B. Long, University of Westminster
V. Getov, University of Westminster
June 6, 2003

A Generic Model for the OGSA Platform

Status of This Memo

This memo provides information to the Grid community interested in OGSA platforms. It does not define any standards or technical recommendations. Distribution is unlimited.

Abstract

A simple architecture for an OGSA Platform model is described that provides a single system image (SSI) for Grid users, compilers, applications and services over a heterogeneous environment without excessively sacrificing flexibility.

Contents

1. Introduction

The draft specification for an Open Grid Services Architecture (OGSA)[7] platform specifies that Grid services are built and managed according to the Open Grid Services Infrastructure (OGSI)[4] specification, and it describes a number of platform interfaces which applications may use to find and manage data, policies, security information, etc. Lastly, it is proposed in the OGSA Platform draft that XSD[14] models of common resources be used to standardize the use and management of common resources.

The draft specification does not express how, on an arbitrary host, in a heterogeneous environment, tasks can be run, binaries that access local resources can be created and executed, and how a user interface can be accessed, perhaps on a non-local host. Some of this functionality is implemented in the Globus Toolkit[6, 16]. This document outlines a solution in which an abstract modeling language is used as an intermediate language to respond to changes in an environment by choosing from among a variety of solutions such as building a specialized binary to take advantage of new hardware or working with a different user interface paradigm. The described solution provides a single system image (SSI)[15] for both service and application binaries by having languages compile to the intermediate language.

2. Background

Quanta is a light-weight core language for describing objects and classes, and systems of objects and classes; an engine can then be used to make inferences about the objects and classes in order to instantiate objects or query and manipulate them. By extending the engine's namespace to include other namespaces or outside objects, Quanta and the engine can be used to manipulate and query such systems as easily as internal objects. A variety of methods for accomplishing a task can be represented allowing the engine to determine an optimal method. Lastly, a Quanta engine can be made to translate sequences, conditionals and repetitions into code providing the ability to produce a C file or a binary specialized for a particular environment. It can be used as an intermediate language by compiling, for example, C++ or Java byte code to Quanta. The Quanta language and engine are described in [12].

3. Environment Binding

Since many of the resources and services upon which an application or service relies during execution vary in format from environment to environment, a generic platform must be able to identify any environment-specific functionality and make an appropriate substitution of identical functionality at build time, load time, or even during execution. To accommodate such specialization, we mention three types of binding common to most environments.

- **Binding to local resources.** High speed local resources such as graphics hardware, FireWire ports, or cards on the local bus can be utilized more efficiently if the binding to them is not through a Grid Service on the local host.

- **Binding to build and run binaries or perform calculations.** The binaries used to perform tasks may vary over environments as well as over the

method of executing the task. On a low level, binaries must be built per environment to bind to the local operating system or processor type. For example, an operation to process a block of data may be implemented as a loop on one machine but a parallel operation on another.

- **Binding to a user interface platform.** Applications or services that must interface with users on a heterogeneous system must be bound to a local user interface platform such as the Windows GUI, Java Swing, or a Web-based UI. If the user is not at the local machine, as is common with distributed gaming, programs may be bound to a user interface proxy.

3.1   Binding via identity substitution

The method being described to bind programs to an environment is to represent situations that may vary over environments and, at the appropriate time, substitute an environment-specific situation that provides equivalent functionality. In the simplest case, such a situation may be of a call to a function or an assignment operation. More complex binding operations, such as binding to custom processing hardware or specializing the creation of a binary, may require substitutions through sequences, conditionals, or repetitions. With the addition of nested substitutions, the creation of complex compositions of services, local resources, and user interface components may be automated.

3.2   The Quanta Language

Quanta models classes, objects, and systems of objects in such a way that the effects of using a modeled system can be inferred. Objects classified can be digital or analog, physical, or abstract. A *Quanta Engine* can use the models to infer what use of resources on the local system or on a network can be substituted to meet given binding requirements. The engine then makes the required substitutions. Because the engine can make substitutions involving the von Neumann structures of sequences, conditionals, and repetitions of information manipulations, entire algorithms can be reworked as needed to take advantage of new resources and accomplish a task or instantiate an object. The following are some of the major components of the Quanta language:

- Numeric, string, and Boolean literals
- Names / functions
- Identity assertions
- Informatic membership, union, difference, intersection and complement
- Block, repetition, and conditional constructions
- A connection to the local system, e.g., ability to make certain system calls.

