Grid Services for Distributed System Integration

The Open Grid Services Architecture enables the integration of services and resources across distributed, heterogeneous, dynamic virtual organizations—whether within a single enterprise or extending to external resource-sharing and service-provider relationships.

Ian Foster
Argonne National Laboratory

Carl Kesselman
University of Southern California

Jeffrey M. Nick
IBM

Steven Tuecke
Argonne National Laboratory

Increasingly, computing addresses collaboration, data sharing, cycle sharing, and other modes of interaction that involve distributed resources. This trend results in an increased focus on the interconnection of systems both within and across enterprises. In addition, companies are realizing that they can achieve significant cost savings by outsourcing nonessential elements of their IT environment to various forms of service providers.

These evolutionary pressures generate new requirements for distributed application development and deployment. Today, applications and middleware developers typically target a specific platform—such as Windows NT, some flavor of Unix, a mainframe, Java 2 Enterprise Edition (J2EE), or Microsoft .NET—that provides a hosting environment for running applications. Such platforms provide capabilities ranging from integrated resource management functions to database integration, clustering services, security, workload management, and problem determination—with different platforms offering different implementations, semantic behaviors, and APIs.

The continuing decentralization and distribution of software, hardware, and human resources make it essential that we achieve the desired quality of service (QoS) on resources assembled dynamically from enterprise, service provider, and customer systems despite this diversity. This requires new abstractions and concepts that let applications access and share resources and services across distributed, wide area networks, while providing common security semantics, distributed resource management performance, coordinated fail-over, problem determination services, or other QoS metrics that are of importance in a particular context.

For some time, such problems have been of central concern to developers of distributed systems for large-scale scientific research. Work within this community has led to the development of *Grid technologies*,[1] which have been widely adopted in scientific and technical computing.[2] Grid technologies and infrastructures support the sharing and coordinated use of diverse resources in dynamic, distributed *virtual organizations*[3]—that is, the creation, from geographically distributed components operated by distinct organizations with differing policies, of virtual computing systems that are sufficiently integrated to deliver the desired QoS.

In particular, the open source Globus Toolkit described in the "OGSA and the Globus ToolKit" sidebar has emerged as a de facto standard for construction of Grid systems. Projects building on the Globus Toolkit range from scientific collaborations concerned with remote access to specialized experimental facilities—for example, the Network for Earthquake Engineering Simulation, NEESgrid; (http://www.neesgrid.org)—to "data grids" for the

0018-9162/02/$17.00 © 2002 IEEE

distributed analysis of large amounts of data—for example, the Grid Physics Network (http://www.griphyn.org); EU DataGrid Project (http://www.eu-datagrid.org); and the Particle Physics Data Grid (http://www.ppdg.net).

Grid technologies, and the Globus Toolkit in particular, are evolving toward an Open Grid Services Architecture (OGSA)[4] in which a Grid provides an extensible set of services that virtual organizations can aggregate in various ways. Building on concepts and technologies from both the Grid and Web services[5] communities, OGSA defines a uniform exposed service semantics (the *Grid service*); defines standard mechanisms for creating, naming, and discovering transient Grid service instances; provides location transparency and multiple protocol bindings for service instances; and supports integration with underlying native platform facilities.

OGSA also defines, in terms of Web Services Description Language (WSDL)[6] interfaces and associated conventions, mechanisms required for creating and composing sophisticated distributed systems, including lifetime management, change management, and notification. Service bindings can support reliable invocation, authentication, authorization, and delegation.

The development of OGSA technical specifications is ongoing within the Global Grid Forum (http://www.gridforum.org), a Grid community and standards organization, and the Globus Project is developing an open source reference implementation. We expect to see both an OGSA-based Globus Toolkit and OGSA-based commercial products by the end of 2002.

GRID TECHNOLOGIES ENTER THE MAINSTREAM

The World Wide Web began as a technology for scientific collaboration and was later adopted for e-business. We foresee—and indeed are experiencing—a similar trajectory for Grid technologies.

The scientific resource sharing applications that motivated the early development of Grid technologies include the pooling of expertise through collaborative visualization of large scientific data sets, the pooling of computer power and storage through distributed computing for computation-

OGSA and the Globus Toolkit

The Globus Toolkit[1] is a community-based, open architecture, open source set of services and software libraries that support Grids and Grid applications. The toolkit addresses issues of security, information discovery, resource management, data management, communication, and portability. Globus Toolkit mechanisms are in use at hundreds of sites and by dozens of major Grid projects worldwide.[2]

OGSA represents a natural evolution of the second version of the Globus Toolkit, GT2. Key concepts, such as factory, registry, and reliable and secure invocation exist in the Globus Toolkit but in a less general and flexible form and without the benefits of a uniform interface definition language. In effect, OGSA refactors key design elements so that, for example, it uses common notification mechanisms for service registration and service state. OSGA further abstracts these elements so that they can be applied at any level to virtualize VO resources.

