A Globus Toolkit Primer

Or, Everything You Wanted to Know about Globus, but Were Afraid To Ask

Describing Globus Toolkit Version 4

An Early and Incomplete Draft

Please send comments, criticisms, and suggestions to: foster@mcs.anl.gov

Preface

The Globus Toolkit (GT) has been developed since the late 1990s to support the development of service-oriented distributed computing applications and infrastructures. Core GT components address basic issues relating to security, resource access and management, data movement and management, resource discovery, and so forth. A broader "Globus universe" comprises numerous tools and components that build on core GT4 functionality to provide many useful application-level functions. These tools have been used to develop many Grid systems and applications.

Version 4 of the Globus Toolkit, GT4, released in 2005, represents a significant advance relative to the GT3 implementation of Web services functionality in terms of the range of components provided, functionality, standards conformance, usability, and quality of documentation. This document provides an introduction to key features of both GT4 and associated tools, and the ways in which these components can be used to develop Grid infrastructures and applications. Its focus is on the user's view of the technology and its application, and the practical techniques that should be employed to develop GT4-based applications.

We discuss in turn the applications that motivate the development of GT4 and related tools; the four tasks involved in building Grids: design, deployment, application, operations; GT4 structure, including its Web services (WS) and pre-WS components; the Globus universe and its various components; GT4 performance, scalability, and functionality testing; porting to GT4 from earlier versions of GT; who's using GT4; likely future directions, and a few words of history.

Table of Contents

Index of Tables

Index of Figures

Chapter 1 Goals and Principles

This document is intended as an introduction to the Globus Toolkit and related tools, and to the use of those tools in building distributed system infrastructure and applications. Our intention is to present how Globus and related software are structured, and how that software can be used most effectively to build applications. To this end, we examine in turn the requirements that motivate Globus design, the various Globus Toolkit components, relevant technical specifications, related software that can be used in conjunction with Globus to address application requirements, and case studies that illustrate how this software has been and can be applied in practical settings. The result is a guide, not a complete reference: in most sections, we point the reader to other sources for further technical information. In particular, while we provide the technical concepts required to develop Globus applications, and describe command line programs, we do not provide any programming examples. Instead, we refer the reader to Borja Sotamayor's excellent tutorial [73] and to other documents on the Globus Web.

Our intention in this document is to speak to the practice, not the theory, of Globus software and its deployments and applications. We speak a little here in this introduction, and at somewhat greater length in a final "Future Directions" section, about the distributed computing vision that has motivated Globus Toolkit design. However, this document is about what Globus and Grid are today, not what they might be tomorrow.

1.1 Motivating Examples

We use six examples to illustrate the infrastructures and applications that have motivated the development of the technologies we describe here.

1.1.1 Enterprise Workload Management

SAP AG recently demonstrated three applications from its flagship R/3 product line that had been modified to use a Globus-based Grid. These demonstrations showed how an enterprise Grid can use Globus technologies to adapt to changing workload demands, reducing hardware requirements (or, alternatively, increasing achievable throughput: see Figure 1) relative to a configuration in which each application runs on dedicated hardware.

Internet Pricing and Configurator (IPC) and Workforce Management (WFM) form part of SAP's Customer Relationship Management (CRM) application suite, while the third, Advanced Planner and Optimizer (APO), is from their Supply Chain Management (SCM) suite. Each application is designed to support large numbers of requests generated by interactive clients using Web browsers or from batch processes, dispatching each request to one of a number of worker processes. In this work, SAP modified each of these applications so that it could (a) dynamically adjust the number of worker processes used to meet computational demands, and (b) use Globus components to discover and reserve the resources used to host those worker processes, and to execute, monitor, and remove the worker processes on those resources.

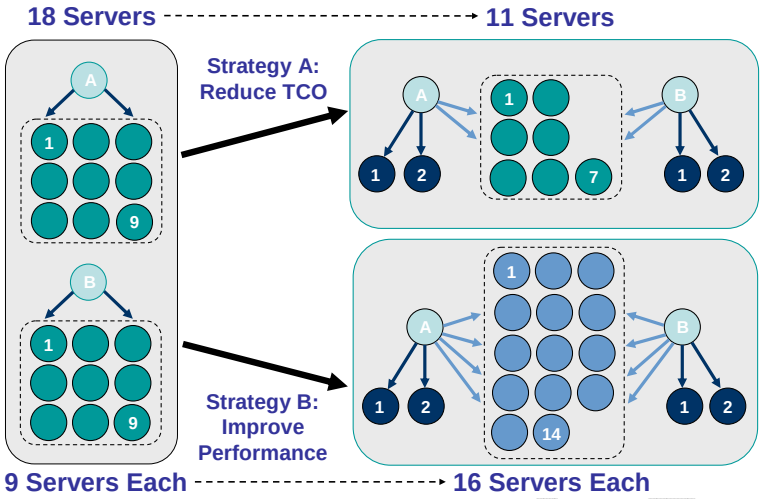

Figure 1: Grid technologies can be used to federate resource pools across two departments either to reduce total cost of ownership (Strategy A) or to increase available peak capacity (Strategy B).

1.1.2 High Performance Data Capture

A company faced the requirement to capture large amounts of data that are received at high sustained rates (5 to10 Gigabits per second). After accurate capture, the solution must perform initial processing of the data and extraction of metadata, reliably store resulting data, make that data available from the storage facility to a distributed community of clients at high rates (10 to 40 Gigabits per second aggregate), and provide the platform on which to perform analysis and data mining jobs against that data.

The platform constructed to address these requirements for a range of customers makes heavy use of Globus components, including GridFTP for reliable data capture and for data movement between data capture and data storage/distribution clusters, between storage/distribution clusters and secondary data systems such as archival and remote storage, and between storage/distribution clusters and clients; and Globus execution management components for job management. The benefit of Globus is that commodity hardware and software can be used. Previously, the company would build such systems using custom hardware and software, resulting in systems that were expensive, difficult to support, inflexible to changing demands, and requiring substantial elapsed time for completion.

1.1.3 Network for Earthquake Engineering Simulation

The U.S. National Science Foundation's Network for Earthquake Engineering Simulation (NEES) links earthquake engineers across the U.S. with each other and with a variety of different resource types, including data repositories, computers used for numerical simulations, and above all experimental facilities such as shake tables and wave tanks [70, 74]. The NEES software builds on Globus security, data access, compute access, and service implementation components to enable secure and reliable remote access to these diverse components, enabling remote users to participate in the design, execution, monitoring, and post mortem analysis of hybrid physical-numerical experiments involving apparatus at multiple sites.

1.1.4 Earth System Grid

The Earth System Grid (ESG) has deployed services across several laboratories to enable remote access to many tens of terabytes of climate simulation data produced by climate change researchers [45]. Registered users pose requests via a Web portal, which may in turn trigger the reading, subsetting, and/or transfer of potentially large quantities of data from ESG data servers.

Globus and related services are used for authentication, authorization, data access, data movement, and system monitoring.

1.1.5 Open Science Grid

The Open Science Grid (initially named Grid2003 [54]) has deployed a multi-virtual organization, application-driven grid laboratory for that has sustained for several months the production-level services required by physics experiments of the Large Hadron Collider at CERN (ATLAS and CMS), the Sloan Digital Sky Survey project, the gravitational wave search experiment LIGO, the BTeV experiment at Fermilab, as well as applications in molecular structure analysis and genome analysis, and computer science research projects in such areas as job and data scheduling. This infrastructure has been operating for well over a year with 30 sites, 3000 processors, work loads from 10 different applications exceeding 1300 simultaneous jobs, and data transfers among sites of greater than 2 TB/day. Globus services are used in conjunction with other Grid software to provide VO management, authentication, resource access, data movement, and other related functions.

1.1.6 Biomedical Informatics Research Network

The Biomedical Informatics Research Network (BIRN) [41] is a National Institutes of Health project that is establishing an information technology (IT) infrastructure for collaborative data sharing to enable fundamentally new capabilities in large-scale studies of human disease. The BIRN consortium currently spans 12 universities and 16 research groups [63]. Standard hardware comprising a compute-storage cluster has been deployed at each of these sites. This hardware runs a standard BIRN software suite based on Globus and related technologies for authentication, resource discovery, executable staging and computation management, and secure and uniform access to (and management of) data storage resources.

1.2 Requirements and Architectural Overview

Six examples cannot do justice to the many applications in which Globus has been used, but these examples do capture some important characteristics that tend to reoccur. For example, we note a frequent need to coordinate the use of multiple resources distributed across different computer systems, work units, departments, or even organizations. In this section, we review important characteristics and introduce architectural constructs that Globus uses to address associated requirements.

1.2.1 Standardized Mechanisms and Interfaces

In each of our six examples, a range of software components must interact via message exchanges over a network to perform some task. While such *distributed systems* can be constructed in many different ways, there are significant practical advantages to standardizing the mechanisms used for such common tasks as describing component interfaces, exchanging messages, validating permission to send messages, and requesting that specific tasks be performed. Such standardization can facilitate the construction and understanding of individual components, interoperability among different implementations of the same interface, the sharing of components, and the development of reusable tooling to help with application development.

Motivated by these considerations, GT4 makes heavy use of *Web services* mechanisms (see Chapter 2) to define its interfaces and structure its components. Web services provide flexible, extensible, and widely adopted XML-based mechanisms for describing, discovering, and invoking network services; in addition, its document-oriented protocols are well suited to the loosely coupled interactions that many argue are preferable for robust distributed systems [62]

(see §2.5). GT4 defines Web services interfaces to most (not yet all) of its major components, thus allowing the use of standardized Web services mechanisms to describe GT4 service interfaces and to invoke GT4 service operations.

A wide range of enabling software has been developed in the past few years to support the development of distributed system components that implement Web services interfaces. This software deals with such issues as message handling, resource management, and security, thus allowing the developer to focus their attention on implementing application logic. GT4 packages such software with additional GT4-specific components to provide *GT4 Web services containers* for deploying and managing GT4 services written in Java, C, and Python.

One disadvantage of Web services is the relatively low performance of current implementations. Thus, we may in some cases need to use other protocols for performance-critical operations (e.g., for high-performance data movement). Eventually, we expect faster implementations to emerge.

1.2.2 Infrastructure

While end-user *applications* will typically be concerned with more abstract operations such as pricing a portfolio or analyzing a gene sequence, computing ultimately requires the manipulation and management of *infrastructure*: physical devices such as computers, storage systems, and instrumentation. The service-oriented concepts and Web services mechanisms used to build end-user applications can also be used to access and manage such infrastructure elements, as long as appropriate interfaces are defined.

GT4 implements Web services interfaces for management of computational elements and activities executing on those elements (Grid Resource Allocation and Management service, or GRAM), to instrumentation (Grid TeleControl Protocol, or GTCP), and for managing data transfers (Reliable File Transfer service, or RFT). It also provides GridFTP, a data transfer service for which no Web service interface has yet been defined. These various components enable reliable, secure, and managed interactions with individual resources, and thus provide a basis for building larger infrastructures.

With the exception of GridFTP, the interfaces implemented by these various GT4 components have not yet been standardized in any standards development organization. However, we hope to see progress in each area in the near future.

1.2.3 Monitoring and Discovery

Monitoring and discovery are two vital functions in a distributed system, particularly when that system spans multiple locations as in that context no one person is likely to have detailed knowledge of all components. Monitoring allows us to detect and diagnose the many problems that can arise in such contexts, while discovery allows us to identify resources or services with desired properties. Both tasks require the ability to collect information from multiple, perhaps distributed, information sources.

In recognition of the importance of these functions, monitoring and discovery mechanisms are built in to GT4 at a fundamental level, as follows.

First, GT4 provides standardized mechanisms for associating XML-based *resource properties* with network entities and for accessing those properties via either pull mode (query) or push mode (subscription). These mechanisms—basically implementations of the WSRF and WS-Notification specifications—are built into every GT4 service and container, and can also be incorporated easily into any user-developed service.

Second, GT4 provides three *aggregator services* that collect recent state information from registered information sources. Recognizing that not all interesting information sources will

support WSRF/WS-notification interfaces, these aggregators can be configured to collect data from any arbitrary information source, whether XML-based or otherwise. The three aggregators implement a registry (MDS-Index), archiver (MDS-Archive), and event-driven data filter (MDS-Trigger), respectively.

Third, GT4 provides a range of browser-based interfaces, command line tools, and Web service interfaces that allow users to query and access the collected information.

These different mechanisms provide a powerful framework for monitoring diverse collections of distributed components and for obtaining information about components for purposes of discovery.

1.2.4 Security

While "security" is important in any computer system, security concerns are particularly important and challenging in distributed systems with resources and/or users that span multiple locations. A range of players may want to exert control over who can do what, including the owners of individual resources, the users who initiate computations, and the "virtual organizations" established to manage resource sharing. "Exerting control" may include variously enforcing policy and auditing behavior. When designing mechanisms to address these requirements, we must work not only to protect communications but also to limit the impact of break ins at end system computers. A complete security "solution" in any given solution must always be a system that combines components concerned with establishing identity, applying policy, tracking actions, etc., to meet specific security goals. GT4 and related tools provide powerful building blocks that can be used to construct a range of such systems.

At the lowest level, GT4's highly standards-based security components implement credential formats and protocols that address message protection, authentication, delegation, and authorization. In GT4's default configuration, each user and service is assumed to have a X.509 public key credential. Protocols are implemented that allow two entities to validate each other's credentials, to use those credentials to establish a secure channel for purposes of message protection, and to create and transport delegated credentials that allow a remote component to act on a user's behalf for a limited period of time. Authorization call outs associated with GT4 services can be used to determine whether specific requested operations should be allowed.

Supporting tools, some in GT4 proper and some available from other sources, support the generation, storage, and retrieval of the credentials that GT4 uses for authentication, and address related issues concerning group membership and the like.

1.2.5 Data

A common theme in Globus applications is the need to manage, provide access to, and/or integrate large quantities of data at one or many sites. This "data" problem is tremendously broad and complex, and no single piece of software can claim to "solve" it in any comprehensive sense. However, several GT4 components implement useful enabling mechanisms that can be used individually and in conjunction with other components to develop interesting solutions.

GridFTP provides both libraries and tools for reliable, secure, high-performance memory-to-memory and disk-to-disk data movement. It implements the GridFTP extensions to the popular FTP data transfer protocol, and can interoperate with conventional FTP clients and servers.

The Globus replica location service (RLS) is a highly scalable and robust system for maintaining and providing access to information about the location of replicated files and datasets.

The Globus Data Access and Integration (DAI) tools developed by the UK eScience program provides access to relational and XML data.

1.2.6 Choreography

A final common theme within Globus applications is the need to coordinate activities at different locations. The "activities" to be coordinated may be computational tasks, data transfers, instrument operations, or monitoring or control operations on services and/or physical resources. As this brief list shows, the notion of "coordination" is broad and multifaceted, and thus no single coordination solution is likely to suffice for all purposes.

GT4 provides only limited direct support for coordination itself, but the uniformity of interface and mechanism enabled by GT4 facilitates the development of higher-level coordination tools such as DAGman (for loosely coupled directed acyclic graphs, or DAGs) and MPICH-G2 (for tightly coupled concurrent computations. We discuss these tools in Chapter 4.

1.3 Globus Architecture Summary

We summarize here the principal GT4 elements introduced in this section, each of which is covered in more detail in subsequent chapters.

Service-oriented architecture. GT4 software is designed to support applications in which sets of services interact via standard protocols. The software includes both complete services and libraries implementing useful protocols. Developers can use these services and libraries, plus other related software, to build both simple and complex systems relatively quickly.

Infrastructure services. GT4 includes built in services for accessing, monitoring, managing, and controlling access to such infrastructure elements as computational and data resources.

Web services. GT4 makes heavy use of industry-standard Web services protocols and mechanisms for service description, discovery, access, authentication, authorization, and the like.

GT4 containers. The GT4 software includes components that can be used to construct GT4 containers for hosting Web services written in Java, C, and Python.

Security. A highly standards-based security subsystem addresses message protection, authentication, delegation, and authorization.

Standards. Wherever possible, Globus implements standards or other broadly adopted specifications, so as to facilitate the construction of operable and reusable components and the use of standard tools.

Related tools. GT4 components do not, in general, address end-user needs directly: most are more akin to a TCP/IP library or Web server implementation than a Web browser. Instead, GT4 enables a range of end-user components and tools that provide the higher-level capabilities needed by specific user communities. These components and tools constitute, together with GT4 itself, the "Globus universe."

Limitations. While the Globus software has proved useful in many settings, it is far from complete in terms of its capabilities, and much room remains for improvement in terms of performance, usability, and robustness. We attempt to identify key limitations in the rest of this document.

1.4 Opportunities to Contribute

A large and diverse Globus community is working hard to improve the scope and quality of the Globus software. We hope that you, the reader, will feel inspired to contribute also. There are many ways in which you can do so.