Using identity assertions, any construction of the above components can be hierarchically mapped to any other one. For example, a function can be mapped to an algorithm or to a system call, while a repetition or sequence can be mapped to the results of running a parallel processor using a given algorithm.

**4.   A Generic OGSA Integrated Platform Model**

The generic Grid platform model being described has three aspects: namespace management, the execution environment, and binary-to-environment binding. While the platform is executing on a local host platform, it is running in the context of one or more virtual organizations (VO)[5]. In addition to any VOs with which the platform engine associates due to membership or affiliations of the owner, a global VO will exist and operate in much the same way that P2P services such as Kazaa operate, yet without the purpose of mass file sharing.

4.1   Namespace Management

The platform engine running on its local host will maintain a hierarchical namespace. The local namespace will include references and Quanta descriptions of such items as the following:

- Local hardware available
- Local file systems and objects/services
- References to local users, their local settings and objects
- Local settings

In addition, each VO of which the system is a member will provide resources to the namespace. In particular, the global Grid VO will provide a global namespace managed by the peer-to-peer cooperation of all the systems in the VO. By storing files and other objects in their global grid namespace folder, users and VOs will be able to access and manage their information globally, even if their own machines are currently down.
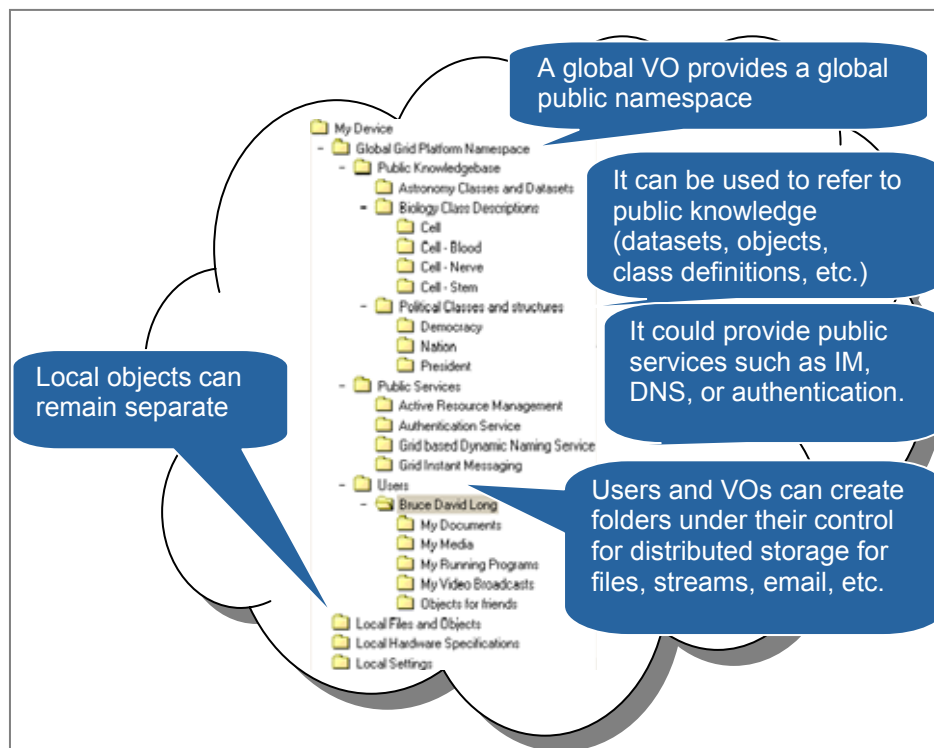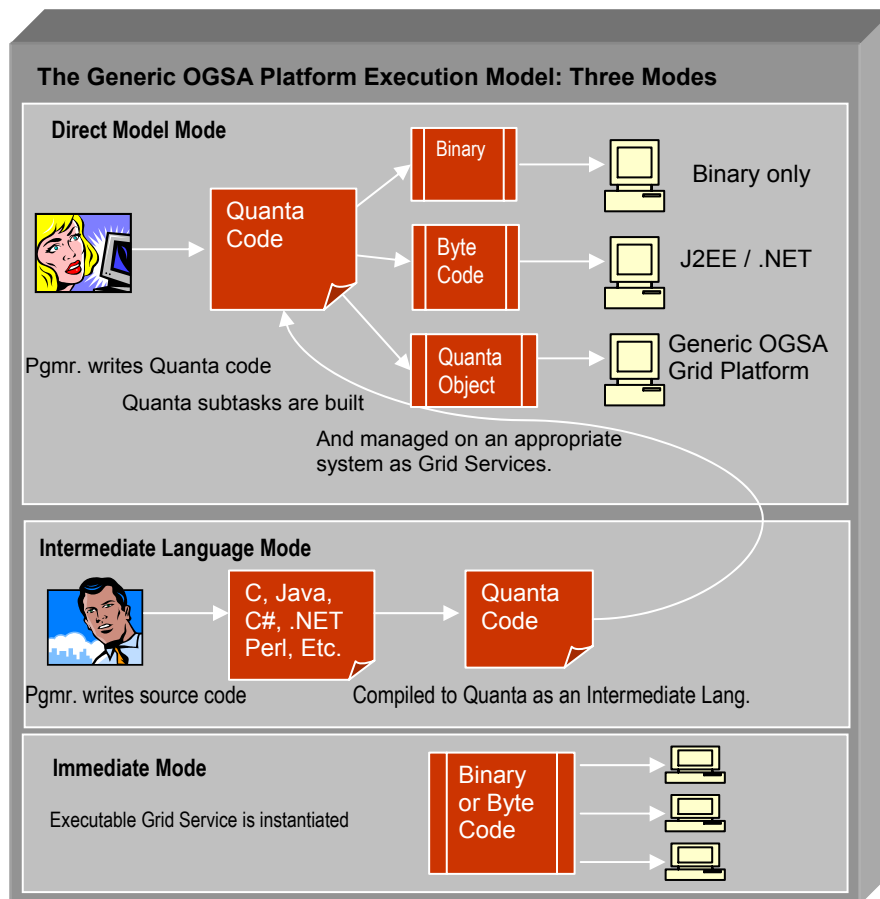