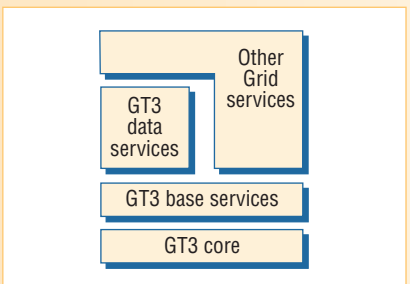

Figure A. GT3 structure.

The Globus Project is evolving the Globus Toolkit code base to exploit OGSA capabilities. The result of this evolution will be Globus Toolkit version 3 (GT3). Figure A illustrates GT3's structure, which includes

- the GT3 core, which implements the Grid service interfaces and behaviors;
- GT3 base services, which exploit the GT3 core to implement both existing Globus Toolkit capabilities (for example, resource management, data transfer, and information services) and new capabilities (for example, reservation and monitoring); and
- higher-level services that may target both GT3 core and GT3 base services such as data management, workload management, and diagnostics.

The first GT3 Core prototype was made available in May 2002, and a full GT3 release is planned for around the end of 2002. This release will be a full open source OGSA implementation that supports existing Globus APIs as well as WSDL interfaces, as described at http://www.globus.org/ogsa.

References

1. I. Foster and C. Kesselman, "Globus: A Toolkit-Based Grid Architecture," *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, eds., Morgan Kaufmann, San Francisco, 1999, pp. 259-278.
2. I. Foster, "The Grid: A New Infrastructure for 21st Century Science," *Physics Today*, vol. 55, no. 2, 2002, pp. 42-47.

ally demanding data analyses, and increasing functionality and availability by coupling scientific instruments with remote computers and archives.[1,7] We expect similar applications to become important in commercial settings—initially for scientific and technical computing applications, where we can already point to success stories—and then for commercial distributed computing applications.

However, we expect that rather than enhancing raw capacity, the most important role for Grid concepts in commercial computing will be to offer solutions to new challenges that relate to the construction of reliable, scalable, and secure distributed systems. These challenges derive from the current rush, driven by technology trends and commercial pressures, to decompose and distribute through the network previously monolithic host-centric services.

The evolution of enterprise computing

In the past, organizations performed computing tasks in highly integrated enterprise computing centers. Although sophisticated distributed systems existed, such as command-and-control and reservation systems, and the Internet Domain Name System, these were specialized, niche entities. The Internet's rise and the emergence of e-business have, however, led to a growing awareness that an enterprise's IT infrastructure also encompasses external networks, resources, and services.

Initially, developers treated this new source of complexity as a network-centric phenomenon and attempted to construct *intelligent networks* that intersected with traditional enterprise IT data centers only at *edge servers*—an enterprise's Web point of presence or the virtual private network server that connects an enterprise network to service provider resources, for example. These developers worked from the assumption that these servers could thus manage and circumscribe the impact of e-business and the Internet on an enterprise's core IT infrastructure.

These attempts have generally failed because IT services decomposition is also occurring *inside* enterprise IT facilities. New applications are being developed for programming models, such as the Enterprise JavaBeans component model, that insulate the application from the underlying computing platform and support portable deployment across multiple platforms. Thus, for example, Web serving and caching applications target commodity servers rather than traditional mainframe computing platforms. Meanwhile, Web access to enterprise resources requires ever-faster request servicing, further driving the need to distribute and cache content closer to the network's edge.

The overall result is decomposition of a highly integrated internal IT infrastructure into a collection of heterogeneous and fragmented systems, often operated by different business units. Enterprises must then reintegrate these distributed servers and data resources with QoS, addressing issues of navigation, distributed security, and content distribution *inside* the enterprise as well as on external networks.

In parallel with these developments, enterprises require an increasingly robust IT infrastructure to handle the unpredictability and rapid growth associated with e-business ventures. Businesses are also expanding the scope and scale of their enterprise resource planning projects as they try to achieve better integration with customer-relationship-management, integrated-supply-chain, and existing core systems.

Enterprises require an increasingly robust IT infrastructure to handle the unpredictability and rapid growth associated with e-business ventures.