Use the software and report your experiences. Simply using the software and reporting back on your experiences, positive or negative, can be immensely helpful. Reports of problems encountered, particularly when carefully documented, can help guide bug fixes and/or prioritize work on new features. Reports of successful deployments and applications can help justify continued support for the development of the software.

Develop documentation and examples. Despite considerable progress, we remain in desperate need of code examples and associated documentation that can help other users to start work with Globus software or related tools. Take the time to document your successful application, and you will be amply repaid in gratitude from other users and perhaps enhance your reputation also.

Contribute to the development of the software. The list of new features wanted by users is always far greater than Globus developers can handle. You can help by contributing bug fixes, test cases, new modules, or even entirely new subsystems. Depending on the component in question, this may involve working on software under the management of the Apache Software Foundation (www.apache.org) or Globus (www.globus.org).

1.5 Further Reading

The book *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)* [49] includes chapters on many topics touched upon in this chapter, including applications, technologies, and research challenges. However, few details are provided on Globus or other specific technologies.

The article "The Grid: Computing without Bounds" [44] provides a high-level introduction to the Grid vision and the state of the art in terms of technology and applications in early 2004.

The book *Grid Computing: Making the Global Infrastructure a Reality* [28] collects many research papers on Grid computing and its applications in the sciences.

The article "The Anatomy of the Grid" [53] introduces the concept of a virtual organization and presents an architectural picture that distinguishes between different protocols required in Grid deployments and applications.

Chapter 2 Service Oriented Architectures

We introduce basic concepts relating to Web services and their use and implementation within GT4, in particular within the "WS Core" components.

2.1 GT4, Distributed Systems, and Web Services

GT4 is a set of software components for building *distributed systems*: systems in which diverse and discrete software agents interact via message exchanges over a network to perform some tasks. Distributed systems face particular challenges relating to sometimes high and unpredictable network latencies, the possibility of partial failure, and issues of concurrency [62]. In addition, system components may be located within distinct administrative domains, thus introducing issues of decentralized control and negotiation [53].

GT4 is, more specifically, a set of software components that (with some exceptions) implement *Web services* mechanisms for building distributed systems. Web services provide a standard means of interoperating between different software applications running on a variety of platforms and/or frameworks:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [29].

Web services standardize the messages that entities in a distributed system must exchange in order to perform various operations. At the lowest level, this standardization concerns the protocol used to transport messages (typically HTTP), message encoding (SOAP), and interface description (WSDL). A client interacts with a Web service by sending it a SOAP message; the client may subsequently receive response message(s) in reply. At higher levels, other specifications define conventions for securing message exchanges (e.g., WS-Security), for management (e.g., WSDM), and for higher-level functions such as discovery and choreography. Figure 2 presents a view of these different component technologies; we discuss specific specifications in §2.4.

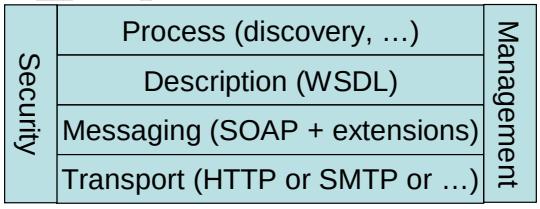

Figure 2: An abstract view of the various specifications that define the Web services architecture

2.2 Service Oriented Applications and Infrastructure

Web services technologies, and GT4 in particular, can be used to build both *service-oriented applications* and *service-oriented infrastructure*. Deferring discussion of the sometimes controversial term "service-oriented" to §2.5, we note that a service-oriented *application* is constructed via the composition of components defined by service interfaces (in the current

context, Web services): for example, a financial or biological database, an options pricing routine, or a biological sequence analyzer. Many descriptions of Web services and SOA focus on the task of defining interfaces to such components, often illustrating their discussion with examples such as a "stock quote service" (the "hello world" of Web services).

Particularly when servicing many such requests from a distributed community, we face the related problem of orchestrating and managing numerous distributed hardware and software components. Web services can be used for this purpose also, and thus we introduce the term service-oriented *infrastructure* to denote the resource management and provisioning mechanisms used to meet quality of service goals for components and applications. Many GT4 features are concerned with enabling the construction of service-oriented infrastructure.

2.3 Web Services Implementation

From the client perspective, a Web service is simply a network-accessible entity that processes SOAP messages. Things are somewhat more complex under the covers. To simplify service implementation, it is common for a Web services implementation to distinguish between:

1. the *hosting environment* (or *container*), the (domain-independent) logic used to receive a SOAP message and identify and invoke the appropriate code to handle the message, and potentially also to provide related administration functions, and:
2. the *Web service implementation*, the (domain-specific) code that handles the message.

This separation of concerns means that the developer need only provide the domain-specific message handling code to implement a new service.

It is also common to further partition the hosting environment logic into that concerned with transporting the SOAP message (typically via HTTP, thus an "HTTP engine" or "Web server"—sometimes termed an "application server") and that concerned with processing SOAP messages (the "SOAP engine" or "SOAP processor"). Figure 3 illustrates these various components.

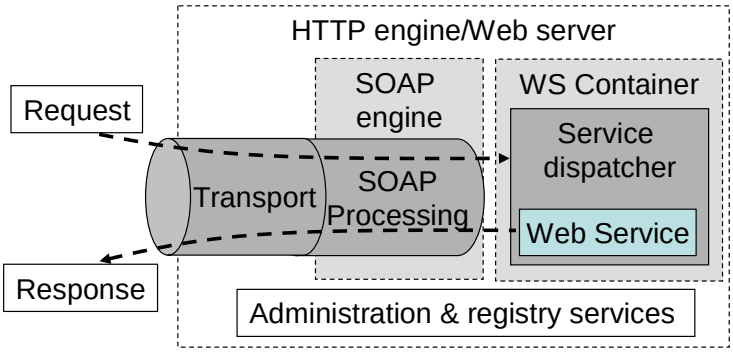

Figure 3: High-level picture of functional components commonly encountered in Web service implementations, showing the path taken by requests and responses.

Many different containers exist, with different performance properties, supported Web services implementation languages, security support, and so forth. We mention below those used in GT4.

2.4 Web Services Specifications

We provide pointers to the Web services specifications that underlie GT4. These comprise the core specifications that define the Web services architecture (XML, SOAP, WSDL); WS-Security

and other specifications relating to security; and the WS-Addressing, WSRF, and WS-Notification specifications used to define, name, and interact with stateful resources. We also speak briefly to emerging specifications that are likely to be important in future GT evolution. An important source of information on the requirements that motivate the use and development of these specifications is the Open Grid Services Architecture [46].

2.4.1 XML, SOAP, WSDL

XML is used extensively within Web services as a standard, flexible, and extensible data format. In addition to XML syntax, other important specifications are **XML Schema** [15] and **XML Namespaces** [32]. Note that while current Web services tools typically adopt a textual serialization, a binary encoding is also possible and may provide higher efficiency.

SOAP 1.2 [14] provides a standard, extensible, composable framework for packaging and exchanging XML messages between a service provider and a service requestor. SOAP is independent of the underlying transport protocol, but is most commonly carried on HTTP.

WSDL 1.1 [38] is an XML document for describing Web services. Standardized binding conventions define how to use WSDL in conjunction with SOAP and other messaging substrates. As discussed in §3.7, WSDL interfaces can be compiled to generate proxy code that constructs messages and manages communications on behalf of the client application. The proxy automatically maps the XML message structures into native language objects that can be directly manipulated by the application. The proxy frees the developer from having to understand and manipulate XML.

2.4.2 WS-Security and Friends

The **WS-Security** family of specifications addresses a range of issues relating to authentication, authorization, policy representation, and trust negotiation in a Web services context [71]. GT4 uses a number of these specifications plus other related specifications, notably Security Authorization Markup Language (**SAML**) [59], to address message protection, authentication, delegation, and authorization, as follows [3]:

- TLS (transport-level) or WS-Security and WS-SecureConversation (message level) are used as message protection mechanisms in combination with SOAP.
- X.509 End Entity Certificates or Username and Password are used as authentication credentials.
- X.509 Proxy Certificates and WS-Trust are used for delegation.
- SAML assertions are used for authorization.

2.4.3 WS-Addressing, WSRF, and WS-Notification

A number of related specifications provide functionality important for service oriented infrastructure in which we need to be able to represent and manipulate stateful entities such as physical resources of various kinds, logical components such as software licenses, and transient activities such as tasks and workflows.

The **WS-Addressing** [30] specification defines transport-neutral mechanisms to address Web services and messages. Specifically, this specification defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages.

The **WS Resource Framework** (WSRF) specifications define a generic and open framework for modeling and accessing stateful resources using Web services [10, 47]. This framework comprises mechanisms to describe views on the state (WS-ResourceProperties), to support

management of the state through properties associated with the Web service (WS-ResourceLifetime), to describe how these mechanisms are extensible to groups of Web services (WS-ServiceGroup), and to deal with faults (WS-BaseFaults).

The **WS-Notification** family of specifications define a pattern-based approach to allowing Web services to disseminate information to one another [9, 56]. This framework comprises mechanisms for basic notification (WS-Notification), topic-based notification (WS-Topics), and brokered notification (WS-BrokeredNotification).

We note that the Web services standards space is in some turmoil due to competing proposed specifications. In particular, Microsoft and others recently proposed WS-Transfer [19], WS-Eventing [31], and WS-Management [23], which define similar functionality to WSRF, WS-Notification, and WSDM (discussed below), respectively, but using different syntax. We hope that these differences will be resolved in the future.

2.4.4 Other Relevant Specifications

GT4 includes an implementation of the **GridFTP** extensions to FTP recently finalized within GGF [20] and the DAIS implementation of the GGF **OGSA-DAI** specification under development within GGF .

The WS-Interoperability (WS-I) organization [24] has produced a number of *profiles* that define ways in which existing Web services specifications can be used to promote interoperability among different implementations. The **WS-I Basic Profile** speaks to messaging and service description: primarily XML, SOAP, and WSDL. The **WS-I Basic Security Profile** speaks to basic security mechanisms. Other profiles are under development.

Web services distributed management (**WSDM**) specifications under development within OASIS [16] are likely to play a role in future GT implementations as a means of managing GT components.

WS-CIM specifications under development within DMTF [8] are likely to play a role in future GT implementations as a means of representing physical and virtual resources.

The Global Grid Forum's **Open Grid Services Architecture** (OGSA) working group has completed a document that provides a high-level description of the functionality required for future service-oriented infrastructure and applications, and a framework that suggests how this functionality can be factored into distinct specifications [1]. The OGSA working group is now proceeding to define OGSA Profiles that, like WS-I profiles, will identify technical specifications that can be used to address specific Grid scenarios.

2.5 Service Oriented Architecture

We provide some additional discussion concerning the term *service oriented architecture* (SOA), which is used widely but not necessarily consistently within the Web services community. One common usage is simply to indicate the use of Web services technologies. However, the intention of those who coined the term seems to be rather to contrast two different styles of building distributed systems. Distributed object systems are [46]:

> distributed systems in which the semantics of object initialization and method invocation are exposed to remote systems by means of a proprietary or standardized mechanism to broker requests across system boundaries, marshall and unmarshall method argument data, etc. Distributed objects systems typically (albeit not necessarily) are characterized by objects maintaining a fairly complex internal state required to support their methods, a fine grained or "chatty" interaction between an object and a program using it, and a focus on a

> shared implementation type system and interface hierarchy between the object and the program that uses it.

In contrast, a Service Oriented Architecture (SOA) is [29]:

> a form of distributed systems architecture that is typically characterized by the following properties:
>
> - Logical view: The service is an abstracted, logical view of actual programs, databases, business processes, etc., defined in terms of what it does, typically carrying out a business-level operation.
> - Message orientation: The service is formally defined in terms of the messages exchanged between provider agents and requester agents, and not the properties of the agents themselves. The internal structure of an agent, including features such as its implementation language, process structure and even database structure, are deliberately abstracted away in the SOA: using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed. A key benefit of this concerns so-called legacy systems. By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application that can be "wrapped" in message handling code that allows it to adhere to the formal service definition.
> - Description orientation: A service is described by machine-processable metadata. The description supports the public nature of the SOA: only those details that are exposed to the public and important for the use of the service should be included in the description. The semantics of a service should be documented, either directly or indirectly, by its description.
> - Granularity: Services tend to use a small number of operations with relatively large and complex messages.
> - Network orientation: Services tend to be oriented toward use over a network, though this is not an absolute requirement.
> - Platform neutral: Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.

It is argued that these features can allow service-oriented architectures to cope more effectively with issues that arise in distributed systems, such as problems introduced by latency and unreliability of the underlying transport, the lack of shared memory between the caller and object, problems introduced by partial failure scenarios, the challenges of concurrent access to remote resources, and the fragility of distributed systems if incompatible updates are introduced to any participant [29, 62].

Web services technologies in general, and GT4 in particular, can be used to build both distributed object systems and service-oriented architectures. The specific design principles to be followed in a particular setting will depend on a variety of issues, including target environment, scale, platform heterogeneity, and expected future evolution.

2.6 Further Reading

The document "Web Services Architecture" [29] provides a good introduction to the concepts and technologies that underlie Web services.

The article “A Note on Distributed Computing” [29] motivates and describes the distinction between service-oriented architectures and distributed object systems.

The article “Grid Services for Distributed System Integration” [62] (a longer version is available as “The Physiology of the Grid” [51]) motivates and introduces the concepts that underlie what is now called the Web services resource framework, which is described in more detail in the article “Modeling Stateful Resources with Web Services” [50].

Chapter 3 GT4 Architecture

XX.

3.1 Architecture Overview

As shown in Figure 4, GT4 comprises both a set of service implementations ("server" code) and associated "client" libraries. GT4 provides both Web services (WS) components (on the left) and non-WS components (on the right). The white boxes in the "client" domain denote custom applications and/or third-party tools that access GT4 services or GT4-enabled services.

Note that all GT4 WS components use WS-Interoperability-compliant transport and security mechanisms, and can thus interoperate with each other and with other WS components. In addition, all GT4 components, both WS and non-WS, support X.509 end entity certificates and proxy certificates. Thus a client can use the same credentials to authenticate with any GT4 WS or non-WS component.

This figure doesn't show all GT4 components: some (e.g., various security and communication libraries) are introduced later. All components are described in more detail below.

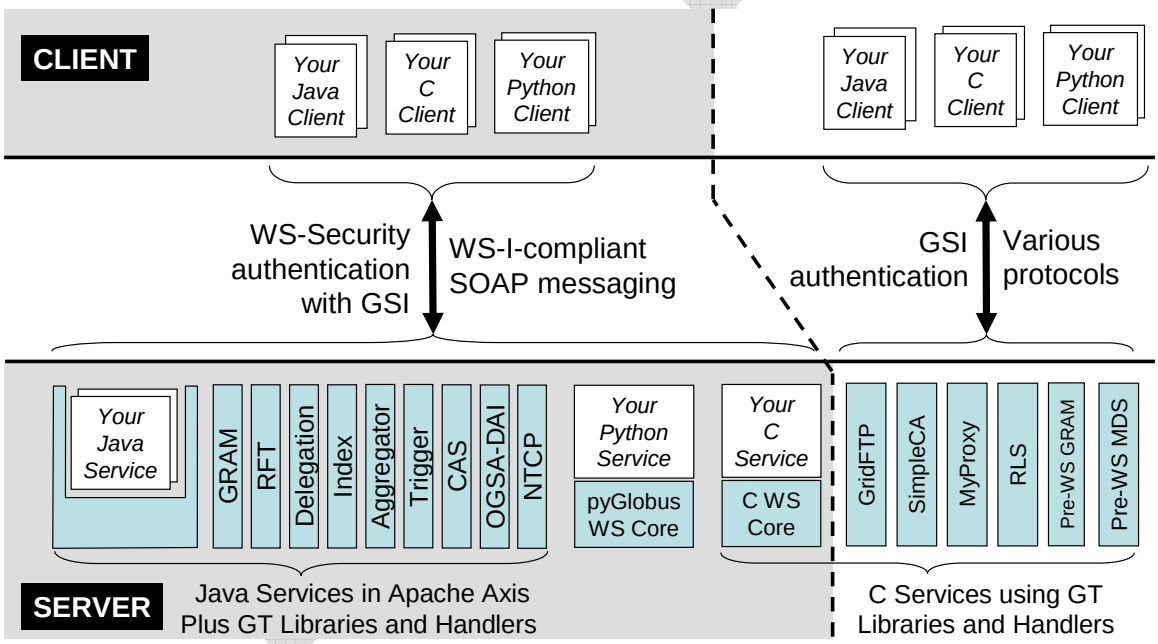

Figure 4: Schematic view of GT4.0 components

3.2 Predefined GT4 Services

GT4 provides a set of predefined services, which we list briefly here and describe in more detail in Chapter 4.

