Grid Services Extend Web Services

Andrew Grimshaw
Avaki Corporation
Burlington, MA
and
The University of Virginia
Charlottesville, VA

Steven Tuecke
Globus Project
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL

1 Background

We are often asked by people who are trying to understand the value Grid technology brings to Web services, "what is the significance of Grid services? They look like Web services." The answer to this question is superficially straightforward: mechanically, the differences between Grid services (as defined in the Grid service specification [14]) and Web services are few: to first order, a Grid service is simply a Web service that conforms to a particular set of conventions. For example, Grid services are defined in terms of standard WSDL (Web Services Definition Language) with minor extensions, and exploit standard Web service binding technologies such as SOAP (Simple Object Access Protocol) and WS-Security (Web Services Security). So yes, Grid services *do* look like Web services.

So what is the significance of Grid services? Well, these Grid service conventions not superficial in their function: they address fundamental issues in distributed computing, relating to how to name, create, discover, monitor, and manage the lifetime of stateful services. More specifically, these conventions support:

- named service instances and a two-level naming scheme that facilitates traditional distributed system transparencies (see sidebar);
- a minimum set of service capabilities, including rich discovery (reflection) facilities; and
- explicitly stateful services with lifetime management.

Before proceeding further, let us state that clearly it is possible using standard Web services to manage and name stateful services using *ad hoc* methods, e.g., extra characters placed in URLs or extra arguments to functions. Similarly one can define an introspection service interface and publish it via WSDL. However, the point is not whether it can be done, but whether it is done in a uniform and consistent manner that everyone understands so that the full power of traditional distributed system naming and binding techniques can be brought to bear on the widest possible set of services. Grid services are important because they provide uniformity and consistency for vital distributed system functions.

2 Named service instances

At the heart of the Grid service specification is the notion of a named service instance. A service instance is named by a Grid service handle (GSH). The GSH is a classic abstract name—it is not necessarily possible to determine by examination of a GSH such things as the location, number, implementation, operational status (e.g., up, down, failed), and failure characteristics of the associated named object. GSHs by themselves are not useful. Instead they must be resolved into a Grid service reference (GSR). A GSR contains sufficient information to communicate with the named grid service instance. Thus, GSHs and GSRs form a two-level naming scheme in which GSHs serve as abstract names and GSRs provide the method and address of delivery. Before we describe the benefits of multi-layer name schemes let us first look at the benefits of having abstract names.

2.1 Why abstract names?

What is an abstract name? An abstract name is refers to a “thing” or “things.” In the case of Grid services, the “things” are *Grid service instances*, which may be stateless, i.e., pure functions that transform or map inputs to outputs in which the outputs are independent of the history of calls on the service instance; or stateful, in which case the output may depend on the history of calls on the service. The advantage of using abstract names rather than addresses is that it permits a level of indirection between the representation of the name and the code that actually implements the service.

Note that the form of the GSH is flexible—syntactically, a GSH is represented as a URI, allowing for many different name schemes. This allows for great flexibility in choosing how GSHs are resolved into GSRs. For example, one could choose to implement a GSH name space that uses DNS addresses as part of the GSH. However, such a scheme is not required, nor does the inclusion of a DNS address in a GSH tell you anything *per se* about the location of the named service instance.

2.2 Why use a two-level name scheme instead of addresses?

Recall that GSHs (names) resolve to GSRs (addresses). Because GSHs are abstract names, the mapping of GSHs to GSRs need not be static – it is possible that a given GSH may resolve to different GSRs over time. For example, a Grid service instance (named by a GSH) may migrate from location to location in response to load, anticipated failure, or to improve performance by co-locating with a client or frequently used service.

Furthermore, the mapping may not only vary over time but need not be one-to-one. Other mappings are possible: one-to-many where the same GSH resolves to different GSRs—e.g., for load balancing.