These developments have the aggregate effect of making the QoS traditionally associated with mainframe host-centric computing essential to the effective conduct of e-business across distributed computing resources, both inside and outside the enterprise. For example, enterprises must provide consistent response times to customers, despite workloads with significant deviations between average and peak utilization. Thus, they require flexible resource allocation in accordance with workload demands and priorities. Yet the current paradigm for delivering QoS to applications via the vertical integration of platform-specific components and services does not work in today's distributed environment: The decomposition of monolithic IT infrastructures is inconsistent with the delivery of QoS through vertical integration of services on a given platform.

Service providers and business-to-business computing

Another key IT trend is the emergence of various types of Web hosting, content distribution, applications, and storage service providers (SPs). By exploiting economies of scale, SPs aim to provide standard e-business processes, such as creation of a Web portal presence, to multiple customers with superior price and performance. Enterprises want to offload such processes because they view them as commodity functions.

Such emerging *e-utilities*—service providers who offer continuous, on-demand access—are beginning to offer a model for carrier-grade IT resource deliv-

Enterprise computing systems operate within virtual organizations with similarities to the scientific collaborations that originally motivated Grid computing.

ery through metered usage and subscription services. Unlike yesterday's computing services companies, which tended to provide offline batch-oriented processes, today's e-utilities often provide resources that both enterprise computing infrastructures and in-house and outsourced business processes use. Thus, one consequence of exploiting the economies of scale that e-utility structures enable is further decomposition and distribution of enterprise computing functions.

To achieve economies of scale, e-utilities require a server infrastructure that can be easily customized on demand to meet specific customer needs and an IT infrastructure that

- supports dynamic resource allocation in accordance with service-level agreement policies, efficient sharing and reuse of the IT infrastructure at high utilization levels, and distributed security from the network edge to application and data servers; and
- delivers consistent response times and high levels of availability—which in turn drive a need for end-to-end performance monitoring and real-time reconfiguration.

A final key IT industry trend is cross-enterprise business-to-business collaboration such as multiorganization supply chain management, virtual Web malls, and electronic market auctions. B2B relationships are, in effect, virtual organizations—albeit with particularly stringent requirements for security, auditability, availability, service-level agreements, and complex transaction-processing flows. Thus, B2B computing represents another source of demand for distributed systems integration, often characterized by large differences among the information technologies that different organizations deploy.

OPEN GRID SERVICES ARCHITECTURE

Enterprise computing systems must increasingly operate within virtual organizations (VO) with similarities to the scientific collaborations that originally motivated Grid computing. Depending on the context, the dynamic ensembles of resources, services, and people that comprise a scientific or business VO can be small or large, short- or long-lived, single- or multi-institutional, and homogeneous or heterogeneous. Individual ensembles can be structured hierarchically from smaller systems and may overlap in membership.

Regardless of these differences, VO application developers face common requirements as they seek to deliver QoS—whether measured in terms of common security semantics, distributed workflow and resource management, coordinated fail-over, problem determination services, or other metrics—across a collection of resources with heterogeneous and often dynamic characteristics.

Service orientation

The Open Grid Services Architecture (OGSA) [3] supports the creation, maintenance, and application of the service ensembles that VOs maintain. OGSA adopts a common representation for computational and storage resources, networks, programs, databases, and the like. All are treated as *services*—network-enabled entities that provide some capability through the exchange of messages. Arguably, we could use the term *object* instead, as in systems like SOS[8] and Legion,[9] but we avoid it because of its overloaded meaning and because OGSA does not require object-oriented implementations. Adopting this uniform service-oriented model makes all components of the environment virtual—although the model must be grounded on implementations of physical resources.

This service-oriented view partitions the interoperability problem into two subproblems: the definition of service interfaces and the identification of protocols that can invoke a particular interface. A service-oriented view addresses the need for standard interface definition mechanisms, local and remote transparency, adaptation to local OS services, and uniform service semantics. A service-oriented view also simplifies virtualization through encapsulation of diverse implementations behind a common interface.

Virtualization

Virtualization enables consistent resource access across multiple heterogeneous platforms. Virtualization also enables mapping of multiple logical resource instances onto the same physical resource and facilitates management of resources within a VO based on composition from lower-level resources. Further, virtualization lets us compose basic services to form more sophisticated services—without regard for how these services are implemented.

Virtualizing Grid services also underpins the ability to map common service semantic behavior seamlessly onto native platform facilities. This virtualization is easier if we can express service functions in a standard form, so that any implementation of a service is invoked in the same man-

ner. We adopt the Web Services Description Language (WSDL) for this purpose.

WSDL distinguishes between the service interface definition and the protocol bindings used for service invocation; a single interface can have multiple bindings, including both distributed communication protocols such as HTTP and locally optimized bindings such as a local IPC for interactions on the same host.