Nine GT4 services implement Web services (WS) interfaces: job management (GRAM: §4.2); reliable file transfer (RFT: §5.2); delegation (§7.3.2); MDS-Index, MDS-Trigger, and MDS-Archive (collectively termed the Monitoring and Discovery System, or MDS: §6.1); community authorization (CAS: §7.3.1); OGSA-DAI data access and integration (§5.4); and GTCP Grid

TeleControl Protocol for online control of instrumentation (§10.2). Of these, MDS-Archive, GTCP, and OGSA-DAI are "tech previews," meaning that their interface and implementation may change in the future.

For two of those services, GRAM and MDS-Index, pre-WS "legacy" implementations are also provided. These pre-WS implementations will be deprecated at some future time as experience is gained with WS implementations.

For three additional GT4 services, WS interfaces are not yet provided (but will be in the future): GridFTP data transport (§5.1), replica location service (RLS: §5.3), and MyProxy online credential repository (§7.4.1).

Other libraries provide powerful authentication and authorization mechanisms (§XX), while the eXtensible I/O (XIO) library provides convenient access to a variety of underlying transport protocols (§10.1). SimpleCA is a lightweight certification authority (§7.4.2).

3.3 GT4 Command Line Programs

As our focus in this document is primarily on the user view of Grids and GT4, it is natural to approach GT4 from the perspective of its command line programs. Table 1 summarizes the command line programs included with GT4. We describe each of these programs at least briefly in this document.

Needless to say this table does not provide a complete picture of the functionality that GT4 makes available to users. In particular, it does not capture the client APIs, libraries, and containers that GT4 provides, or the powerful capabilities provided by related tools.

- Client APIs: Most GT4 components also provide client APIs that allow components to be invoked directly from programs written in Java, C, and Python.
- Libraries and containers: Some GT4 XX
- Other … XX

Table 1: GT4 command line programs. Each line represents a set of programs and gives the number of programs in the set, a brief description, and a pointer to a section with more details.

Command	#	Description	§
cas-*	14	Community authorization service	7.2.1
globus-credential-*	2	Install and refresh credentials in Delegation service.	7.2.2
myproxy-*	10	Store credentials in, and administer, a MyProxy service.	7.4.1
grid-ca-sign	1	SimpleCA for generating X.509 credentials.	7.4.2
gsi*	3	GSI-enabled OpenSSH, with ssh, scp, and sftp clients.	7.5
grid-*	10	Manage X.509 proxy credentials and gridmap files.	XX
globus-url-copy	1	GridFTP client.	5.1
rft*	2	Reliable file transfer client.	5.2
globus-rls-*	3	RLS client, administration, server.	5.3
globusrun-ws	1	GRAM client.	4.2
mds-servicegroup-add	1	Add an entry to a MDS servicegroup.	6.1
globus-*-container	2	Start and stop a Globus container.	XX
wsrf-*	8	Interact with WSRF resource properties.	XX
wsn-*	4	Create and manage WS-Notification subscriptions.	XX

3.4 GT4 Security

XX.

3.5 GT4 Containers

The GT4 source code includes implementations of a set of Web services specifications important for Grid applications. Some of these specifications are, or can be expected to become, "standards" (e.g., WSRF, WS-Notification) while others are currently unique to Globus (e.g., GRAM, RFT).

These implementations can be combined with other components (Web servers, SOAP engines, implementations of other Web services specifications, etc.) to produce a variety of different *GT4 containers* (the white boxes in Figure 4) a term that we introduce to denote Web service containers with a set of common features. A GT container:

- implements SOAP over HTTP as a message transport protocol, and both transport-level and WS-Security message-level security for all communications;
- implements WS-Addressing, WSRF, and WS-Notification functionality;
- supports logging via Log4j [47], which implements the Jakarta Commons Logging API; and
- defines WSRF WS-Resources with properties providing access to information about services deployed in the container and container properties such as version and start time.

Thus, any GT4 container can (see Figure 5):

1. host services whose client interfaces are defined in terms of only basic WS specifications ("custom Web services") and
2. host services whose client interfaces make use of WSRF and related mechanisms ("custom WSRF Web services").

If appropriate software is installed, then a GT4 Java container can also:

3. host advanced services provided by GT4, such as GRAM, MDS, and RFT ("GT4 WSRF Web services").

Regardless of what services are deployed, application clients can:

4. use GT4 container registry interfaces to determine which services are hosted in a particular GT4 container, and GT4 container administration interfaces to perform basic administration functions.

We note that all GT4 container interfaces are compliant with the WS-I Basic Profile, and can be configured to be compliant with the WS-I Basic Security Profile (see §2.4.4).

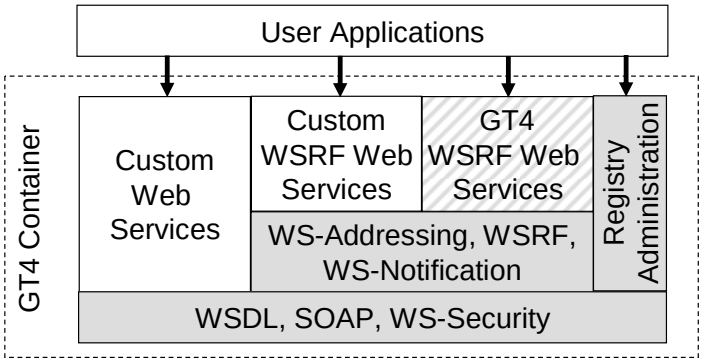

Figure 5: GT4 containers incorporate services and tools (shaded) that allow them to host different services, including optional GT4 WSRF Web services, and support discovery and administration.

The GT4 distribution includes software and instructions to construct GT4 containers (Figure 6) for three different Web service implementation languages (Java, C, and Python).

GT4 Java Containers. GT4 support for Web services implemented in Java comprises the GT4 Java WS Core code, which implements WSRF and WS-Notification as well as supporting code for security and management. This code is designed to be used with Apache Axis as a SOAP engine plus other relevant Apache components such as the WS-Addressing and WS-Security implementations. To produce a complete GT4 Java container, we can host the GT4 Java WS Core + Axis combination either in a "simple Java container" provided by GT4 or alternatively in a more featureful but complex servlet container such as Tomcat. The former approach provides for easier installation and administration and is recommended unless a site or resource already runs Tomcat for other purposes. Such a GT4 Java container can also host various GT4 services implemented in Java, such as GRAM, RFT, MDS-Index, MDS-Trigger, and MDS-Archive.

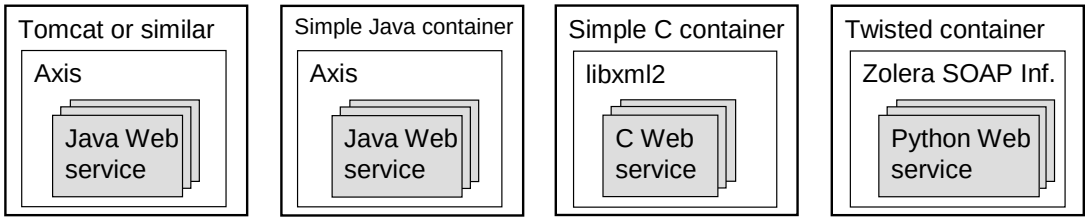

Figure 6: Four different GT4 container configurations.

GT4 C Container. GT4 support for Web services written in C comprises the GT4 C WS Core, which implements WSRF and WS-Notification as well as supporting code for security and management.

Python Web services. GT4 support for Web services written in Python comprises the GT4 Python WS Core, which implements WSRF and WS-Notification as well as supporting code for security and management. To produce a complete GT4 Java container, we combine this code with the Zolera SOAP Infrastructure and Twisted container.

3.6 Deploying GT4 Web Services

The term "deploy" refers to the task of installing, and initiating the execution of, a Web service on a particular computer. Depending on the environment into which the service must be deployed, we may need first to install and configure a GT4 container. We discuss here the general techniques that are used to deploy both GT4 predefined services (e.g., GRAM, RFT) and other Web services into GT4 containers.

A Web service container can host zero or more Web services that each may make use of different container functions. Thus, depending on context, "deploying a Web service" may involve:

1. Deploying the Web service into an already deployed GT4 container;
2. First loading additional libraries (e.g., WSRF) to "GT4-enable" an already deployed non-GT4 container, and then deploying the service into that container;
3. First deploying a new GT4 container, into which the Web service is then deployed; or
4. Deploying the combination of GT4 container and Web service at the same time.

A GT4 distribution will typically provide convenient packages to facilitate various of these tasks in different environments, so that the user is not forced to execute many installation and configuration steps to produce a system capable of running a specific service. For example, if faced with the task of deploying GT4 GRAM (written in Java) on a computer, then:

- If a GT4 Java container is already installed, we can deploy the GRAM service into that directly.
- If a site already has Tomcat installed, but no Globus-specific code, then we need to deploy WSRF, WS-Notification, and GRAM.
- For a deployment "from scratch," we might be provided with a complete distribution that comprises the GT4 Java container and GRAM Web service. This distribution can simply be installed, configured, and run (approach #4).

3.7 Developing GT4 Web Services: Java Specifics

We make a few additional comments concerning the specific mechanisms used to develop and deploy Java services with Apache Axis. The primary steps involved are depicted in Figure 7 and described in the following:

1. *Define the service's interface*. Here, we write a Web Service Description Language (WSDL) file describing the service's abstract interface.
2. *Implement the service*. Here, we develop Java routines for the service implementation and, if WSRF mechanisms are used, for associated resource properties.
3. *Define deployment parameters*. Here, we write a Web Services Deployment Descriptor (WSDD) file that describes various aspects of the service's configuration.
4. *Compile everything and generate a GAR file*. Compilation generates application-specific interface routines that handle the demarshalling/marshalling of the Web service's arguments from/to SOAP messages.
5. Deploy the service.

Borja Sotamayor's excellent tutorial [6] provides details on these various steps; here we provide just a few notes what is involved in each of the first three stages.

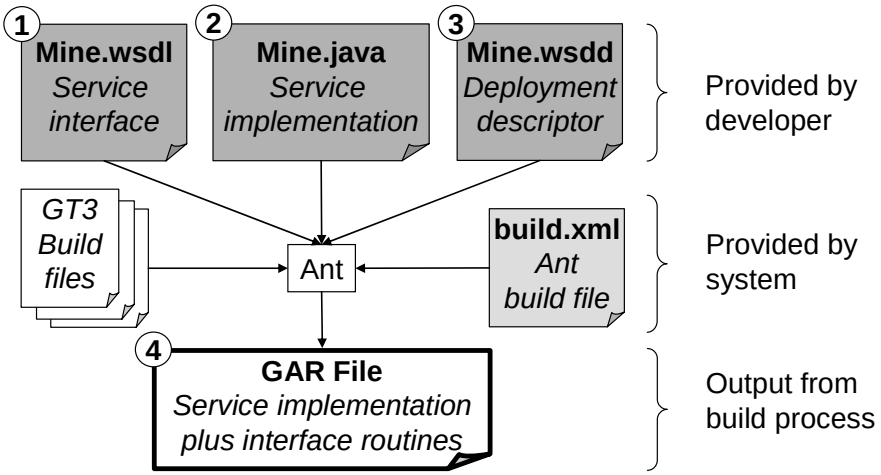

Figure 7: A Java Web service and the various components (shaded) that Axis uses to implement its interface and describe its configuration.

We note that Axis also provides tools that can generate WSDL directly from either the Java source code (if the service's arguments are simple Java types) or from the WSDD file. However, WSDL generated automatically from service code often contain language-specific or implementation-specific types that may not interoperate with other implementations of the same service. Thus, GT4 requires that the developer supply WSDL.

3.7.1 Interface: WSDL

A WSDL document describes a Web service, providing information that a client needs to connect and communicate with the service. Specifically, it defines via its *portType* component the Web service's abstract interface, specifying the operations that the service supports and, for each operation, the format of the messages that the service sends and receives. If the service implements operations on WSRF resource properties, then the WSDL will include components that define and deal with resource properties: see, for example, the Sotamayor tutorial [73].

A complete WSDL document must also include a binding component to specify how the abstract interface maps to concrete protocol messages. GT4 generates this component automatically.

3.7.2 Implementation: Java

The service implementation proper provides the Java code for the various service operations. If a service uses WSRF mechanisms, then the service implementation must also include code for the *resource implementation* and the *resource home*. Again, see the Sotamayor tutorial [73] for details.

3.7.3 Deployment Descriptor: WSDD

The deployment descriptor file tells the web server how it should publish our Web service, by specifying, for example, our service's URI. The deployment descriptor is written in WSDD (Web Service Deployment Descriptor) format.

3.8 Further Reading

We have already referred to *The Globus Toolkit 4 Programmer's Tutorial* [73], the single most important text for those wanting to learn to write GT4 programs.

Chapter 4 Execution Management

"Nothing focuses the mind like the prospect of imminent execution." (Anonymous.)

So you want to:

- Make a program available as a network service (with size varying)
- Dispatch
- Run an executable on a remote computer.
- Run an parallel program across multiple distributed computers.
- Run a set of loosely coupled tasks
- Steer a computation (?)

These tasks all fall within the purview of execution management.

…

Execution management tools are concerned with the initiation, monitoring, management, scheduling, and/or coordination of remote computations. GT4 supports the Grid Resource Allocation and Management (GRAM) interface as a basic mechanism for these purposes. The GT4 GRAM server is typically deployed in conjunction with Delegation and RFT servers to address data staging, delegation of proxy credentials, and computation monitoring and management in an integrated manner.

Associated tools fall into three main classes. First, we have GRAM-enabled schedulers for clusters or other computers on a local area network (Condor [73], OpenPBS, Torque, PBSPro, SGE, LSF). Second, we have systems that provide different interfaces to remote computers (OpenSSH) or that implement various parallel programming models in Grid environments by using GRAM to dispatch tasks to remote computers (Condor-G [64], DAGman, MPICH-G2 [55], GriPhyN VDS, Nimrod-G [60]). Third, we have various "meta-schedulers" that map different tasks to different clusters (CSF, Maui).

In this chapter, and those that follow, we summarize Globus and related tools in tabular form. Table 2 provides this summary for execution management tools. In this list, we *identify GT4 components by using italics*. In addition to a brief description of each component, we provide a rough gauge of its estimated maturity ("M"). This field indicates whether a component is a staple of many Grid deployments (3); has some amount of promising deployment (2); or is research software and/or has not been widely deployed (1). The "G" column indicates, for GT4 components, whether the component is a core toolkit component (*T*) or alternatively a contributed component (*C*), and for other components, whether they are already compatible with the GT4 release (*Y*) or is expected to be soon (*N*).

The Globus Toolkit provides a suite of Web services with which clients can interact to submit, monitor, and cancel jobs on local or remote computing resources. This system is termed collectively "GRAM" or sometimes "WS_GRAM" to distinguish it from the pre-Web services system that offers similar functionality. We describe here only the Web services-based system, as this offers more functionality.

GT4 provides both a GRAM command line client and Java, C, and Python client APIs. We describe the command line functionality here; the APIs provide access to essentially the same functionality.

GRAM is also interesting as an example of how to use WSRF and WS-Notification mechanisms to structure a Web services management interface. Thus, we spend some time describing the GRAM implementation.

GRAM is likely to evolve significantly in the future both to exploit emerging standards and to encompass additional resource management functions, including virtualization. We comment briefly on this work also.

Table 2: Globus and related execution management tools

Name	Purpose	M	G
Grid Resource Allocation & Management service	*GRAM service supports submission, monitoring, and control of jobs on computers. Interfaces to Unix shell ("fork"), Platform LSF, PBS, and Condor schedulers; others may be developed. Includes support for MPICH-G2 jobs: multi-job submission, process coordination in a job, sub-job coordination in a multi-job.*	*3*	*T*
Java CoG Kit Workflow	*Uses the Karajan workflow engine that supports DAGs, conditions, & loops; directs tasks to GRAM servers for execution.*	*1*	*C*
Community Scheduler Framework	*CSF is an open source meta-scheduler based on the WS-Agreement specification.*	*1*	*C*
Condor-G [18]	Manage the execution of jobs on remote GRAM-enabled computers, addressing job monitoring, logging, notification, policy enforcement, fault tolerance, and credential management.	3	Y
DAGman	Manage the execution of directed acyclic graphs (DAGs) of tasks that communicate by writing/reading files; works with Condor-G.	3	Y
MPICH-G2 [55]	Execute parallel Message Passing Interface (MPI) programs over one or more distributed computers.	3	Y
Nimrod-G [60]	Declarative specification of parameter studies, and management of their execution on distributed computers. Scheduler based on computational economy provides soft real-time deadlines. User interaction via command line or web portal.	3	Y
Ninf-G	An implementation of the GridRPC remote procedure call specification, for accessing remote services.	3	Y
GriPhyN Virtual Data System	Tools for defining, scheduling, and managing complex data-intensive workflows. Workflows can be defined via a high-level virtual data language; a virtual data catalog is used to track current and past executions. Includes heuristics for job and data placement. Uses DAGman/Condor-G for execution management.	2	Y
Condor, OpenPBS, Torque, PBSPro, Sun Grid	Schedulers to which GSI-authenticated access is provided via a GRAM interface. The open source Condor is specialized for managing pools of desktop systems. OpenPBS and Torque are open source versions of the Portable Batch System (PBS) cluster	3	Y

Engine, Load Sharing Facility	scheduler; PBSPro is a commercial version produced by Altair. SGE is also available in both open source and commercial versions. LSF is a commercial system produced by Platform.		
Maui Scheduler	An advanced job scheduler for use on clusters and supercomputers, with support for meta-scheduling.	3	?