Other entities referenced in the namespace are running processes and version information through Quanta descriptions.

4.2   The Execution Environment

A process begins executing when it is created in the namespace and its "executing" property is set to true. Other properties express whether it is running on the local machine, a remote machine, or on multiple machines from one or more VOs.

To support the widest variety of situations and preferences, there are three executions modes under this proposed generic platform model. In the *Direct Model mode*, the algorithms to be executed or the queries to be run are specified as Quanta models. The Quanta engine decides how best to execute them, whether to do so itself, build specialized binaries and execute them on remote machines, or execute pre-specified processes or Java code[8] on remote machines via GRAM[9, 13]. Perhaps a particular task will be divided into parts and executed on different machines by all three methods. In *Intermediate mode*, programs written in traditional languages such as C++, Perl, or .Net CLR can be compiled to Quanta as an intermediate language. The Quanta code can then be executed in Direct Model mode. Lastly, in *Immediate mode*, binaries and Java code built by other means can be executed directly using GRAM.


The Generic OGSA Platform Execution Model: Three Modes

Direct Model Mode

Quanta Code

Binary → Binary only

Byte Code → J2EE / .NET

Quanta Object → Generic OGSA Grid Platform

Pgmr. writes Quanta code

Quanta subtasks are built

And managed on an appropriate system as Grid Services.

Intermediate Language Mode

C, Java, C#, .NET Perl, Etc. → Quanta Code

Pgmr. writes source code   Compiled to Quanta as an Intermediate Lang.

Immediate Mode

Executable Grid Service is instantiated

Binary or Byte Code

4.3   Binding: Specialized binaries for each machine and user interface type

The heterogeneous nature of Grid computing requires generalizing over both hardware and user interface types. By using an intermediate language which can model hardware, user interface types, and applications, the Generic platform engine

may build specialized binaries for each new situation. There need not be a "Grid user interface"; rather, each situation can be mapped to the location, available UI platforms, and preferences of the user.