Other binding properties can include reliability and other forms of QoS, as well as authentication and delegation of credentials. The choice of binding should always be transparent to the requestor with respect to service invocation semantics, but not with respect to other things—for example, a requestor should be able to choose a particular binding for performance reasons.

The service interface definition and access binding are also distinct from the service implementation. A service can support multiple implementations on different platforms, facilitating seamless overlay not only to native platform facilities but also, via the nesting of service implementations, to virtual resource ensembles. For example, depending on the platform and context, we might use the following implementation approaches:

- construct a reference implementation for full portability across multiple platforms to support the execution environment for hosting a service;
- use a platform possessing specialized native facilities for delivering service functionality to map from the service interface definition to the native platform facilities; or
- apply these mechanisms recursively, constructing a higher-level service from the composition of multiple lower-level services, which themselves can either map to native facilities or decompose further.

The service implementation for the third option would then dispatch operations to lower-level services.

The ability to adapt to operating system functions on specific hosts is central to virtualization of resource behaviors. Enabling exploitation of native capabilities presents a significant challenge when developing these mappings—whether we focus on performance monitoring, workload management, problem determination, or enforcement of native platform security policy—so that the Grid environment does not become the least common denominator of its constituent pieces. Service discovery mechanisms are important in this regard, allowing higher-level services to discover what capabilities a particular interface implementation supports. For example, if a native platform supports reservation capabilities, a resource-management interface implementation can exploit those capabilities.

The ability to adapt to operating system functions on specific hosts is central to virtualization of resource behaviors.

Thus, our service architecture supports local and remote transparency with respect to service location and invocation. It also provides multiple protocol bindings to facilitate localized optimization of services invocation when the service is hosted locally with the service requestor. In addition, it enables protocol negotiation for network flows across organizational boundaries to allow choosing between several interGrid protocols, each optimized for a different purpose. Finally, the implementation of a particular Grid service interface can map to native, nondistributed, platform functions and capabilities.

Service semantics: The Grid service

Our ability to virtualize and compose services depends on more than standard interface definitions. We also require standard semantics for service interactions so that, for example, we have standard mechanisms for discovering service properties and different services follow the same conventions for error notification. To this end, OGSA defines a *Grid service*—a Web service that provides a set of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; the conventions address naming and upgradeability. Grid services also address authorization and concurrency control. This core set of consistent interfaces, from which we implement all Grid services, facilitates the construction of hierarchal, higher-order services that can be treated uniformly across layers of abstraction.

As Figure 1 shows, a set of interfaces configured as a WSDL portType defines each Grid service. Every Grid service must support the GridService interface; in addition, OGSA defines a variety of other interfaces for notification and instance creation. Of course, users also can define arbitrary application-specific interfaces. The Grid service's serviceType, a WSDL extensibility element, defines the collection of portTypes that a Grid service supports, along with some additional information relating to versioning.

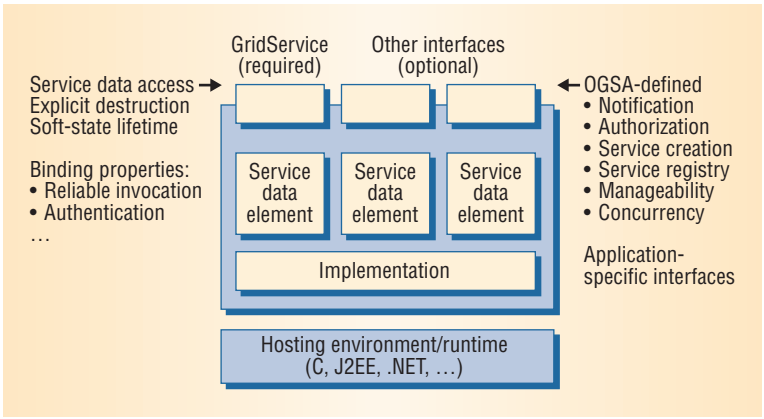

Figure 1. OGSA Grid service. The service consists of data elements and various required and optional interfaces, with potential instantiation via different implementations, possibly in different hosting environments.

Associated with each interface is a potentially dynamic set of *service data elements*—named and typed XML elements encapsulated in a standard container format. Service data elements provide a standard representation for information about Grid service instances. This important aspect of the OGSA model provides the basis for discovery and management of potentially dynamic Grid service properties. Finally, as Figure 1 shows, users can implement a particular Grid service—as defined by its interfaces and associated service data elements—in a variety of ways and host it in different environments.

Grid services can maintain internal state for their lifetime. The existence of state distinguishes one instance of a service from another instance that provides the same interface. The term *Grid service instance* refers to a particular instantiation of a Grid service. The interfaces and conventions that define a Grid service are concerned, in particular, with behaviors related to the management of *transient service instances*.

