburgh. Berry is a member of the ACM and the British Computer Society. Contact him at [Dave.Berry@ed.ac.uk](mailto:Dave.Berry@ed.ac.uk).