4.1 Context

XX

4.2 GRAM Overview

Imagine that you want to allow remote users to execute a program. One approach is to make this program accessible as a Web service. To do this, you define and implement a Web service operation that executes your program. Then, for example, a client might simply call "foo(a,b)" to request that "foo" be executed on a local or remote computer with "a" as input and returning "b."

This simple approach to enabling remote access to programs can be quite appropriate in many settings. However, there are also requirements that this simple approach does not address:

- *State*. The computational task that is to be run (the "job") may perform input/output operations while running that affect the state of the computational resource on which the job runs, and/or its associated filesystems. Thus, "exactly once" execution semantics are important: a user cannot simply resubmit a request if no acknowledgement was received, as the job may have completed but the acknowledgement message was lost.
- *User executables*. We may wish to allow users to supply the programs to be executed.
- *Staging of input and output*. Executables, input data, and output data may be large, remote, and/or shared between different invocations. The ability to manage staging of this data can be important.
- *Streaming output*. Some users, particularly interactive ones, benefit from accessing output data files as the job is running.
- *Control*. A user may require the ability to terminate a job that consumes many resources.
- *Schedulers*. Larger computing resources are typically operated under the control of a scheduler that implements allocation and prioritization policies while optimizing the execution of all submitted jobs for efficiency and performance.
- *Monitoring*. Data staging and schedulers introduce the potential for more complex job state transitions: a job may not simply be running, completed, or failed, but may be pending, suspended, staging, and so forth. Some users require the ability to query and/or subscribe to determine job state.

The GRAM system and its client software have been developed to address these issues. *GRAM is meant to be used in situations where the ability to run arbitrary programs, achieve reliable operation, perform stateful monitoring, manage credentials, stage files, and interact with schedulers are important*. If these capabilities are not required, then GRAM may not be appropriate. If an application has only modest input and output data, operates in a stateless manner (i.e., all that matters is the result data or fault signal), and will be invoked many times, it may be a good candidate for exposure as an application-specific Web service.

As we shall see, the GRAM interface addresses this need for advance management functionality by creating for each successful job submission a stateful entity—a ManagedJob—on the compute host, with lifetime similar to that of the associated job. (In fact, the lifetime is typically somewhat longer, so that a client can determine a job's state even after it has terminated.) A successful GRAM "submit" operation returns a handle—specifically, a WS-Addressing endpoint reference, or EPR—for the new ManagedJob. A client can then use this handle to query the job's status, kill the job, and/or "attach" to the job to obtain notifications of changes in job status and output produced by the job. The client can also communicate this handle to other clients, who can perform the same operations if authorized. It is this ManagedJob construct that allows GRAM to support many of the advanced features mentioned in the preceding subsection.

We will also describe GRAM from the perspective of the *system administrator*. A key notion here is that GRAM is not a resource scheduler, but rather a protocol engine for communicating with a range of different local resource schedulers using a standard message format. The GT4 GRAM implementation includes interfaces to Condor, LSF, SGE, and PBS schedulers, as well as a "fork scheduler" that simply forks a new Unix process for each request. The system administrator must perform appropriate local configuration operations for these local scheduler interfaces.

Finally, we describe various aspects of the GRAM *implementation*. As we shall see, the GRAM implementation makes use of various other GT4 components to achieve a modular implementation that provides substantial functionality.

4.3 The globusrun-ws Command Line Client

As we will see, globusrun-ws is a powerful tool. It supports not only basic job submission, monitoring, and management functions, but also more sophisticated capabilities such as file staging, credential delegation, exactly-once job submission, timeouts, message privacy, and authorization, as we describe in the following. We will introduce these various features in stages.

4.3.1 Basic Job Submission

As a first example, we present the globusrun-ws command that would be used to execute a program "/bin/touch" with argument "touched_it."

```
% globusrun-ws -submit -job-command /bin/touch touched_it
```

The components of this command provide in turn the command name, the -submit flag to indicate that this is a job submission, the -job-command flag to indicate that the remainder of the command line comprises a command, the program name, and (in this case) the program's single argument. Assuming no errors, the program will be submitted and will run. The globusrun-ws command prints some status information and then terminates.

```
% globusrun-ws -submit -job-command /bin/touch touched_it
Submitting job...Done.
Job ID: uuid:c51fe35a-4fa3-11d9-9cfc-000874404099
Termination time: 12/17/2004 20:47 GMT
Current job state: Active
Current job state: CleanUp
Current job state: Done
Destroying job...Done.
```

GT4 GRAM also supports the use of an XML-based job description language to describe job submissions. We will examine this language in more detail later, and thus we simply show here how the preceding example is expressed in this language. We first display ("cat") the file and then execute the program, using the -job-description-file option on the command line.

```
% cat touch.xml
<job>
   <executable>/bin/touch</executable>
   <argument>touched_it</argument>
</job>
% globusrun-ws -submit -job-description-file touch.xml
```

We have not yet specified where our program is to execute. In the general case, the target GRAM server is specified by an EPR provided in a file named in the command. For example:

```
% cat gram.epr
<factoryEndpoint xmlns:gram="http://www.globus.org/namespaces/2004/10/gram/job"
                 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
   <wsa:Address>
      https://viz-login.isi.edu:9000/wsrf/services/ManagedJobFactoryService
   </wsa:Address>
   <wsa:ReferenceProperties>
      <gram:ResourceID>PBS</gram:ResourceID>
   </wsa:ReferenceProperties>
</factoryEndpoint>
% globusrun-ws -submit -job-description-file touch.xml -factory-epr-file gram.epr
```

Clearly, providing such an EPR can be unwieldy, and thus globusrun-ws also provides the option of constructing an EPR automatically from a specified GRAM service URL (-factory) and type (-factory-type). Thus, instead of providing the EPR as above, I can write:

```
% globusrun-ws -submit \
      -factory https://viz-login.isi.edu:9000/wsrf/services/ManagedJobFactoryService \
      -factory-type PBS -job-command /bin/touch touched_it
```

The globusrun-ws command follows a convention for constructing an EPR from a URL and a type. This convention is not standardized, so the globusrun-ws command might not work against an implementation of GRAM provided by someone else.

Recall that the first two examples in this section did not specify a target GRAM server. In the absence of any specification, -factory and -factory-type default to localhost and fork, respectively, and thus the first example command is equivalent to a request to submit the specified program to a fork GRAM server running on the local computer.

```
% globusrun-ws -submit -factory localhost -factory-type fork \
                -job-command /bin/touch touched_it
```

4.3.2 Interacting with a Submitted Job

In globusrun-ws's default submission mode, used in the examples above, the globusrun-ws client remains running after the job is submitted. This persistent connection to the job serves two purposes. First, as shown in the example above, globusrun-ws can display status information about the job. Second, globusrun-ws can propagate kill signals to the submitted job: killing globusrun-ws results in an attempt to kill the submitted job.

This *interactive mode* is useful in many circumstances. However, a client may also want to have globusrun-ws submit a job and return immediately (*batch mode*: -batch). If a job is submitted in batch mode, or if network connectivity is lost following an interactive mode submission, then a user may wish to query to determine the status of a job. To support this GRAM allows a user to request that an EPR to the new ManagedJob resource created for a new job be returned following job submission. This is done by using the -job-epr-output-file flag to specify a file in which this EPR should be placed:

```
% globusrun-ws -submit -job-epr-output-file mj.epr -job-command /bin/touch touched_it
```

A client that obtains such a ManagedJob EPR can use it to enquire as to the job's status, as follows:

```
% globusrun-ws  -status  -job-epr-file mj.epr
```

or to (re)attach to the job:

```
% globusrun-ws  -monitor  -job-epr-file mj.epr
```

or to terminate the job:

```
% globusrun-ws  -monitor  -job-epr-file mj.epr
```

4.3.3 Job Status and Lifecycle

We referred to job status. As detailed in Table 3 and Figure 8, GRAM assumes a job execution model by which a job proceeds through a series of states, from initially *Unsubmitted* to eventually *Done* or *Failed*.

Table 3: GRAM job states

State	Description
Unsubmitted	Job has not yet been submitted.
StageIn	Job is waiting for stage in of executable or input files to complete.
Pending	The local scheduler has not yet scheduled the job for execution.
Active	Job is executing.
Suspended	Job execution has been suspended.
StageOut	Job execution has completed; output files are being staged out.
CleanUp	Job execution and stage out have completed; clean up tasks are underway.
Done	Job has completed successfully.
Failed	Job failed.

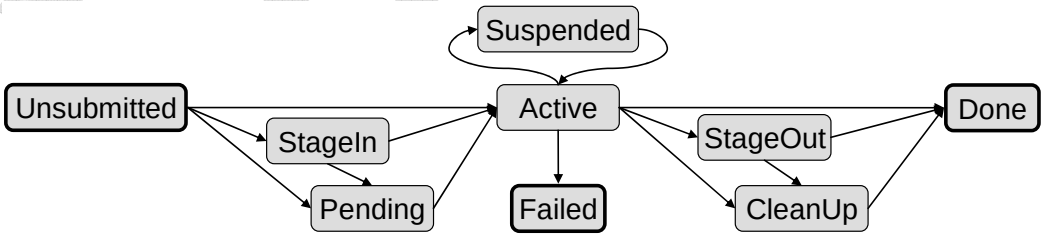

Figure 8: State transition diagram for GRAM jobs.

4.3.4 File Staging and Other Features

In all examples so far, the executable has been assumed to be located on the target computer, input data is passed as arguments, and output data has been written to files on the target computer. However, there will also be cases in which the executable or other files need to be *staged in* to the target computer and/or output files must be *staged out*. We specify file staging by adding file transfer directives to the job description. These directives follow the RFT syntax, which enables third-party transfers. Each file transfer must therefore specify a source URL and a destination

URL. URLs are specified as GridFTP URLs (for remote files) or as file URLs (for local files). For example, in the following example of a stage-in request, the source URL is a GridFTP URL (here, gsiftp://job.submitting.host:2888/tmp/mySourceFile) that resolves to a source document accessible on the file system of the job submission machine (here, /tmp/mySourceFile). At run-time, the Reliable File Transfer service used by the GRAM service on the remote machine fetches the remote file using GridFTP and writes this file to the specified local file (here, file:///${GLOBUS_USER_HOME}/my_file, which resolves to ~/my_file).

```
<fileStageIn>
  <transfer>
    <sourceUrl>gsiftp://submit.host:2888/tmp/mySourceFile</sourceUrl>
    <destinationUrl>file:///${GLOBUS_USER_HOME}/my_file</destinationUrl>
  </transfer>
</fileStageIn>
```

Note: additional RFT-defined quality of service requirements can be specified for each transfer. See the RFT documentation for more information.

The symbol ${GLOBUS_USER_HOME} is one of four that the GRAM server will substitute at run time, as follows:

- GLOBUS_USER_HOME: The path to the home directory for the local account/user
- GLOBUS_USER_NAME: The local account the job is running under
- GLOBUS_SCRATCH_DIR: An alternative directory where a file system is shared with the compute nodes that a user might want to use. Typically it will provide more space than the users default HOME dir. This directory's value may contain ${GLOBUS_USER_HOME}, which will be replaced with value of that substitution.
- GLOBUS_LOCATION: Path to the Globus Toolkit installation

We use the job description below to illustrate not only job staging but also a range of further job description language features. The submission of this job to the GRAM services causes the following sequence of actions:

- The /bin/echo executable is transferred from the submission machine to the GRAM host file system. The destination location is the HOME directory of the user on behalf of whom the job is executed by the GRAM services (see <fileStageIn>).
- The transferred executable is used to print a test string (see <executable>, <directory> and the <argument> elements) on the standard output, which is redirected to a local file (see <stdout>).
- The standard output file is transferred to the submission machine (see <fileStageOut>).
- The file that was initially transferred during the stage-in phase is removed from the file system of the GRAM installation (see <fileCleanup>).

```
 1 <job>
 2   <executable>my_echo</executable>
 3   <directory>${GLOBUS_USER_HOME}</directory>
 4   <argument>Hello world!</argument>
 5   <stdout>${GLOBUS_USER_HOME}/stdout</stdout>
 6   <stderr>${GLOBUS_USER_HOME}/stderr</stderr>
 7   <fileStageIn>
 8     <transfer>
 9       <sourceUrl>gsiftp://submit.host:2888/bin/echo</sourceUrl>
10       <destinationUrl>file:///${GLOBUS_USER_HOME}/my_echo</destinationUrl>
11     </transfer>
```

```
12   </fileStageIn>
13   <fileStageOut>
14     <transfer>
15       <sourceUrl>file://${GLOBUS_USER_HOME}/stdout</sourceUrl>
16       <destinationUrl>gsiftp://submit.host:2888/tmp/stdout</destinationUrl>
17     </transfer>
18   </fileStageOut>
19   <fileCleanUp>
20     <deletion> <file>file://${GLOBUS_USER_HOME}/my_echo</file> </deletion>
21   </fileCleanUp>
22 </job>
```

In more detail, and referring to line numbers:

2. The executable is /bin/echo
3. The working directory is that specified by $GLOBUS_USER_HOME.
4. A single argument is provided, the string “Hello World!”
5. Standard output is to be written to the file ${GLOBUS_USER_HOME}/stdout
6. Standard error is to be written to the file ${GLOBUS_USER_HOME}/stderr

7-12. A single file is to be staged in, namely the executable.

13-18. A single file is to be staged out, namely the standard output.

20. The executable should be deleted when execution is complete.

4.3.5 Credential Delegation

A job submitted to a GRAM server may require delegated user credentials if it is going either to (a) perform additional remote operations, such as further GRAM job submissions or other Web service calls, in which case it requires a *job credential*; or (b) engage in staging operations, in which it requires a *staging credential*. GT4 allows credentials to be supplied in two different ways. In the first approach, the user passes the required credential(s) with the job submit request. These credentials are then available for use by the job and are discarded following use. In the second, potentially more efficient, approach, the user uploads the credential(s) to a Delegation service associated with the GRAM server, so that they can be accessed directly by subsequent job requests.

The GT4 job description language and command line tools support these two modes of use as follows:

- A client can use the command line client program globus-credential-delegate to delegate one or both credentials to a GRAM server. This command returns an EPR to the newly delegated credential. (A client can also use globus-credential-refresh to renew a delegated credential.)
- Having delegated credentials in this way, the client can then communicate that they are to be used in subsequent request by providing such EPR(s) in a job description, via the jobCredential, stagingCredential, and/or transferCredential elements (Table 5).
- Alternatively, the client always has the option of using the -job-delegate and -staging-delegate flags to globusrun-ws to request that credentials be transferred for a single job.

The flags mentioned in the third bullet are interpreted as follows:

- -job-delegate: If supplied *and* the job description does not already provide a jobCredential element, globusrun-ws delegates the client credential to GRAM and introduces the corresponding element to the submitted job description.
- -staging-delegate: If supplied *and* the job description includes staging or cleanup directives *and* the job description does not already provide the necessary stagingCredential or transferCredential element(s), globusrun-ws delegates the client credential to GRAM (and thus to RFT), and introduces the corresponding elements to the submitted job description.

4.3.6 The Primary globusrun-ws Options, and Other Details

We list in Table 4 the primary globusrun-ws options. The flags -submit -stream request interactive streaming mode (-stream), in which not only status updates but also standard output produced by the remote program is returned to the globusrun-ws client incrementally.