VO participants often want to instantiate new transient service instances dynamically to handle the management and interactions associated with the state of particular requested activities. When the activity's state is no longer needed, the service can be destroyed. For example, in a videoconferencing system, establishing a videoconferencing session might involve creating service instances at intermediate points to manage end-to-end dataflows according to QoS constraints. A Web serving environment might instantiate services dynamically to provide a consistent user response time by managing application workload through dynamically added capacity.

Other examples of activities that can be represented and managed as transient service instances are a query against a database, a data mining operation, a network bandwidth allocation, a running data transfer, and an advance reservation for processing capability. These examples emphasize that service instances can be extremely lightweight entities, created to manage even short-lived activities.

Because Grid services are dynamic and stateful, we need a way to distinguish one dynamically created service instance from another. Thus, every Grid service instance receives a globally unique name, the *Grid service handle*. This handle distinguishes a specific Grid service instance from all other Grid service instances that have existed, exist now, or will exist in the future.

An attractive feature of WSDL is that it allows the definition of multiple protocol bindings for a particular interface. A protocol binding can define delivery semantics that address, for example, reliability. Services interact with one another by exchanging messages. In distributed systems prone to component failure, however, we can never guarantee that a sent message has been delivered. The existence of internal state can make it important that we guarantee a service has received a message once or not at all, even with failure-recovery mechanisms such as *retry* in use. In such situations, using a protocol that guarantees *exactly-once* delivery or similar semantics can be desirable, as can the protocol-binding behavior of mutual authentication during communication. OGSA is defining such bindings.

Standard interfaces. OGSA defines standard behaviors and associated interfaces.

Discovery. Applications require mechanisms for discovering available services, determining their characteristics, and configuring themselves and their requests to those services. In addition to the service data element, which defines a standard representation for information about Grid service instances, OGSA defines a standard operation, FindServiceData, which retrieves service information from individual Grid service instances, and a standard interface for registering information about Grid service instances with registry services.

Dynamic service creation. The ability to dynamically create and manage new service instances, a basic tenet of the OGSA model, necessitates using service-creation services. The model defines a standard interface, Factory, and semantics that any service-creation service must provide.

Lifetime management. Because OGSA services can be created and destroyed dynamically, they can be destroyed explicitly. They also can be destroyed or become inaccessible through a system failure such as an operating system crash or a network partition. Interfaces are defined for managing a service's life-

time and, in particular, for reclaiming the services and state associated with failed operations. For example, termination of a videoconferencing session might also require the termination of services created at intermediate points to manage flows.

OGSA addresses this requirement by defining a standard SetTerminationTime operation within the required GridService interface for soft-state lifetime management of Grid service instances. Soft-state protocols[10] let OGSA eventually discard the state established at a remote location unless a stream of subsequent *keepalive* messages refreshes it. Such protocols have the advantages of being both resilient to failure—a single lost message need not cause irretrievable harm—and simple because they require no reliable discard protocol message. The GridService interface also defines an Explicit-Destruction operation.

Notification. A collection of dynamic, distributed services must be able to notify each other asynchronously of significant changes to their state. OGSA defines common abstractions and service interfaces for subscription to and delivery of such notifications, so that services constructed by the composition of simpler services can deal in standard ways with notifications of, for example, errors. Specialized protocol bindings can allow OGSA notifications to exploit various commonly and commercially available messaging systems for the delivery of notification messages with a particular QoS.

Manageability. In operational settings, we may need to monitor and manage potentially large sets of Grid service instances. A manageability interface defines relevant operations.

Role of hosting environments

OGSA defines the semantics of a Grid service instance: how it is created and named, has its lifetime determined and communication protocols selected, and so on. However, while it is prescriptive on matters of basic behavior, OGSA does not place requirements on what a service does or how it performs that service. OGSA does not address issues such as the implementation programming model, programming language, implementation tools, or execution environment.

In practice, a specific execution or hosting environment instantiates Grid services. A hosting environment defines not only the implementation programming model, programming language, development tools, and debugging tools, but also how a Grid service implementation meets its obligations with respect to Grid service semantics.

Today's e-science Grid applications typically rely on *native operating system processes* as their hosting environment with, for example, creation of a new service instance involving the creation of a new process. Such an environment can implement a service in a variety of languages such as C, C++, Java, Fortran, or Python. Grid semantics may be implemented directly as part of the service, or provided via a library linked into the application. Typically, external services do not provide semantics beyond those the operating system provides. Thus, for example, lifetime management functions must be addressed within the application itself, if required.