A Grid Services "Meta-Platform"
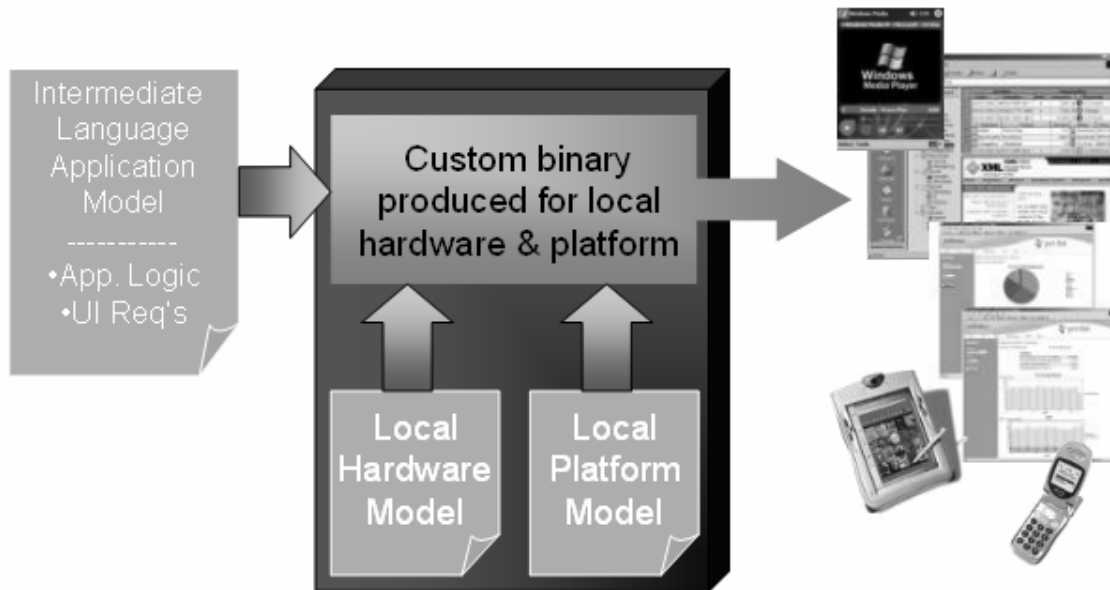
4.4   Architecture and the user experience

In addition to the standard OGSA services, the generic platform running on a machine offers a number of services to both users and other such running platforms if they are in a shared VO. The system will retain policies about which users are offered which services, as well as which machines or VOs can request them. When a user is identified at one of the connection points the platform is monitoring, perhaps at the local machine or by a web interface, services will be offered and delivered by finding or building a binary that instantiates the requested application with the user interface required. Some aspects of the task may be done through interpretation rather than building a binary.

By offering and expecting certain services from peers, a VO or a union of VOs will be made to have a single system image. This SSI can carry out tasks or offer services independent of any particular machine. Thus, once a user starts an application, it may not be apparent which machine or machines are actually carrying out the computations or storing the information. Because messages from users are marked as originating from a user in the namespace, not from a machine, the users can switch machines in the middle of an application and have the UI follow them and even adjust for heterogeneity.

The decentralized nature of such a process, together with the existence of a global VO, makes possible applications such as Grid-based DNS, public authentication, or decentralized IM to run outside the context of a hosting organization. It also makes a high degree of cooperation possible; for example, while a cell phone may be able to broadcast video over TCP/IP to one or two viewers, if it were broadcast to a folder in the global VO (perhaps the folder represents a URI), the VO could ensure that the

broadcast was replicated at strategic points and could serve the video at almost any scale.

5. Examples of models for various host environments

The models that populate the system's namespace are held and communicated in Quanta documents. Based on their use and lifetime, four categories of documents can be identified. Below are examples of documents that might exist in a finished product; they are in no way prescriptive and may be altered considerably in the final platform.

Library documents are static and available on all platform hosts. They include models of mathematical classes, identities and functions. Other library documents may describe typical computer hardware, data structures and algorithms. Objects such as queues, strings and streams, as well as memory, network nodes and common CPUs may be described. Also, documents may describe Grid concepts such as the OGSA protocols, OGSI and other API's and utilities. For example, a Quanta document might map names and functionality to LDAP[10] or UDDI[17] documents to facilitate the use of such services. Those document maps would be built on a lower Quanta document mapping the names implied by XML[14], XSL[14] and intermediates such as SOAP[1].

Local system documents give the engine enough information to identify local users, use local hardware, understand policies, offer services, and log on to and participate in VOs including the global VO. Such documents might describe local hardware, local users, and have descriptions of services to be offered, and to whom they should be offered.

Host environment documents may describe particular types of hosting environment such as MS Windows, Java or SUN environments. Documents detailing how to compile and build binaries or C programs are an example, as well as one detailing the execution environment; how to run and interact with processes. Also, any local user interface types can be detailed.

Lastly, a number of Quanta and WSDL[2] documents will be exchanged when an engine connects to any VOs, including the global VO. Such documents might include models expressing network topology, authentication information, collective namespace negotiation, or the locations of GIIS [3] servers.

6. Conclusions

The problem of performing an action in a heterogeneous environment must be solved by identifying identicals, whether code, objects, functions or otherwise, and making a substitution that is compatible with the host environment. This substitution can be done by the programmer with conditionals, or at build-time, load-time or runtime. Since the logic of substituting identicals is the same whatever the problem or time, rather than creating a special tool for each situation, Quanta can be used to specify which situations can be substituted for and what substitutions are valid. The Quanta engine, when coded to be an OGSA platform, can be made to find optimum substitutions and make them, whether at code-time, run-time, or in-between, in the context of the Global Grid. Such a system would facilitate the use of new and legacy code in many languages, provide an extensible, global namespace, and change the

perspective for Grid-based applications and services from that of "running on a local host while accessing distributed resources" to "running on the Grid."