Table 4: The primary globusrun-ws options

Option	Description
-submit	Submit a job in *interactive mode*, meaning globusrun-ws returns only when job finishes or fails; status changes are communicated back to the client; and killing globusrun-ws also kills the job.
-submit -stream	Submit a job in *interactive streaming* mode, which is equivalent to interactive mode except that the job's standard output is made available to the client as the standard output of the globusrun-ws call.
-submit -batch	Submit a job in *batch* mode, which means that the globusrun-ws call returns immediately upon job submission.
-monitor EPR	Attach to a running job. A globusrun-ws -submit -batch followed by a globusrun-ws -monitor is equivalent to an interactive mode submission, i.e., a simple globusrun-ws -submit.
-status EPR	Return the current status of the job at the specified EPR.
-kill EPR	Terminate the job at the specified EPR.
-validate	Verify the valdidity of the globusrun-ws call represented by the other arguments.
-version	Return the version number of this implementation of globusrun-ws.
-help	Return verbose help information
-usage	Return usage information for the various command options.

Modify timeout values. Unreliable computers and networks can cause requests and acknowledgements to be lost or delayed. Thus, globusrun-ws returns with failure if a request to a GRAM server does not return within a timeout period. By default, this period is two minutes; the client can modify it via the -http-timeout argument.

Job resubmission for exactly-once job submission semantics. If a job submission request times out, then the client cannot know whether or not the job was successfully submitted: perhaps the request succeeded, but the acknowledgement was lost. If this is a concern, then the client can obtain (or supply) a submission identifier for a job, and then, if a timeout occurs, resubmit the job with the same identifier. GRAM then ensures that the job is not submitted more than once.

Lifetime management. A GRAM server will destroy a job after its lifetime has expired. By default, a job's lifetime is infinite, which means that the job only terminates if the program that it is executing terminates or if the client or server explicitly terminate the job for some reason. A client can use the -termination option on a submit request to request a specific lifetime.

Parallel jobs. A user can use globusrun-ws to submit a job description with a count attribute greater than the default value of one. Such a description requests that multiple copies of the specified executable be created on the target computer.

Multijobs. GRAM also supports "multijobs," i.e., job descriptions that specify not a request to execute N copies of F, but rather a request to execute different programs: say F, G, and H. (Each of those requests can be itself a parallel job, but not a multijob.) The GT4 documentation provides details on how this feature is used.

Job and process rendezvous. GRAM offers a mechanism to perform synchronization between processes in a multiprocess job and between subjobs in a multijob. The job application can in fact register binary information, for instance process information or subjob information, and get notified when all the other processes or subjobs have registered their own information. This is for instance useful for parallel jobs which need to rendezvous at a "barrier" before proceeding with computations, in the case when no native application API is available to help do the rendezvous.

4.3.7 Further Job Description Language Details

The more complex example in Figure X introduces some additional features that we discuss in the following. First, we present in Table 5 a list of the job description elements that GRAM supports. Those identified by a "Y" in the third column support substitution.

Table 5: Job description elements, with their cardinality (blank=[0..1], *=[0..*]), whether or not they support substitution, and their description.

Element name	C	S	Description
argument	*	Y	A command line argument for the executable. Spaces and quotes within an argument are preserved.
count			Number of executions of the executable. Default = 1 if empty/missing.
directory		Y	The path of the default directory the jobmanager should use for the job.
dryRun			(Unimplemented) If "true" then the jobmanager will not submit the job for execution and will return success.
environment	*	Y	The environment variables that will be defined for the executable in addition to default set that is given to the job by the jobmanager.
executable		Y	The name of the executable file to run on the remote machine.
extensions			Currently used only to specify client-specific data which the client wishes to associate with the job it is controlling.
Factory Endpoint			The Managed Job Factory service endpoint to which this job description should be submitted.
fileCleanUp		Y	A file local to the job that should be removed following job termination via a GridFTP-compatible file server.
fileStageIn		Y	A ("remote URL" "local file") pair that should be staged to the node(s) that will run the job, prior to job execution.

fileStageOut		Y	A ("local file" "remote URL") pair that should be staged from the node(s) running the job following job completion, to a GridFTP-compatible file server.
hostCount			Only applies to clusters of SMP computers, such as Linux clusters. Defines the number of nodes to distribute the "count" processes across.
jobCredential Endpoint			An EPR that points to the delegated credential resource.
jobType			How the jobmanager should start the job: one of mpi, single, multiple, condor.
libraryPath	*	Y	A path to be appended to system-specific library path environment variables.
maxCpuTime			Explicitly set the maximum cputime, in minutes, for a single execution of the executable. If the scheduler cannot set cputime, then an error will be returned.
maxMemory			Explicitly set the maximum amount of memory, in megabytes, for a single execution of the executable. If the GRAM scheduler cannot set maxMemory, then an error will be returned.
maxTime			The maximum walltime or cputime, in minutes, for a single execution of the executable. Walltime or cputime is selected by the GRAM scheduler being interfaced.
maxWallTime			Explicitly set the maximum walltime, in minutes, for a single execution of the executable. If the GRAM scheduler cannot set walltime, then an error will be returned.
minMemory			Explicitly set the minimum amount of memory, in megabytes, for a single execution of the executable. If the GRAM scheduler cannot set minMemory, then an error will be returned.
project			Target the job to be allocated to a project account as defined by the scheduler at the defined (remote) resource.
queue			Target the job to a queue (class) name as defined by the scheduler at the defined (remote) resource.
remoteIoUrl			Writes the given value (a URL base string) to a file, and adds the path to that file to the environment through the GLOBUS_REMOTE_IO_URL environment variable. If this is specified as part of a job restart RSL, the job manager will update the file's contents. This is intended for jobs that want to access files via GASS, but the URL of the GASS server has changed due to a GASS server restart.
Staging Credential Endpoint			An EPR which points to the delegated credential resource used to make remote calls to RFT.
stderr *and*		Y	Each stderr or stdout element specifies a file name or URL in which the job's standard error or standard output, respectively, are to be stored. The standard error (output) from the job is staged after the execution to the specifies path(s) or URL(s). Values that are local

stdout			paths are staged relative to the user's home directory. Values that are absolute paths are staged as if 'file://' were prepended to the path.
stdin		Y	The name of the file to be used as standard input for the executable on the remote machine.

4.4 GRAM Client APIs

The GRAM server's client interface is simple. As illustrated in Figure 9, four portTypes are provided that define operations on four WS-Resources, which represent the state of the GRAM server, the state of a single job, the certificate chain associated with a delegation factory, and a credential, respectively. Those operations include both job- and delegation-specific functions (CreateManagedJob, Release, RequestSecurityToken, Refresh) as well as various WSRF and WS-Notification operations on the WS-Resources. GT4 includes Java, Python, and C bindings for these four portTypes.

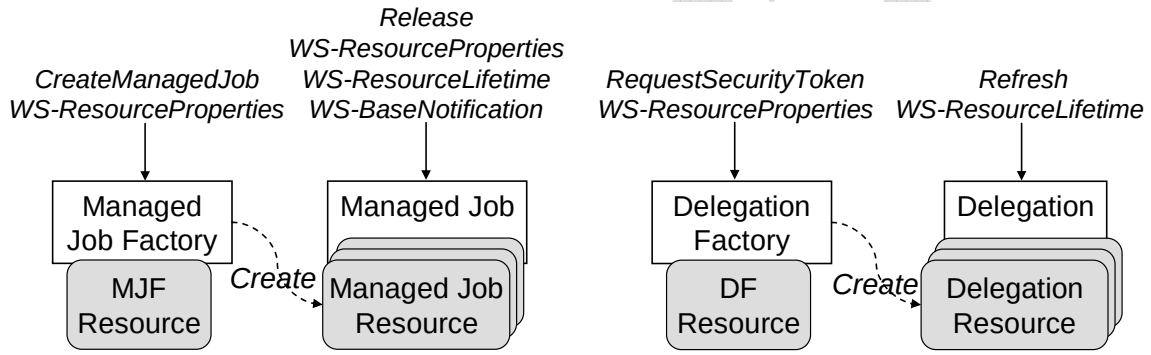

Figure 9: From bottom to top, the four WS-Resources (shaded) associated with GRAM, the four interfaces used to operate on those WS-Resoures, and the operations that those interfaces support.

In more detail, the ManagedJobFactory **Managed Job Factory portType** portType supports WS-ResourceProperties query operations on the managed job factory resource that is used to maintain information about GRAM server state (see Table 6). It also defines a createManagedJob operation, which takes as input a job description, an optional initial termination time for the job resource, and an optional state notification subscription request. Successful execution of this operation results in the creation of a ManagedJob WS-Resource and the return of an EPR for the new ManagedJob. If subscription is requested, then the client is subscribed for notifications and a second EPR is returned for the new subscription. The globusrun-ws command line client uses this operation to implement much of its functionality, generating (or passing along) an appropriate job description and in the case of interactive or streaming interactive mode job submissions, requesting a subscription.

The **Managed Job** portType (MJPT) defines operations on the ManagedJob WS-Resource. It supports WS-ResourceProperties query operations for accessing information about job state (Table 7), WS-ResourceLifetime lifetime management operations for managing job lifetime, and WS-BaseNotification operations to enable clients to request notifications of changes. It also defines an operation release used to release a hold placed on a state through the use of the holdState field in the job description. This operation takes no parameters and returns nothing.

A ManagedJob WS-Resource may represent either an executable or a multijob. Somewhat different resource properties are defined in each case.

The **Delegation Factory** portType supports WS-ResourceProperties query operations on the delegation factory resource that is used to maintain information about the certificate chain associated with the delegation factory. It also defines a RequestSecurityToken operation, which requests the upload of a security token to the delegation service and the return of an EPR to the new WS-Resource.

The **Delegation** portType supports WS-ResourceLifetime operations on a security token WS-Resource. It also defines a refresh operation, which a client can use to upload a new credential.

Table 6: Managed Job Factory Resource Properties

Resource Property	Description
localResourceManager	Local resource manager type: e.g., Condor, Fork, LSF, Multi, PBS.
globusLocation	The location of the Globus Toolkit installation that these services are running under.
hostCPUType	The job host CPU architecture (i686, x86_64, etc...)
hostManufacturer	The host manufacturer name. May be “unknown.”
hostOSName	The host OS name (Linux, Solaris, etc...)
hostOSVersion	The host OS version.
scratchBaseDirectory	The directory recommended by the system administrator to be used for temporary job data.
delegationFactoryEndpoint	The endpoint reference to the delegation factory used to delegate credentials to the job.
stagingDelegationFactoryEndpoint	The endpoint reference to the delegation factory used to delegate credentials to the staging service (RFT).
condorArchitecture	Condor architecture label (for Condor schedulers).
condorOS	Condor OS label (for Condor schedulers).
GLUECE	GLUE data (data in GLUE schema format [18]).
GLUECESummary	GLUE data summary.

Table 7: Managed Job Resource Properties and those additional Resource Properties associated with Managed Executable Job and Managed Multi Job Resources

Resource Property	Description
serviceLevelAgreement	A wrapper around fields containing the single-job and multi-job descriptions or RSLs. Only one sub-field shall have a non-null value.
state	The current state of the job.
fault	The fault (if generated) indicating the reason the job did not complete.
localUserId	The job owner’s local user account name.
userSubject	The job owner’s GSI certificate DN.
holding	Indicates whether a hold has been placed on this job.

Managed Executable Job Resource only	
stdoutURL	A GridFTP URL to the file generated by the job that contains the stdout.
stderrURL	A GridFTP URL to the file generated by the job that contains the stderr.
credentialPath	The path (relative to the job process) to the file containing the user proxy used by the job to authenticate to other services.
exitCode	The exit code generated by the job process.
Managed Multi Job Resource only	
subJobEndpoint	A set of endpoint references to the sub-jobs created by this multi-job

4.5 GRAM Configuration and Administration

We install a GRAM server to enable Web services access to a compute resource. Installation involves two principal configuration steps. Defaults are defined in both cases.

- *Local scheduler interface*. GRAM depends on a local mechanism for starting and controlling jobs. If the fork-based GRAM mode is to be used, no special software is required. For batch scheduling mechanisms, such as Condor, SGE, PBS, and LSF, the local scheduler must be installed and configured for local job submission prior to deploying and operating GRAM.
- *Authorization to user mapping*. GRAM depends on an authorization callout to determine both whether a specific request should be allowed and the local user as which an authorized request should be executed. By default, GRAM looks for the file `/etc/grid-security/grid-mapfile`, which is assumed to contain a set of one-line entries, each specifying the DN of an authorized user and the local user as which they are to execute. For example:

  ```
  "/O=Grid/OU=GlobusTest/OU=simpleCA-foo.bar.com/OU=bar.com/CN=John" john
  ```

  GRAM can also be configured to access other sources for authorization and mapping information: see §XX.

4.6 Related Software and Tools

Table XX …

4.6.1 DAGman, Condor-G, and GriPhyN Virtual Data System

You have a large number of jobs to run.

Difficulty is managing them all and the inevitable failures: what has run, where is your output?

Condor-G

4.6.2 Nimrod-G: Managing Parameter Studies on the Grid

A particularly simple class of parallel …

XX.

4.6.3 MPICH-G2: Message Passing on the Grid

XX.

4.6.4 Ninf-G: Remote Procedure Call on the Grid

XX.

4.7 Case Studies

How many? What? Here or attached to earlier sections?

4.7.1 Execution Management Case Study 1

XX

4.7.2 Execution Management Case Study 2

XX

4.8 How GRAM Works

We review briefly the GRAM implementation. This information should be of interest as background to the discussion of GRAM configuration (§4.5) and also to users who want some insight into how GRAM works and to developers as an example of a WSRF/WSN-based service.

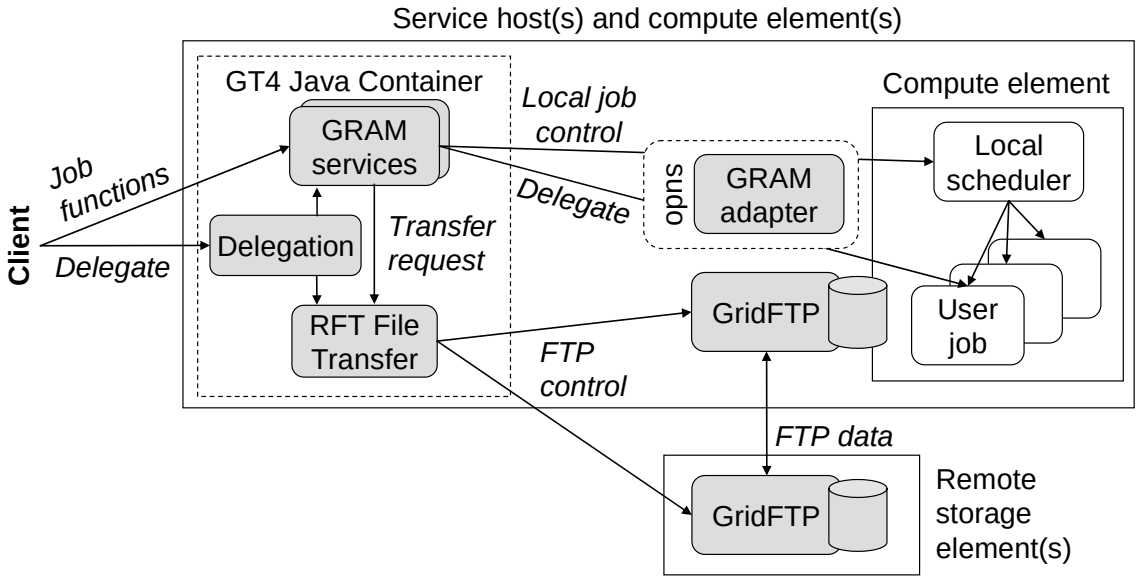

Figure 10: GRAM implementation structure

As illustrated in Figure 10, the primary elements of a GRAM deployment are as follows:

- A set of services running in a GT4 Java container, as follows:
  - GRAM-specific services for creating, monitoring, and managing jobs
  - A general-purpose delegation service, used to manage delegated credentials

    - A general-purpose reliable file transfer (RFT) service, used to manage data staging operations.

- A scheduler-specific GRAM adapter, used to map GRAM requests into appropriate requests to a local scheduler.
- A GridFTP server used to execute data staging commands.

The client view of this ensemble is precisely those interfaces defined by the GRAM and Delegation services. As discussed in §4.3, those interfaces allow the user to submit, monitor, and manage jobs, and to install and refresh delegated credentials.

4.8.1 Use of WS-Resource Mechanisms

The Web services within the GT4 Java container use WS-Resources to represent state associated with jobs ("ManagedJobs"), delegated credentials, and transfers in progress. This representation allows for the use of WSRF and WS-Notification mechanisms to access state, manage lifetime, perform notification, and so forth. This reuse of existing machinery, and the use of general-purpose Delegation and RFT services, means that relatively little GRAM-specific code had to be developed for the GRAM implementation.

The state associated with each submitted job is just the WS-Resource used to represents its current state: perhaps a few thousands of bytes, depending on the size of job arguments. Thus, GRAM can in principle scale to extremely large numbers of submitted jobs. In practice, features of current Java hosting environments can limit scalability: see §12.3.