OGSA specifies interactions between services independent of any hosting environment.

More sophisticated container- or component-based hosting environments such as J2EE, Websphere, .NET, and Sun ONE can implement Web services. Such environments define a framework within which to instantiate and compose components adhering to environment-defined interface standards for building complex applications. Compared with the low levels of functionality that native hosting environments provide, container and component hosting environments tend to offer superior programmability, manageability, flexibility, and safety. Consequently, these environments enjoy widespread use for building e-business services.

In the OGSA context, the container has primary responsibility for ensuring that the services it supports adhere to Grid service semantics and for offloading some service responsibilities from the service implementer. Thus, OGSA may motivate modifications or additions to the container or component interface.

By defining service semantics, OGSA specifies interactions between services independent of any hosting environment. However, specifying baseline characteristics that all hosting environments must possess—defining the internal interface from the service implementation to the global Grid environment—can facilitate successful implementation of Grid services. These characteristics would then be rendered into different implementation technologies such as J2EE, .NET, or shared libraries.

A hosting environment should address the following:

- mapping of Grid-wide names, or Grid service handles, into implementation-specific entities such as C pointers and Java object references;
- dispatch of Grid invocations and notification events into implementation-specific actions such as events and procedure calls;

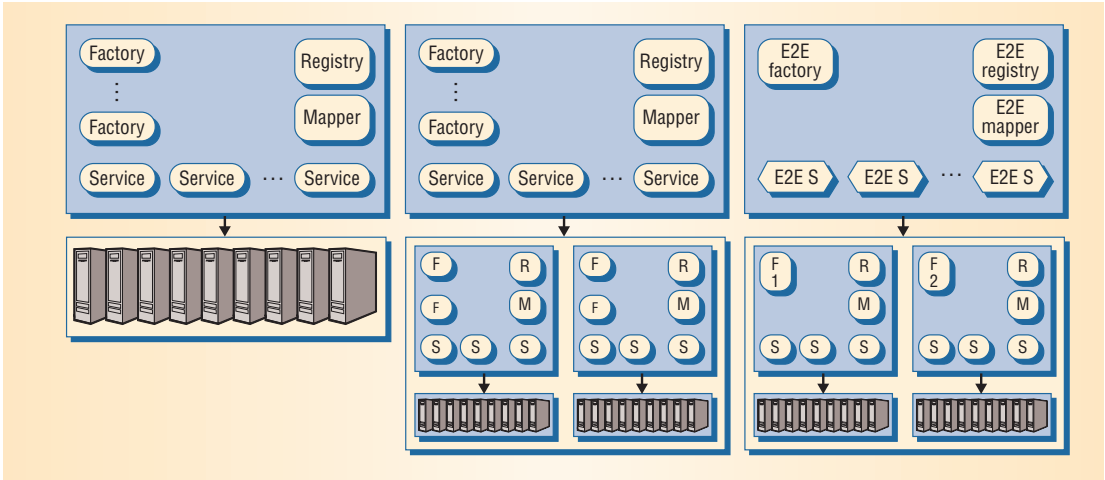

Figure 2. Three different VO structures, from left to right: a simple hosting environment, a virtual hosting environment, and collective services.

- protocol processing and data formatting for network transmission;
- lifetime management of Grid service instances; and
- interservice authentication.

An important consequence of OGSA's support for virtualization is that the user need not be aware of how a particular hosting environment implements OGSA interfaces and behaviors. Figure 2 illustrates this point, showing how a simple hosting environment, a virtual hosting environment, and collective services can implement the same interfaces.

Simple hosting environment. A simple execution environment provides a set of resources located within a single administrative domain that supports native facilities for service management, such as a J2EE application server, Microsoft .NET system, or Linux cluster. In OGSA, the user interface to such an environment will typically be structured as a registry, one or more factories, and a handleMapper for mapping from a globally unique Grid service handle to binding information. Each factory is recorded in the registry so that clients can discover available factories.

Virtual hosting environment. In more complex environments, the resources associated with a VO will span heterogeneous, geographically distributed *hosting environments*. For example, in Figure 2 these resources span two simple hosting environments. Nevertheless, this virtual hosting environment, which could, for example, correspond to the set of resources associated with a B2B partnership, can be made accessible to a client via exactly the same interfaces used for the simple hosting environment.

Collective services. We can also construct a virtual hosting environment that provides VO participants with more sophisticated virtual, collective, or end-to-end services. In this case, the registry tracks and advertises factories that create higher-level service instances. Such instances are implemented by asking lower-level factories to create multiple service instances and by composing the behaviors of those instances into a single, higher-level service instance.