The point is that by utilizing a two-level naming scheme, Grid services enable flexible realization of the traditional distributed system transparencies described in the sidebar. This has been done in many distributed systems in the past [8][9][10][11][12][13], as well as in contemporary Grid systems such as Legion [6][7] and Avaki [1].

3 Minimum Set of Services

The second important aspect of Grid services is that a Grid service must export (expose) a minimum set of mandatory interfaces and a minimum set of what are called *service data elements* (SDEs)—essentially service meta-data and state. These mandatory services and data elements are interfaces that all GSs can be *assumed* to have. They include functions and data elements for life-time management (e.g., set termination time) and functions to query and manipulate SDEs.

The importance of SDEs in the Grid cannot be underestimated. They provide the basic mechanism by which discovery and monitoring of Grid services is achieved, including interface discovery, discovery of other meta-data that might be used in services such as scheduling (e.g., policy information), and even the current state of the service instance (e.g., current load).

SDEs that convey dynamic state merit particular attention, for they truly distinguish Grid services from what is typically found in Web services. Whereas Web service technologies such as WSDL and UDDI (Universal Description, Discovery, and Integration) allow for introspection and discovery of static information such as service interfaces and associated policy, dynamic SDEs provide a much richer basis for discovery and monitoring. This dynamicity is crucial when building dynamic systems in which the set of service interactions may be impossible to know *a priori* (at configuration time), in which decisions about which services to use may depend heavily on dynamic service state, and in which monitoring of a service's dynamic state is necessary to respond to unpredictable events in the overall system. Service data exploits XML strengths, including XML schemas for rich modeling of the state and extensible support for rich query languages such as XPath and XQuery for introspection on that modeled state, to provide for a consistent and powerful basis on which discovery and monitoring can be built across all Grid services.

By clearly defining a minimum mandatory set of interfaces that all Grid services must support, as well as optional interfaces that play core roles, Grid service specification makes it possible to write higher-level software services and applications that can make assumptions about both the services and the infrastructure of the Grid. This uniformity vastly simplifies the definition and implementation of the various higher-level services that must be defined to construct a complete Open Grid Services Architecture (OGSA). In addition, by defining base service type interfaces, the Grid service specification sets the tone and philosophy of both how the Grid will operate and how future extensions will be shaped.

4 Instantiation and Lifetime Management

The final major extension is the explicit concept of state in Grid services as described earlier in sections 2 and 3 when discussing names, service data elements, etc. While Web services may have state, it is not explicit, nor are different discrete units of state (i.e., instances) named in a consistent manner. Once there is namable state the obvious issues are (1) how are new instances created?, and (2) how long do they last?

To address these two issues the Grid service specification defines *factories* and *lifetime management* services.

Factories. New web services are instantiated in the case of Grid services by factories. The Factory *PortType* defines an operation that creates a new service instance and returns its GSH (handle) and its initial termination time.

TerminationTime is a powerful concept supported by the Grid service specification. A significant problem in distributed systems in general, and Grid systems in particular, is the issue of reclamation of resources associated with services in the event of failures (e.g., loss of network connectivity) or lack of interest by any relevant clients (e.g., a service is no longer referenced by any other active process or service).

TerminationTime addresses this problem by setting a time when a service will self-destruct unless kept alive by subsequent increases in its termination time. Thus, a service can be started, and the progenitor need not explicitly destroy the service. Instead it will automatically be terminated.

5 Summary

The Grid service specification provides the foundation upon which a wide range of Grid services will be defined, built, and interconnected. Grid services marry important concepts from the Grid computing community with Web services. Grid services extend basic Web services by defining a two-layer naming scheme that enables support for the conventional distributed system transparencies, by requiring a minimum set of functions and data elements that support discovery, and by introducing explicit service creation and lifetime management.

These extensions are more than syntactic sugar. They extend and enhance the capabilities of Web services by providing common, powerful mechanisms that service consumers can rely upon across all services, instead of the service-specific, often ad-hoc approaches typically employed in standard Web services.