4.8.2 Security Issues

An incoming job management request is subject to multiple levels of security checks:

1. WS-Security mechanisms are used to validate the credentials associated with the request and thus to authenticate the requestor.
2. Authorization is performed via an authorization callout. Depending on configuration, this callout may consult a gridmapfile, a SAML server, or other mechanism.
3. If authorization succeeds, then it also yields the local identity under which the job is to execute. The Unix utility sudo is used to invoke local resource management mechanisms under this user id, thus enabling the application of site authorization mechanisms.

GRAM has been designed to minimize the privileges required and to minimize the risks of service malfunction or compromise. Services operating in the GT4 Java container do not require special privileges to perform their operations: privileged operations are performed exclusively via sudo functions.

To protect users from each other, jobs submitted by different users are typically executed in separate local security contexts: e.g., under specific Unix user IDs based on details of the job request and authorization policies. (Dynamic account management mechanisms can be used to generate such security contexts in an on-demand manner: see §4.9.)

To assist with normal accounting functions as well as to further mitigate risks from abuse or malfunction, GRAM uses a range of audit and logging techniques to record a history of job submissions and critical system operations.

4.8.3 Data Operations

GRAM services hand off data staging operations to the RFT service that is associated with any GRAM deployment. This strategy permits reuse of the general-purpose RFT code and also means that job submissions that do not require staging do not incur any data management costs. Upon receiving a data staging request, the RFT service initiates a GridFTP transfer between the specified source and destination.

In addition to conventional data staging operations, GRAM supports a mechanism for incrementally transferring output file contents from the computation resource while the job is running. This mechanism (incorporated in GridFTP) allow arbitrary numbers of files to be transferred in this fashion.

4.9 Execution Management Futures

XX

4.10 Further Reading

XX

Chapter 5 Data Management

Data management tools are concerned with the location, transfer, and management of distributed data. GT4 provides various basic tools, including GridFTP for high-performance and reliable data transport, RFT for managing multiple transfers, RLS for maintaining location information for replicated files, and OGSA-DAI for accessing and integrated structured and semistructured data.

Associated tools enhance GT4 components by addressing storage reservation (NeST), providing a command-line client for GridFTP (UberFTP), providing a uniform interface to distributed data (SRB), and supporting distributed data processing pipelines (DataCutter, STORM).

Other interesting systems include the LBNL Storage Resource Manager (SRM), which provides for GSI-authenticated GridFTP access to managed storage, and GridNFS, a GSI-enabled implementation of NSFv4 being developed at U.Michigan.

See also: GriPhyN VDS (Execution).

Table 8: Globus and related data management tools

Name	Purpose	M	G	
GridFTP server	*Enhanced FTP server supporting GSI authentication and high-performance throughput. Iinterfaces to Unix POSIX, HPSS, GFPS, and Unitree provided; others can be developed.*	*3*	*I*	*5.1*
globus-url-copy	*Non-interactive command-line client for GridFTP.*	*3*	*I*	
Replica Location Service	*RLS is a decentralized service for registering and discovering information about replicated files.*	*3*	*I*	
Reliable File Transfer service	*RFT controls and monitors third-party, multi-file transfers using GridFTP. Features exponential back-off on failure, all or none transfers of multi-file sets, optional use of parallel streams and TCP buffer size tuning, and recursive directory transfer.*	*1*	*I*	
Lightweight Data Replicator	*LDR is a tool for replicating data to a set of sites. It builds on GridFTP, RLS, and pyGlobus.*	*1*	*P*	
OGSA Data Access & Integration	*OGSA-DAI is an extensible framework for accessing and integrating data resources, including relational and XML databases and semistructured files.*	*2*	*P*	
Network Storage [4]	NeST allows GridFTP clients to negotiate reservations for disk space, which then apply to subsequent transfers.	1	Y	
UberFTP [27]	Interactive command-line client for GridFTP.	3	Y	
Storage Resource Broker [12, 13]	Client-server middleware that provides a uniform interface for connecting to heterogeneous, distributed data resources. GSI authentication and GridFTP transport.	3	?	
DataCutter & STORM	DataCutter supports processing of large datasets via the execution of distributed pipelines of application-specific processing modules; STORM supports relational data.	2	Y ?	

5.1 GridFTP

XX.

5.2 Reliable File Transfer Service

XX.

5.3 Replica Location Service

The Globus Toolkit's replica location service (RLS)

RLS does not currently support a Web services interface.

XX

5.4 Data Access and Integration

XX.

5.5 Related Software and Tools

XX.

5.6 Case Studies

XX.

5.6.1 Data Case Study 1

XX

5.6.2 Data Case Study 2

XX

5.7 How it Works

XX.

5.8 Further Reading

The article "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets" [26] is a somewhat dated introduction to topics relating to data management.

The book chapter "Data Access, Integration, and Management" [37] provides a comprehensive introduction to topics discussed here.

Chapter 6 Monitoring and Discovery

The unexamined life is not worth living – Socrates

Monitoring and discovery mechanisms are concerned with obtaining, distributing, indexing, archiving, and otherwise processing information about the configuration and state of services and resources. In some cases, the motivation for collecting this information is to enable discovery of services or resources; in other cases, it is to enable monitoring of system status.

GT4's support in its Java, C, and Python WS Core for WSRF and WS-Notification interfaces provides useful building blocks for monitoring and discovery, enabling the definition of properties for which monitoring and discovery is be provided, and subsequent pull- and push-mode access. GT4 services such as GRAM and RFT define appropriate resource properties, providing a basis for service discovery and monitoring. Other GT4 services are designed to enable discovery and monitoring, providing for indexing (MDS-Index), archiving (MDS-Archive), and analysis of data for significant events (MDS-Trigger). Every GT4 container incorporates a built-in MDS-Index service for discovery of services within the container; these MDS-Index services can be linked to build Grid-wide indices.

Associated tools include those that monitor individual entities (e.g., Ganglia and Hawkeye for clusters, NetLogger for individual components) or that coordinate the invocation of tests at multiple locations (Inca, Nagios).

See also: WebMDS (interface).

Table 9: Globus and related monitoring and discovery tools

Name	Purpose	M	G
Java, C, and Python WS Cores	*Implements WSRF and WS-Notification specifications, thus allowing Web services to define, and allow access to, resource properties. Container incorporates a local index, enabling discovery of services.*	*3*	*T*
MDS-Index	*Collects live monitoring information from services and enable queries against that information.*	*2*	*T*
MDS-Trigger	*Compares live monitoring information against rules to detect fault conditions, and notifies operators (for example, by email)*	*2*	*T*
MDS-Archive	*Store historical monitoring data data and enable queries against that data.*	*2*	*T*
Aggregator framework	*The aggregator framework facilitates the building of aggregating services (for example the index, trigger and archive services).*	*2*	*T*
Hawkeye	Monitor individual clusters, using Condor as a base. GT4 includes data provider that makes status information available in GLUE schema.	3	Y
Ganglia [24, 43]	Monitor individual clusters and sets of clusters. GT4 includes data provider that makes status information available in GLUE schema.	3	Y
Nagios [2]	A widely used open source system for monitoring networks, computers, and services in distributed systems.	3	Y

Inca [7]	Monitor services in a distributed system by performing a set of specified tests at specified intervals; publish results of these tests.	2	N
NetLogger	Generate, collect, and analyze high frequency data from distributed system components.	3	N

6.1 MDS4

The Monitoring and Discovery System (MDS4) component of GT4 can streamline the tasks of monitoring and discovering services and resources in a distributed system.

Monitoring is the process of observing resources or services (e.g., computers and schedulers), for such purposes as fixing problems and tracking usage. A user might use a monitoring system to identify resources that are running low on disk space, in order to take corrective action.

Discovery is the process of finding a suitable resource to perform a task: for example, finding a compute host on which to run a job. This process may involve both finding which resources are suitable (e.g., have the correct CPU architecture) and choosing a suitable member from that set (e.g., the one with the shortest submission queue).

Both monitoring and discovery applications require the ability to collect information from multiple, perhaps distributed, information sources. To meet this need, MDS4 provides so-called *aggregator services* that collect recent state information from registered *information sources*; and browser-based interfaces, command line tools, and Web service interfaces that allow users to query and access the collected information.

MDS4 provides three different aggregator services with different interfaces and behaviors (although all built on a common framework): *MDS-Index*, which supports Xpath queries on the latest values obtained from the information sources; *MDS-Trigger*, which performs user-specified actions (such as send email or generate a log-file entry) whenever collected information matches user determined criteria; and *MDS-Archiver*, which stores information source values in a persistent database that a client can then query for historical information.

MDS4 makes heavy use of XML and Web service interfaces to simplify the tasks of registering information sources and locating and accessing information of interest. In particular, all information collected by aggregator services is maintained as XML, and can be queried via Xpath queries (as well as other Web services mechanisms).

6.1.1 Aggregators and Information Sources

The key to understanding MDS4 is the aggregator-information source framework. The basic ideas are as follows:

- *Information sources* for which discovery or access is required are explicitly registered with an *aggregator service*.
- Registrations have a lifetime: if not renewed periodically, they expire. (Thus, an aggregator is self-cleaning: outdated entries disappear automatically when they cease to renew their registrations.)
- The aggregator periodically collects up-to-date state or status information from all registered information sources, by using an information-source-specific access mechanism.
- The aggregator then makes all information obtained from registered information sources available via an aggregator-specific Web services interface.

MDS4 aggregators are distinguished from a traditional static registry such as UDDI by their soft-state registration of information sources and periodic refresh of the information source values that they store. This dynamic behavior enables scalable discovery, by allowing users to access "recent" information without accessing the information sources directly.

6.1.2 Information Sources and Registration

An information source can be essentially any entity from which an aggregator service can obtain information: for example, a file, a program, a Web service, or another network-enabled service.

As noted above, information sources must be registered periodically with any aggregator service that is to provide access to its data values. Registration is performed via a Web service (WS-ServiceGroup) Add operation. Two registration modes are supported; each also defines the mechanism to be used to access the associated information source.

The more general registration mode allows information to be obtained from an arbitrary source. In this mode, an information source is registered by providing a user-supplied program that is run periodically to obtain up-to-date data. This user program can either generate the information locally or use a source-specific protocol to access the information remotely. The program must convert non-XML data into an appropriate XML representation.

A more streamlined form of registration is supported for WSRF-compliant Web services. Such services need simply to make status and state information available as WSRF resource properties. At registration time, the user specifies whether the aggregator should use either *pull* resource properties, using the WSRF "get resource property" interface, or to subscribe to resource property changes so that values are *pushed* via WS-Notification subscription methods.

6.1.3 The Three Types of Aggregator

The *MDS-Index* service makes data collected from information sources available as XML documents. More specifically, the data is maintained as WSRF resource properties. Thus:

- Users can write their own applications that collect information using standard *Web services interfaces*, namely the WSRF get-property and WS-Notification subscribe operations, for which GT4 provides C, Java, and Python APIs.
- The command line tool *wsrf-get-property* can be used to retrieve resource properties, with the desired resource property specified via an XPath expression.
- A tool called *WebMDS* presents MDS-Index information in a standard web browser. WebMDS is highly configurable, using XLST transformations to describe how MDS-Index resource properties are converted to HTML. Standard transformations included in GT4 provide an interface that displays overview information, with hyperlinks giving the ability to drill down and view more detailed information about each monitored resource.

The *MDS-Trigger* service defines a Web service interface that allows a client to register an Xpath query and a program to be executed whenever a new value matches a user-supplied matching rule.

The *MDS-Archive* service stores all values received from information sources in persistent storage. Client requests can then specify a time range for which data values are required.

6.1.4 Built-In Information Sources and MDS-Index Services

Every GT4 Web services container includes a default MDS-Index service with which any GT4 services running in that container (e.g., GRAM, CAS, RFT) are automatically registered. Thus, each installation on a platform has an index that allows one to discover what services are available.

In addition, Grids often need to keep track of all available WS-Resources. To accommodate this common case, GT4 also provides a simple method for specifying one or more default indexes to be a Grid-wide MDS-Index so that each WS-Resource registered to a default MDS-Index is also registered in the Grid MDS-Index.

GT4 is configured to use MDS4 mechanisms to good effect itself, for discovery and monitoring of GT4 services. Every GT4 Web service supports a minimal set of resource properties (an informal service name and a service startup time) and thus can be registered easily into one or more aggregators for monitoring and discovery. Further, two GT4 Web services, GRAM and RFT, also publish a larger set of service-specific information, as described in the documentation for the service in question. Finally, the GT4 distribution also includes information source executables to enable registering GridFTP and RLS into aggregators.

6.1.5 MDS4 and MDS2 Compared

MDS4 has similar features to, but does not interoperate with, previous versions of MDS (MDS2 and MDS3). Important differences include a more powerful query language (Xpath instead of LDAP); a simpler and therefore more robust implementation, due to fewer components; simpler (indeed, in simple cases, totally automated) configuration, due to fewer components and tight integration with the GT4 implementation; convenient interface to arbitrary information sources due to extensible architecture; no requirement for pre-defined schema in information providers; and lower performance due to use of XML protocols.

6.2 Related Software and Tools

6.3 Case Studies

6.3.1 Earth System Grid Monitor

6.3.2 Example 2

6.4 How it Works

6.5 Further Reading

The article “Grid Information Services for Distributed Resource Sharing” [72], while somewhat dated, still provides a good introduction to some key architectural concepts adopted in MDS4.

The article “A Performance Study of Monitoring and Information Services for Distributed Systems” [39] introduces important performance issues that arise in Grid information services, and quantifies the impact of such techniques as caching.

Chapter 7 Security

Security tools are concerned with establishing the identity of users or services (authentication), protecting communications, and determining who is allowed to perform what actions (authorization), as well as with supporting functions such as managing user credentials and maintaining group membership information.

GT4 provides distinct WS and pre-WS authentication and authorization capabilities. Both build on the same base, namely standard X.509 end entity certificates and proxy certificates, which are used to identify persistent entities such as users and servers and to support the temporary delegation of privileges to other entities, respectively.

GT4's WS security [78] comprises (a) *Message-Level Security* mechanisms, which implement the WS-Security standard and the WS-SecureConversation specification to provide message protection for GT4's SOAP messages, (b) *Transport-Level Security* mechanisms, which uses transport-level security (TLS) mechanisms; and (c) an *Authorization Framework* that allows for a variety of authorization schemes, including a "grid-mapfile" access control list, an access control list defined by a service, a custom authorization handler, and access to an authorization service via the SAML protocol. For non-WS components, GT4 provides similar authentication, delegation, and authorization mechanisms, although with fewer authorization options.

Associated security tools provide variously for the storage of X.509 credentials (MyProxy [3] and Delegation services), the mapping between GSI and other authentication mechanisms (e.g., KX509 and PKINIT for Kerberos, MyProxy for one-time passwords), and maintenance of information used for authorization (VOMS, GUMS, PERMIS [68]). Likely also to be of interest in the future is work underway in the UK and US to integrate Grid security with Shibboleth [36, 42], enabling access to campus directories.

See also: PURSe (Interface).

Table 10: Globus and related security tools

Name	Purpose	M	G
Message-Level Security	*Implements WS-Security standard and WS-SecureConversation specification to provide message protection for SOAP messages.*	*3*	*T*
Authorization Framework	*Allows for a variety of authorization schemes, including file- and service-based access control lists, custom handles, SAML protocol.*	*3*	*T*
Pre-WS A&A	*Authentication, delegation, authorization for non-WS components.*	*3*	*T*
Delegation Service	*Enable storage and subsequent (authorized) retrieval of proxy credentials, thus enabling delegation when using WS protocols.*		*T*
Community Authorization Service	*Issues assertions to users granting fine-grained access rights to resources. Servers recognize and enforce the assertions. CAS is currently supported by the GridFTP server.*	*2*	*T*
Simple CA	*A simplified certification authority for issuing X.509 credentials.*	*3*	*T*
MyProxy [76] service	*Allow federation of X509 and other authentication mechanisms (e.g., username/password, one-time passwords) via SASL/PAM.*	*3*	*T*
GSI-OpenSSH	*Version of OpenSSH that supports GSI authentication. Provides*	*3*	*C*

	remote terminal (SSH), file copy (SCP), and FTP (SFTP) functions.		
VOMS	Database of user roles and capabilities, and user client interface that supports retrieval of attribute certificates for presentation to VOMS-enabled services.	2	Y
VOX & VOMRS	Extends VOMS to provide Web registration capabilities, rather like PURSe.	2	Y
PERMIS [68]	Authorization service accessible via SAML protocol.	2	Y
GUMS	Grid User Management System: an alternative to grid map files.		
KX509 & KCA	KX509 is a "Kerberized" client that generates and stores proxy credentials, so users authenticated via Kerberos can access the Grid; KCA is a Kerberized certification authority used to support KX509.	3	
PKINIT	A service that allows users with Grid credentials to authenticate to a Kerberos domain.	3	
Shibboleth [36]	XXX		

7.1 Security Principles

XX.