EXAMPLE: A DATA MINING SERVICE

Figure 3 illustrates some aspects of OGSA's use and operation. This figure depicts a situation in which a user wants to discover, acquire, and employ remote capabilities to create a new database using data mined from a number of online databases. The figure illustrates the following steps:

1. The user—or, more likely, a program or service acting on the user's behalf—contacts a registry that a relevant VO maintains to identify service providers who can provide the required data mining and storage capabilities. The user request can specify requirements such as cost, location, or performance.
2. The registry returns handles identifying a miner factory and database factory maintained by service providers that meet user requirements—or perhaps a set of handles representing candidate services. In either case, the user identifies appropriate services.
3. The user issues requests to the miner and database factory specifying details such as the data mining operation to be performed, the form of the database to be created to hold results, and initial lifetimes for the two new service instances.

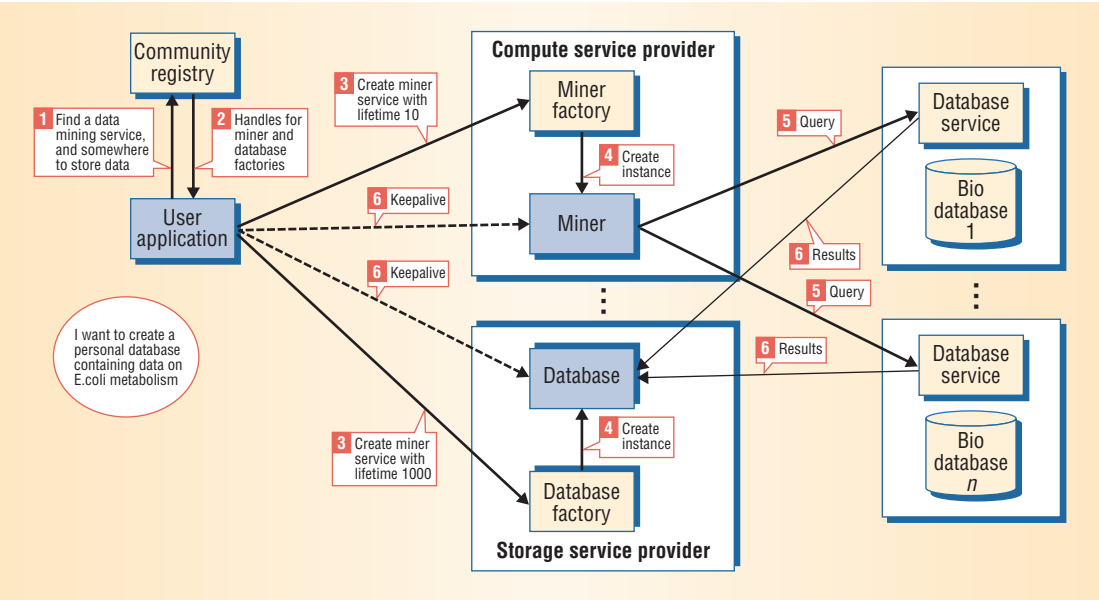

Figure 3. OGSA data mining example. The user contacts a registry, which returns handles identifying miner and database factories. User requests to those factories result in the creation of a miner service, which initiates queries against remote databases—in effect, now acting as a client on the user's behalf—and a database service that stores the mining results.

4. Assuming that this negotiation process proceeds satisfactorily, two new service instances are created with appropriate initial state, resources, and lifetimes.
5. The miner service initiates queries against appropriate remote databases, in effect now acting as a client on the user's behalf as it engages in further remote operations. OGSA security mechanisms address the delegation issues that arise in this context.
6. Results are returned from queries, either to the miner or, as Figure 3 shows, directly to the newly created database. Meanwhile, depending on the initially negotiated lifetimes, the user may have started to issue periodic *keepalive* messages to indicate continued interest.

A successful outcome of this process is that the miner service instance notifies the user of completion and terminates. The user then retains a handle for the newly created database, which it can retain and use for as long as it wants or can afford, with periodic keepalives signaling continued interest. On the other hand, the failure of the user's computation will result in eventual reclamation of service provider resources, due to lifetime expiry.

OGSA's service orientation means that it can implement the components shown in Figure 2 in various ways. For example, the registry could be a distributed service that integrates information from several sources, while the "database factory" could be a broker that negotiates with a variety of service providers to identify resources that meet user requirements. From the user's perspective, these implementation details are visible only to the extent that they affect delivered performance.

Grid concepts and technologies are transitioning from scientific collaborations to industry. We believe that the Open Grid Services Architecture will help accelerate that transition by recasting the Grid technologies that the Globus Toolkit provides in a uniform service-oriented architecture and integrating those technologies with emerging Web services standards. OGSA thus represents a natural evolution of both Grid technologies and Web services.