Acknowledgements

We'd like to thank Marty Humphrey (University of Virginia), David Snelling (Fujitsu), and the OGSI Working Group for their contribution to establishing open standards for Grid technology.

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

A Sidebar on Transparencies

Distributed systems research has a rich literature. In this literature, virtualization of resources and objects is a common solution to many problems. This virtualization results in a "transparency." Nine of these show up again and again. They are used with respect to accessing a remote service or object. Usually the intent is that the programmer/user need not know or deal with something, but can if they want to. The transparencies are:

Access. The mechanism used for a local procedure call is the same as for a remote call. Many have argued this is a special case on location transparency.

Location. The caller need not know where the object is located, California or Virginia - it makes no difference.

Failure. If the object fails the caller is unaware. Somehow the requested service or function is performed, the object is restarted, or whatever is needed happens.

Heterogeneity. Architecture and OS boundaries are invisible. However, objects cannot migrate without limitation. At a bare minimum communication with objects on other architectures requires no data coercion.

Migration. The caller need not know whether an object has moved since they last communicated with it.

Replication. Is there one object or many objects behind the name? The caller need not know or deal with coherence issues.

Concurrency. Are there other concurrent users of an object? The caller need not be aware of them.

Scaling. An increase or decrease in the number of servers requires no change in the software. Naturally performance may vary.

Behavioral. Is it live or is it Memorex? Whether an actual object or a simulation of the object is used is irrelevant. For example, am I talking to a host object, or a virtual host object?

References

[1] Avaki Corporation, www.avaki.com.

[2] H. Bal, J. Steiner, and A. Tanenbaum, “Programming Languages for Distributed Computing Systems,” *ACM Computing Surveys*, pp. 261-322, vol. 21, no. 3, Sept. 1989.

[3] R. Chin and S. Chanson, “Distributed Object-Based Programming Systems,” *ACM Computing Surveys*, pp. 91-127, vol. 23, no. 1, March., 1991.

[4] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration,” Open Grid Services Infrastructure WG, Global Grid Forum, June 22, 2002.

[5] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "Grid Services for Distributed System Integration,” Computer, 35(6), 2002.

[6] A.S. Grimshaw, "Enterprise-Wide Computing," *Science*, 256: 892-894, Aug 12, 1994.

[7] A. S. Grimshaw and W. A. Wulf, "The Legion Vision of a Worldwide Virtual Computer," *Communications of the ACM*, pp. 39-45, vol. 40, number 1, January, 1997

[8] S. Mullender ed., *Distributed Systems*, ACM Press, 1989.

[9] *Operating Systems: An Advanced Course*, R. Bayer, R. M. Graham, and G. Seegmuller Eds., Springer-Verlag, New York, Berlin, Heidelberg, Tokyo, 1979.

[10] D. Notkin, N., et al., “Heterogeneous Computing Environments: Report on the ACM SIGOPS Workshop on Accommodating Heterogeneity,” *Communications of the ACM*,vol. 30, no. 2, pp. 132-140, February, 1987.

[11] D. Notkin, et al., “Interconnecting Heterogeneous Computer Systems,” *Communications of the ACM*, vol. 31, no. 3, pp. 258-273, March, 1988.

[12] J. Stankovic, K. Ramamritham, and W. H. Kohler, “A Review of Current Research and Critical Issues in Distributed System Software,” *IEEE Distributed Processing Technical Committee Newsletter*, pp. 14-47, vol. 7, no. 1, March, 1985.

[13] A. S. Tanenbaum and R. van Renesse, “Distributed Operating Systems,” *ACM Computing Surveys*, pp. 419-470, vol. 17, no. 4, December, 1985.

[14] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman," Grid Service Specification," Open Grid Services Infrastructure WG, Global Grid Forum, Draft 3, July 17, 2002.