7.2 Supporting Infrastructure

grid-proxy-init, etc. XX.

7.3 Web Services Authentication and Authorization

XX.

7.3.1 Community Authorization Service

XX.

7.3.2 Delegation Service

XX.

7.3.3 Authorization Framework

XX.

7.3.4 Message/Transport-level Security

XX.

7.4 Credential Services

GT4 provides two components relating to credential management: MyProxy, an online credential repository, and SimpleCA, a package for generating credentials. (SimpleCA is *not* a service, but a program that you install and run to generate credentials..)

7.4.1 MyProxy

MyProxy is an online credential repository. While this may sound daunting, the concept (and practice) is straightforward: it's a service into which you can store X.509 proxy credentials, protected by a passphrase, for later retrieval over the network. Storing credentials in a MyProxy repository eliminates the need for manually copying private key and certificate files between machines. A credential stored in MyProxy can also be accessed at times when the user's credential would not otherwise be accessible: for example, when the user wants to authenticate to a grid portal from a Web browser, or if a job manager wants to renew the user's credential and the user is not available.

MyProxy does not currently provide a Web services interface, although it will eventually.

XX.

New developments: OTP support.

7.4.2 SimpleCA

The SimpleCA package provides a simple certification authority that a user can install and use to issue credentials to Globus Toolkit users and services. This package is meant for use in situations where the user wants public key credentials, for example in order to test GT's operation, but does not have access to a proper certification authority.

7.5 GSI-OpenSSH

GSI-OpenSSH is a modified version of the OpenSSH secure shell server that adds support for X.509 proxy certificate authentication and delegation, providing a single sign-on remote login and file transfer service. GSI-OpenSSH can be used to login to remote systems and transfer files between systems without entering a password, relying instead on a valid proxy credential for authentication. GSI-OpenSSH forwards proxy credentials to the remote system on login, so commands requiring proxy credentials (including GSI-OpenSSH commands) can be used on the remote system without the need to manually create a new proxy credential on that system.

The GSI-OpenSSH distribution provides gsissh, gsiscp, and gsiftp clients that function equivalently to ssh (secure shell), scp (secure copy), and sftp (secure FTP) clients except for the addition of X.509 authentication and delegation.

7.6 Related Software and Tools

Not clear that a list is what is wanted here, but rather some details on the whole set.

Mention integration with VOMS?

7.7 Case Studies

XX.

7.7.1 The Earth System Grid Portal

XX.

7.7.2 A Second Example

XX.

7.8 How it Works

Details of PKI, etc.

7.9 Further Reading

Chapter 8 of the book *Handbook of Applied Cryptography* [42] deals exclusively with public key cryptography.

The article "A Security Architecture for Computational Grids" [66], while somewhat dated, still provides a good introduction to some key architectural concepts adopted in Globus security.

The article "Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective" [52] provides an excellent introduction to GT4's implementation of security standards.

The wonderful book *Security Engineering* [3] has not direct connection to Grid security, but is an absolute must read for anyone considering building secure systems.

Chapter 8 User Interfaces

Grid portal and user interface tools support the construction of graphical user interfaces for invoking, monitoring, and/or managing activities involving Grid resources.

Many (but not all) of these tools are concerned with enabling access to Grid systems from Web browsers. Many (but not all) of such Web browser-oriented systems are based on a three-tier architecture, in which a middle-tier *portal server* (e.g., uPortal with Tomcat, or GridSphere) hosts JSR 168-compliant *portlets* [22] that both (a) generate the various elements of the first-tier Web interface with which users interact and (b) interact with third-tier Grid resources and services.

Table 11: GT4 and related user interface tools

Name	Purpose	E	G
Java CoG Desktop	*Java application that provides a "desktop" interface to a Grid, so that for example a job is run, and a file copied, by dragging and dropping its description to a computer and storage system, respectively.*	*1*	*Y*
WebMDS	*Uses XSLT to generate custom displays of monitoring data, whether from active services or archives.*	*1*	*Y*
Portal User Registration Service	*PURSe provides for the Web-based registration of users and the subsequent generation and management of their GSI credentials, thus allowing easy access to Grid resources by large user communities.*	*1*	*Y*
Open Grid Computing Environment	OGCE packages a range of components, including JSR 158-compliant portlets for proxy management, remote command execution, remote file management, and GPIR-based information services.	3	Y
GridPort	??	3	?
Sakai	JSR 168-compatible system for distributed learning and collaborative work, with tools for chat, shared documents, etc.	2	?

Chapter 9 Packaging and Distribution

Name	Purpose	M	G
Grid Packaging Toolkit	*GPT is an RPM-like tool providing relocatability, and multi-flavor (e.g., threaded vs. non-threaded) support.*	*2*	*T*
NSF Middleware Initiative	The NMI software release of Globus bundled with other useful Grid components for the general user community.	3	?
Virtual Data Toolkit	Based on NMI, VDT packages a variety of tools for data-intensive applications.	3	?
Rocks “Grid Roll”	Uses the NSF Middleware Initiative (NMI) Release to provide Globus connectivity for Rocks clusters.	2	N
PACman	A package manager that allows transparent fetch, install, and management of software packages.	2	?
Cluster on Demand	XXX		

Chapter 10 Miscellaneous but Important Tools

As the title of this chapter implies, we describe here a set of unrelated tools that each provides considerable valuable functionality in its specific domain.

Name	Purpose	E	G
eXtensible I/O library	*XIO is a pluggable communication library enabling convenient access to a variety of underlying transport protocols.*	*3*	*T*
Grid TeleControl Protocol service	*GTCP enables secure and reliable telecontrol of remote instrumentation.*	*2*	*T*
Handle System	Scalable and secure directory service.	3	Y
GT4IDE	Eclipse-based Interactive Development Environment for GT4.	1	Y

10.1 The eXtensible I/O Library

The GT4 eXtensible I/O (XIO) library [17] is used within various GT4 components, particularly GridFTP, to implement file I/O and communication functions. It should also be of interest to developers of other similar systems.

XIO provides a single POSIX-like API (open/close/read/write) that supports multiple wire protocols, with protocol implementations encapsulated as drivers. This structure facilitates the

The XIO drivers distributed with GT4 are listed in Table 12.

Table 12: XIO drivers included in GT4

Driver Name	Description
TCP	
UDP	
File	
HTTP	
GSI	
GSSAPI_FTP	
Telnet	
Queuing	

In addition, Globus XIO provides a driver development interface for use by protocol developers. This interface allows the developer to concentrate on writing protocol code rather than infrastructure, as XIO provides a framework for error handling, asynchronous message delivery, timeouts, etc.

The XIO driver-based approach maximizes the reuse of code by supporting the notion of a driver stack. XIO drivers can be written as atomic units and stacked on top of one another. This modularization provides maximum flexibility and simplifies the design and evaluation of individual protocols.

10.2 Grid TeleControl Protocol

XX.

10.3 Handle System

The Handle system [21] is a scalable and secure directory service developed and supported by the Corporation for National Research Initiatives (CNRI). In brief, Handle allows for the distributed definition and maintenance of a hierarchical name space, the binding of arbitrary key-value pairs to names in that name space, and client workloads with a read-mostly, write-not-too-often character. Security is incorporated in a nice way, so that users can control their own part(s) of the name space. CNRI runs root servers, and any other organization can run subsidiary servers. The Handle server software supports replication and failover. The various servers route queries and perform caching. The system is being deployed on a large scale in various interesting contexts.

The Handle system has recently been released as open source software, and a number of interesting ideas are being developed for integration with GT4: for example, recording of resource properties in the directory, name resolution for services whose address may change, and as a basis for WS-ServiceGroup implementations.

10.4 GT4IDE

Chapter 11 Designing and Building Applications

Intro text XX.

11.1 The Grid Development Process

The development of a Grid system involves four interrelated tasks: *design* of the overall system, *deployment* of required services, development of the end-user *application*(s) that use those services, and ongoing *operation*s.

11.1.1 Design

The *design* task involves identifying the resources, services, and end-user application components that must be assembled to meet user requirements. Key issues include identifying the target user community and their required capabilities; quantifying quality of service requirements associated with those capabilities, including the security policies that will govern access; and determining the resources and services that will be used to deliver those capabilities with desired qualities of service.

GT4 provides no specific tools to assist with Grid design, but the many GT Grid deployments and applications provide a considerable body of relevant experience. Thus, for example, if our goal is to federate large numbers of computers for data-intensive computation, we can refer to systems such as Open Science Grid [5] and EU DataGrid; if our goal is to enable community access to large quantities of data, scientific simulation codes, or scientific instrumentation, we can study Earth System Grid [54], Fusion Collaboratory [45], and NEESgrid [61]. We refer to these and other examples below.

11.1.2 Deployment

The *deployment* task establishes the persistent infrastructure that will support our users and their applications. This infrastructure may comprise physical resources such as computers, databases, storage systems, networks, and scientific instruments; services that enable secure and reliable remote access to those resources; and other services that manage resources and services and implement the policies that govern their use. Depending on context, we may have to install and configure all of these services ourselves, or alternatively may be able to negotiate access to deployments provided and operated by groups such as TeraGrid, Open Science Grid, or EGEE.

GT4 and associated tools provide a rich set of functionality for creating the services used in Grid deployments. These tools include both core GT4 components such as the GRAM resource management service for access to computers, the GridFTP and OGSA-DAI services for providing access to data, and index services for monitoring and discovery—and systems that implement various programming models.

11.1.3 Application

The *application* task builds on deployed Grid infrastructure services to deliver required end-user capabilities. Issues here may involve constructing end-user graphical interfaces and defining, initiating, monitoring, and controlling workflows that access Grid infrastructure services. Here, GT4 and associated tools provides a wide range of tools that support a variety of different programming models, from loosely coupled parameter studies to tightly coupled parallel programs, and from data analysis to data distribution and inter-personal collaboration.

11.1.4 Operations

The *operations* task sustains the underlying Grid infrastructure, manages the evolution of policy elements such as group membership and access control lists, and detects, troubleshoots, and recovers from various forms of failure. GT4 and associated tools provide fewer tools for operations than for other aspects of Grid development, but several relevant tools are listed below.

11.2 How Do I?

Here we present a set of recipes, decision trees, and/or case studies illustrating how to apply GT and related technologies to address a range of problems.

XX.

11.3 Who's Using GT

XX.

11.4 Further Reading

XX.

Chapter 12 GT4 Practicalities

The difference between theory and practice is much greater in practice than in theory – Anonymous

We describe a set of practical issues relating to the use of GT4.

12.1 Migrating to GT4

XX.

12.2 Testing

GT4 components have been tested on a range of Unix and Linux platforms. The Java WS-Core and authentication and authorization components have been tested on Windows 2000.

12.3 Performance

12.4 Reporting Problems

XX.

12.5 Further Reading

XX.

Chapter 13 Future Directions

Prediction is difficult; especially about the future – Yogi Berra

Roadmap XX. Or rather, things we'd like to see happen.

Sections for different areas, perhaps?

SRM.

Provisioning.

Security.

VOs.

13.1 Further Reading

XX.

Chapter 14 Historical Notes

"*He who does not learn from history is doomed to repeat it*." (George Santayana)

A one-page summary: Internet, metacomputing, Grid, anatomy, Grid services.

Lessons learned.

CORBA.

The development of Grids has been spurred and enabled by the staged development of increasingly sophisticated and broadly used technologies. As illustrated in Figure 11, early experiments with "metacomputing" (e.g., [34, 35, 57, 67, 70]) worked primarily with custom tools or specialized middleware that emphasized message-oriented communication between computing nodes.

The mid-1990s saw the introduction of middleware designed to support diverse online processing and data-intensive applications. Systems such as the Storage Resource Broker [40], Globus Toolkit® [26], Condor [48, 55], and Legion [65] were developed primarily for scientific applications and demonstrated at various levels of scale on a range of applications. Other developers attempted to leverage the middleware structure being developed for the World Wide Web by using HTTP servers or Web browsers as computing platforms [25, 58, 75]. However, these systems did not gain significant use, partly because the middleware requirements for distributed information systems such as the Web are different from those for Grid applications.

14.1 Globus Toolkit

By 1998, the open source Globus Toolkit (GT) [33] had emerged as a de facto standard software infrastructure for Grid computing. GT2 defined and implemented protocols, APIs, and services used in hundreds of Grid deployments worldwide. By providing solutions to common problems such as authentication, resource discovery, resource access, and data movement, GT2 accelerated the construction of real Grid applications. And by defining and implementing "standard" protocols and services, GT pioneered the creation of interoperable Grid systems and enabled significant progress on Grid programming tools. This standardization played a significant role in spurring the subsequent explosion of interest, tools, applications, and deployments.

Commoditization of Grids …

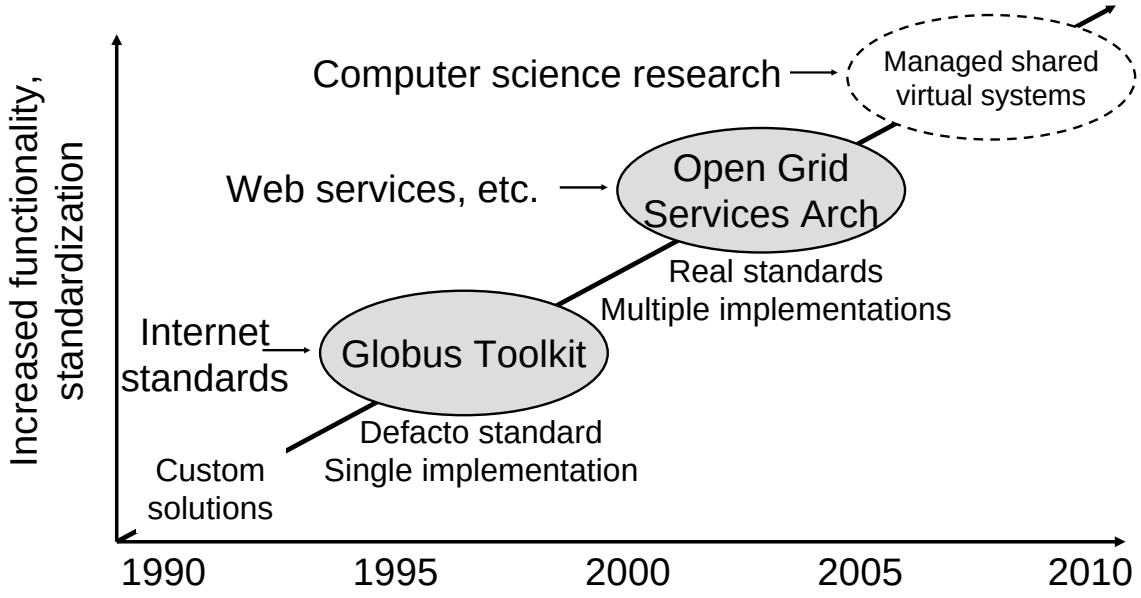

Figure 11: Evolution of Grid technology

14.2 Open Grid Services Architecture

As interest in Grids continued to grow, and in particular as industrial interest emerged, the importance of true standards increased. In particular, 2002 saw the emergence of the Open Grid Services Architecture [48] (OGSA), a community standard with multiple implementations—including the OGSA-based GT version 3 [51] released in 2003. Building on and significantly extending GT2 concepts and technologies, OGSA aligns Grid computing with broad industry initiatives in service-oriented architecture and Web services.

In addition to defining a core set of standard interfaces and behaviors that address many of the technical challenges introduced above, OGSA provides a framework within which can defined a wide range of interoperable, portable services. OGSA thus provides a foundation on which can be constructed a rich Grid technology ecosystem comprising multiple technology providers. Thus, we see, for example, major efforts underway to develop data access and integration services [69, 77].

Concurrent with these developments we see a growing recognition that large-scale development and deployment of Grid technologies is critical to future success in a wide range of disciplines in science, engineering, and the humanities, and increasingly large investments within industry in related areas. While research and commercial uses can have different concerns, they also have much in common, and there are promising signs that the required technologies can be developed in a strong academic-industrial partnership. The current open source code base and emerging open standards provide a solid foundation for the new open infrastructure that will emerge from this work.