By integrating support for transient, stateful service instances with existing Web services technologies, OGSA extends the power of the Web services framework significantly, while requiring only minor extensions to existing technologies. OGSA facilitates the realization of Grid concepts in practical settings by adopting an industry-standard interface definition language and enabling the use of Web services tooling.

OGSA abstractions and services provide building blocks that developers can use to implement a variety of higher-level Grid services, for example for data and resource management.[11,12] We are working within the Global Grid Forum with both industry and the academic and open source communities to define a variety of such services that will, collectively, address the diverse requirements of e-business and e-science applications. ■

Acknowledgments

We thank Karl Czajkowski, Jeffrey Frey, Steve Graham, Thomas Sandholm, Jarek Gawor, John Bresnahan, Ravi Madduri, and Peter Lane for their many contributions to the Open Grid Services Architecture. We also thank the many colleagues who provided helpful comments on versions of this paper, particularly Malcolm Atkinson, Brian Carpenter, Andrew Grimshaw, Keith Jackson, Bill Johnston, Kate Keahey, Gregor von Laszewski, Miron Livny, Norman Paton, Jean-Pierre Prost, and Von Welch. This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, US Department of Energy, under Contract W-31-109-Eng-38; by the National Science Foundation; by the NASA Information Power Grid program; and by IBM.

References

1. I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, 1999.
2. W.E. Johnston, D. Gannon, and B. Nitzberg, "Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid," *Proc. 8th Int'l Symp. High-Performance Distributed Computing* (HPDC8); http://www.computer.org/proceedings/hpdc/0287/02870034abs.htm (current June 2002).
3. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int'l J. High-Performance Computing Applications*, vol. 15, no. 3, 2001, pp. 200-222; http://www.globus.org/research/papers/anatomy.pdf (current June 2002).
4. I. Foster et al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," tech. report, Glous Project; http://www.globus.org/research/papers/ogsa.pdf (current June 2002).
5. S. Graham et al., *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*, Sams Technical Publishing, Indianapolis, Ind., 2001.
6. E. Christensen et al., "Web Services Description Language (WSDL) 1.1," W3C Note, 15 Mar. 2001; http://www.w3.org/TR/wsdl (current June 2002).
7. C. Catlett and L. Smarr, "Metacomputing," *Comm. ACM*, June 1992, pp. 44-52.
8. M. Shapiro, "SOS: An Object-Oriented Operating System—Assessment and Perspectives," *Computing Systems*, vol. 2, no. 4, 1989, pp. 287-337.
9. A.S. Grimshaw and W.A. Wulf, "The Legion Vision of a Worldwide Virtual Computer," *Comm. ACM*, vol. 40, no. 1, 1997, pp. 39-45.
10. S Raman and S. McCanne, "A Model, Analysis, and Protocol Framework for Soft State-Based Communication," *Computer Communication Rev.*, vol. 29, no. 4, 1999, pp. 15-25.
11. N.W. Paton et al., "Database Access and Integration Services on the Grid," tech. report UKeS-2002-3, National e-Science Centre; http://www.nesc.ac.uk, 2002.
12. M. Livny, "High-Throughput Resource Management," *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, 1999.

Ian Foster *is a senior scientist and associate director of the Mathematics and Computer Science Division at Argonne National Laboratory, a professor of computer science at the University of Chicago and a Senior Fellow in the Computation Institute at the University of Chicago. His research interests include distributed and collaborative computing and computational science. Foster received a PhD in computer science from Imperial College, London. He is a member of the ACM and the American Association for the Advancement of Science. Contact him at foster@mcs.anl.gov.*

Carl Kesselman *is a senior project leader at the University of Southern California's Information Sciences Institute. His research interests include distributed computing and networking. Kesselman received a PhD in computer science from the University of California at Los Angeles. He is a member of the IEEE. Contact him at carl@isi.edu.*

Steven Tuecke *is a software architect in the Distributed Systems Laboratory in the Mathematics and Computer Science Division at Argonne National Laboratory and a Fellow in the Computation Institute at the University of Chicago. Tuecke is also the codirector of the Global Grid Forum Security area. He received a BA in mathematics and computer science from St. Olaf College. Contact him at tuecke@mcs.anl.gov.*

Jeffrey M. Nick*, an IBM Fellow and Director of Advanced Systems Architecture working in Poughkeepsie, New York, is chief architect for IBM's Project eLiza and Grid computing initiative. He is a member of the IBM Academy of Technology. Nick received a BS in computer science from Marist College. Contact him at jnick@us.ibm.com.*