7. Security Considerations

As with any platform, security is a primary concern. With a Quanta based OGSA platform the greatest security threat is that a rogue Quanta document with false information may be introduced to the engine. For example, one can imagine a definition for a square-root function that actually causes harm to local data. Such a threat is analogous to introducing harmful binary code to a Windows operating system in such a way that it executes. Unlike binary code, however, Quanta documents contain names for every entity they directly access or modify. One step in establishing trust of a document is to scan for what objects the document refers to and verify that it abides by policies defined, for example, by black lists, or white lists.

In addition to establishing the trustworthiness of documents, care should be taken to ensure that any platform engines use secure protocols such as HTTPS and utilize security mechanisms such as Kerberos or PKI. Work should proceed with the results of the OGSI security working group under consideration.

Author Information

Bruce Long
School of Computer Science
University of Westminster
Watford Rd, Northwick Park
Harrow, London HA1 3TP, U.K.
B.D.Long@westminster.ac.uk

Vladimir Getov
School of Computer Science
University of Westminster
Watford Rd, Northwick Park
Harrow, London HA1 3TP, U.K.
V.S.Getov@westminster.ac.uk

Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director.

References

1. D. Box, et al, "Simple Object Access Protocol (SOAP)", W3C, http://www.w3.org/TR/SOAP/, (May 8, 2000).
2. E, Christensen, "Web Services Description Language (WSDL)", W3C, (March 15, 2001).
3. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing," Proc. 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, http://www.globus.org/research/papers/MDS-HPDC.pdf (2001)
4. K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, D. Snelling, S. Tuecke, and P. Vanderbilt, "Open Grid Services Infrastructure (OGSI)," Open Grid Service Infrastructure WG, Global Grid Forum, http://www.ggf.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-26_2003-03-13.pdf (March 13, 2003).
5. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," Intl J. Supercomputer Applications, vol. 15, no. 3, http://www.globus.org/research/papers/anatomy.pdf (2001).
6. I. Foster and C. Kesselman, "The Globus Toolkit," The Grid: Blueprint for a New Computing Infrastructure, I. Foster and C. Kesselman, ed., Morgan Kaufmann Publishers, San Francisco, California, 1999, pp. 259-278.
7. I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Open Grid Service Infrastructure WG, Global Grid Forum, http://www.globus.org/research/papers/ogsa.pdf, (June 22, 2002).
8. V. Getov, G. von Laszewski, M. Philippsen, I. Foster. "Multi-Paradigm Communications in Java for Grid Computing," Communications of the ACM, vol. 44, no. 10, 118-125, (October 2001).
9. The Globus Project, "GRAM: Grid Resource Allocation and Management," http://www.globus.org/about/events/US_tutorial/slides/Dev-06-ResourceManagement1.pdf (2002).
10. J. Hodges, "An LDAP Roadmap & FAQ," http://www.kingsmountain.com/ldapRoadmap.shtml, (December 6, 2001)
11. K. Kennedy, "Compilers, Languages, and Libraries," The Grid: Blueprint for a New Computing Infrastructure, I. Foster and C. Kesselman, ed., Morgan Kaufmann Publishers, San Francisco, California, 1999, pp. 181.-204
12. B. Long, "Quanta: a Language for Modeling and Manipulating Information Structures," http://perun.hscs.wmin.ac.uk/pages/bruce/ (December 2002).
13. S. Martin, "GT3 GRAM Overview," http://wwwunix.globus.org/ogsa/docs/alpha/gram/ gt3_gram_overview.htm (January 8, 2003).

14. L. Quin, "Extensible Markup Language (XML), W3C," http://www.w3.org/XML/, (February 26, 2003).
15. B. Rajkumar, T. Cortes, and H. Jin, "Single System Image (SSI)," The International Journal of High Performance Computing Applications, vol. 15, no. 2, summer 2001, pp. 124-135.
16. The Globus Project, "Status and Plans for Globus Toolkit 3.0," http://www.globus.org/toolkit/gt3-factsheet.html (February 19, 2003).
17. UDDI.org, Universal Description, Discovery and Integration, "UDDI Technical White Paper," http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf (September 6, 2000).