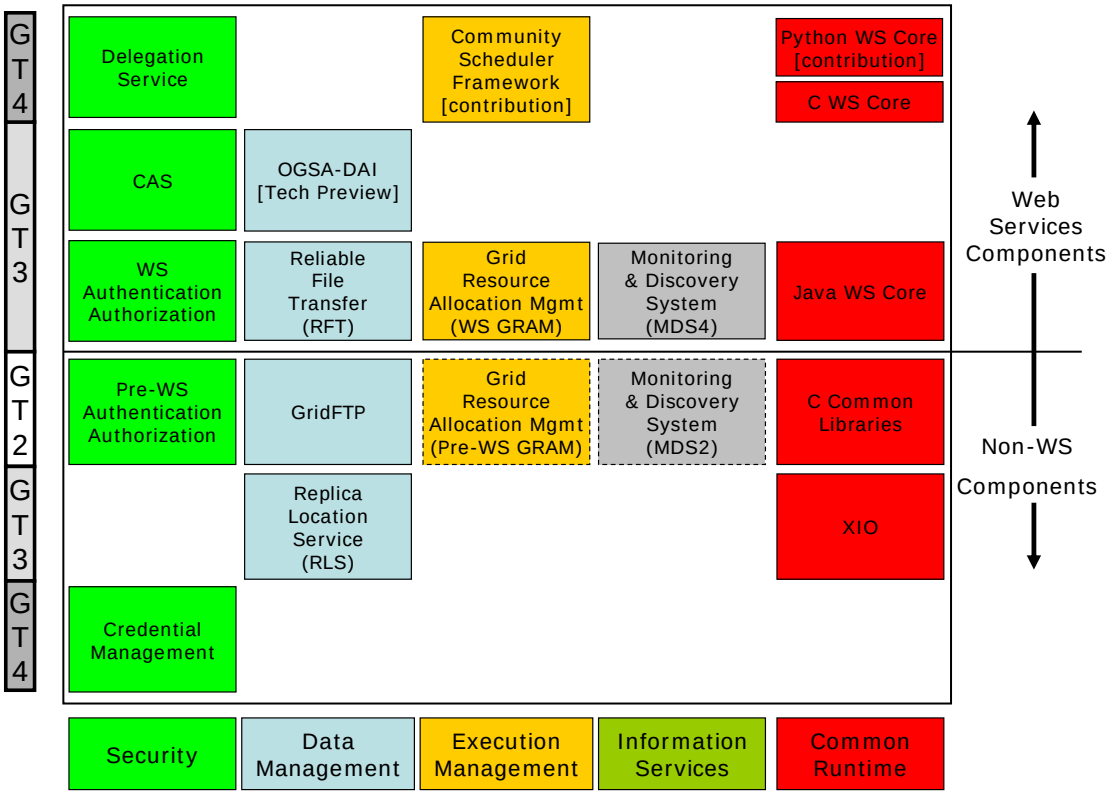

Figure 12: Evolution of Globus Toolkit components (see text for details)

Acknowledgements

GT4 is the culmination of many years of effort by many people, who have in turn benefited from many years of financial support from a variety of sources. My poor memory will certainly have led me to omit some contributors in the following. Thus, if your name is missing, please do not feel annoyed but instead let us know.

The Globus Alliance comprises individuals at Argonne National Laboratory, the University of Chicago, the University of Southern California's Information Sciences Institute, the University of Edinburgh, the Swedish Center for Parallel Computers, and the National Center for Supercomputing Applications. Staff at this institutions have been responsible for much but certainly not all Globus research, design, and development. Other significant contributors include the Condor team at the University of Wisconsin, the pyGlobus and NetLogger groups at Lawrence Berkeley National Laboratory, the EU DataGrid group at CERN, the Grid and Web services groups at IBM and HP, and Univa Corporation. Many other people have contributed by deploying, applying, and using the software, by proposing requirements, and by working on relevant technical specifications. It is the support of this community that has made the work worthwhile.

Financial support has been provided first and foremost by several U.S. federal agencies, in particular the Department of Energy, National Science Foundation, National Aeronautics and Space Administration, and Defense Advanced Research Projects Agency. The DOE National Collaboratories program and the NSF Middleware Initiative program have been particularly supportive. Los Alamos National Laboratory provided support for GridFTP development. IBM supported much of the work on the Java WS Core. Microsoft, Sun, and Cisco have also made contributions. In the U.K., funding for the development of DAIS has been provided by the UK eScience program, while work in Sweden has been supported by XXX.

GT4 builds extensively on other *open source software systems*. We gratefully acknowledge, in particular, our considerable debt to Apache software, such as Axis, Web services core, security, and WS-Addressing—to some of which Globus Alliance members have made substantial contributions.

This document has borrowed heavily from other sources, in particular the Globus online documentation. We are grateful to the authors of that material for allowing us to paraphrase their material.

Glossary

Endpoint reference

WS-Addressing

WSRF

WS-Notification

DN

Handle

WS-Transfer

SOAP

WSDL

WS-Eventing

WSDM

WS-Management

References

1. DMTF Common Information Model (CIM) Standards, 2004. www.dmtf.org/standards/cim.
2. The Ganglia Project, 2004. http://ganglia.sourceforge.net.
3. Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective, 2004. http://www-unix.globus.org/toolkit/docs/development/4.0-drafts/security/GT4-GSI-Overview.pdf.
4. GLUE Schemas Activity and Specifications, 2004. www.cnaf.infn.it/~sergio/datatag/glue.
5. Handle System, 2004. www.handle.net.
6. Log4j Logging Services, 2004. http://logging.apache.org/log4j.
7. Nagios Web Site, 2004. www.nagios.org.
8. OASIS Web Services Distributed Management (WSDM) Technical Committee, 2004. www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm.
9. OASIS WS-Notification Technical Committee, 2004. www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn.
10. OASIS WS Resource Framework Technical Committee, 2004. www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.
11. Open Grid Services Architecture Data Access and Integration (OGSA-DAI) Project. www.ogsa-dai.org.uk.
12. Storage Resource Broker (SRB), 2004. www.npaci.edu/DICE/SRB/.
13. UberFTP GridFTP-Enabled FTP Client, 2004. http://dims.ncsa.uiuc.edu/set/uberftp.
14. W3C SOAP Activity, 2002. www.w3.org/TR/SOAP.
15. W3C XML Schema Activity, 2001. www.w3.org/XML/Schema.
16. Web Services Interoperability Organization, 2004. www.ws-i.org.
17. Abdelnur, A. and Hepper, S. Portlet API Specification Version 1.0. Java Community Process, JSR-000168, 2003.
18. Abramson, D., Giddy, J. and Kotler, L., High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, Cancun, Mexico, 2000, 520-528.
19. Alexander, J., Box, D., Cabrera, L.F., Chappell, D., Daniels, G., Geller, A., Janecek, R., Kaler, C., Lovering, B., Orchard, D., Schlimmer, J., Sedukhin, I. and Shewchuk, J. Web Services Transfer (WS-Transfer), 2004.
20. Allcock, W. GridFTP: Protocol Extensions to FTP for the Grid. Global Grid ForumGFD-R-P.020, 2003.
21. Allcock, W., Bresnahan, J., Kettimuthu, R. and Link, J., The Globus eXtensible Input/Output System (XIO): A Protocol-Independent I/O System for the Grid. *Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models in conjunction with International Parallel and Distributed Processing Symposium*, 2005.
22. Anderson, R. *Security Engineering*. Wiley, 2001.
23. Arora, A., Geller, A., He, J., Kaler, C., McCollum, R., Milenkovic, M., Montgomery, P., Saiyed, J. and Suen, E. Web Services for Management (WS-Management), 2004. www.intel.com/technology/manage/downloads/ws_management.pdf.
24. Atkinson, M., Chervenak, A., Kunszt, P., Narang, I., Paton, N., Pearson, D., Shoshani, A. and Watson, P. Data Access, Integration, and Management. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 2004.
25. Baratloo, A., Karaul, M., Kedem, Z. and Wyckoff, P., Charlotte: Metacomputing on the Web. *9th International Conference on Parallel and Distributed Computing Systems*, 1996.

26. Baru, C., Moore, R., Rajasekar, A. and Wan, M., The SDSC Storage Resource Broker. *8th Annual IBM Centers for Advanced Studies Conference*, Toronto, Canada, 1998.
27. Bent, J., Venkataramani, V., LeRoy, N., Roy, A., Stanley, J., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H. and Livny, M., Flexibility, Manageability, and Performance in a Grid Storage Appliance. *11th IEEE International Symposium on High Performance Distributed Computing*, 2002, IEEE Computer Society Press.
28. Berman, F., Fox, G. and Hey, T. (eds.). *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons, 2003.
29. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. and Orchard, D. Web Services Architecture. W3C, Working Draft http://www.w3.org/TR/2003/WD-ws-arch-20030808/, 2003.
30. Bosworth, A., Box, D., Christensen, E., Curbera, F., Ferguson, D., Frey, J., Kaler, C., Langworthy, D., Leymann, F., Lucco, S., Millet, S., Mukhi, N., Nottingham, M., Orchard, D., Shewchuk, J., Storey, T. and Weerawarana, S. Web Services Addressing (WS-Addressing). 2003.
31. Box, D., Cabrera, L.F., Critchley, C., Curbera, F., Ferguson, D., Geller, A., Graham, S., Hull, D., Kakivaya, G., Lewis, A., Lovering, B., Mihic, M., Niblett, P., Orchard, D., Saiyed, J., Samdarshi, S., Schlimmer, J., Sedukhin, I., Shewchuk, J., Smith, B., Weerawarana, S. and Wortendyke, D. Web Services Eventing (WS-Eventing), 2004. www-106.ibm.com/developerworks/webservices/library/specification/ws-eventing/.
32. Bray, T., Hollander, D., Layman, A. and Tobin, R. Namespaces in XML 1.1. W3C, 2004. www.w3.org/TR/xml-names11.
33. Brecht, T., Sandhu, H., Shan, M. and Talbot, J. ParaWeb: Towards World-Wide Supercomputing. *Proc. Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications*, 1996.
34. Catlett, C. In Search of Gigabit Applications. *IEEE Communications Magazine* (April). 42-51. 1992.
35. Catlett, C. and Smarr, L. Metacomputing. *Communications of the ACM*, *35* (6). 44-52. 1992.
36. Chadwick, D.W. and Otenko, A., The PERMIS X.509 Role Based Privilege Management Infrastructure. *7th ACM Symposium on Access Control Models and Technologies*, 2002.
37. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C. and Tuecke, S. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets. *J. Network and Computer Applications*, *23* (3). 187-200. 2000.
38. Christensen, E., Curbera, F., Meredith, G. and Weerawarana., S. Web Services Description Language (WSDL) 1.1. W3C, Note 15, 2001. www.w3.org/TR/wsdl.
39. Czajkowski, K., Fitzgerald, S., Foster, I. and Kesselman, C., Grid Information Services for Distributed Resource Sharing. *10th IEEE International Symposium on High Performance Distributed Computing*, 2001, IEEE Computer Society Press, 181-184.
40. Eickermann, T. and Hommes, F. Metacomputing in a Gigabit Testbed West. *Workshop on Wide Area Networks and High Performance Computing*, Springer-Verlag, 1999, 119-129.
41. Ellisman, M. and Peltier, S. Medical Data Federation: The Biomedical Informatics Research Network. *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*, Morgan Kaufmann, 2004.
42. Erdos, M. and Cantor, S. Shibboleth Architecture. Internet 2, 2002. http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v05.pdf.
43. Federico, D., Sacerdoti, M.J.K., Massie, M.L. and Culler, D.E., Wide Area Cluster Monitoring with Ganglia. *IEEE International Conference on Cluster Computing*, 2003, IEEE Press.
44. Foster, I. The Grid: Computing without Bounds. *Scientific American*, *288* (4). 78-85. 2003.

45. Foster, I., Alpert, E., Chervenak, A., Drach, B., Kesselman, C., Nefedova, V., Middleton, D., Shoshani, A., Sim, A. and Williams, D., The Earth System Grid II: Turning Climate Datasets Into Community Resources. *Annual Meeting of the American Meteorological Society*, 2002.
46. Foster, I., Berry, D., Djaoui, A., Grimshaw, A., Horn, B., Kishimoto, H., Maciel, F., Savva, A., Siebenlist, F., Subramaniam, R., Treadwell, J. and Reich, J.V. Open Grid Services Architecture V1. 2004.
47. Foster, I., Frey, J., Graham, S., Tuecke, S., Czajkowski, K., Ferguson, D., Leymann, F., Nally, M., Sedukhin, I., Snelling, D., Storey, T., Vambenepe, W. and Weerawarana, S. Modeling Stateful Resources with Web Services, www.globus.org/wsrf, 2004.
48. Foster, I. and Kesselman, C. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, *11* (2). 115-129. 1998.
49. Foster, I. and Kesselman, C. (eds.). *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*. Morgan Kaufmann, 2004.
50. Foster, I., Kesselman, C., Nick, J. and Tuecke, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG, Global Grid Forum, 2002.
51. Foster, I., Kesselman, C., Nick, J.M. and Tuecke, S. Grid Services for Distributed Systems Integration. *IEEE Computer*, *35* (6). 37-46. 2002.
52. Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S., A Security Architecture for Computational Grids. *5th ACM Conference on Computer and Communications Security*, 1998, 83-91.
53. Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, *15* (3). 200-222. 2001.
54. Foster, I. and others, The Grid2003 Production Grid: Principles and Practice. *IEEE International Symposium on High Performance Distributed Computing*, 2004, IEEE Computer Science Press.
55. Frey, J., Tannenbaum, T., Foster, I., Livny, M. and Tuecke, S. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, *5* (3). 237-246. 2002.
56. Graham, S., Niblett, P., Chappell, D., Lewis, A., Nagaratnam, N., Parikh, J., Patil, S., Samdarshi, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W. and Weihl, B. Publish-Subscribe Notification for Web Services, 2004. www-106.ibm.com/developerworks/library/ws-pubsub.
57. Grimshaw, A., Weissman, J., West, E. and E. Lyot, J. Metasystems: An Approach Combining Parallel Processing and Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, *21* (3). 257-270. 1994.
58. Grimshaw, A.S. and Wulf, W.A. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, *40* (1). 39-45. 1997.
59. Hughes, J. and Maler, E. Technical Overview of the OASIS Security Assertion Markup Language (SAML) v1.1, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, 2004.
60. Karonis, N., Toonen, B. and Foster, I. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, *63* (5). 551-563. 2003.
61. Keahey, K., Fredian, T., Peng, Q., Schissel, D.P., Thompson, M., Foster, I., Greenwald, M. and McCune, D. Computational Grids in Action: The National Fusion Collaboratory. *Future Generation Computer Systems*, *18* (8). 1005-1015. 2002.
62. Kendall, S.C., Waldo, J., Wollrath, A. and Wyant, G. A Note on Distributed Computing. Sun MicrosystemsTR-94-29, 1994.
63. Lin, A., Maas, P., Peltier, S. and Ellisman, M. Harnessing the Power of the Globus Toolkit. *Cluster World*, *2* (1). 2004.

64. Litzkow, M. and Livny, M. Experience with the Condor Distributed Batch System. *IEEE Workshop on Experimental Distributed Systems*, 1990.
65. Litzkow, M.J., Livny, M. and Mutka, M.W. Condor - A Hunter of Idle Workstations. *8th International Conference on Distributed Computing Systems*, 1988, 104-111.
66. Menezes, A., Oorschot, P.v. and Vanstone, S. *Handbook of Applied Cryptography*. CRC Press, 1996.
67. Messina, P. Distributed Supercomputing Applications. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 55-73.
68. Novotny, J., Tuecke, S. and Welch, V., An Online Credential Repository for the Grid: MyProxy. *10th IEEE International Symposium on High Performance Distributed Computing*, San Francisco, 2001, IEEE Computer Society Press.
69. Paton, N.W., Atkinson, M.P., Dialani, V., Pearson, D., Storey, T. and Watson, P. Database Access and Integration Services on the Grid. U.K. National eScience Center, 2002. www.nesc.ac.uk.
70. Pearlman, L., Kesselman, C., Gullapalli, S., Spencer, B.F., Futrelle, J., Ricker, K., Foster, I., Hubbard, P. and Severance, C., Distributed Hybrid Earthquake Engineering Experiments: Experiences with a Ground-Shaking Grid Application. *13th IEEE International Symposium on High Performance Distributed Computing*, 2004.
71. Rosenberg, J. and Remy, D. *Securing Web Services with WS-Security*. Sams, 2004.
72. Smallen, S., Olschanowsky, C., Ericson, K., Beckman, P. and Schopf, J.M., The Inca Test Harness and Reporting Framework. *SC'2004 High Performance Computing, Networking, and Storage Conference*, 2004.
73. Sotamayor, B. The Globus Toolkit 4 Programmer's Tutorial, 2005. www.casa-sotomayor.net/gt4-tutorial.
74. Spencer, B., Finholt, T., Foster, I., Kesselman, C., Beldica, C., Futrelle, J., Gullapalli, S., Hubbard, P., Liming, L., Marcusiu, D., Pearlman, L., Severance, C. and Yang, G., NEESgrid: A Distributed Collaboratory for Advanced Earthquake Engineering Experiment and Simulation. *13th World Conference on Earthquake Engineering*, Vancouver, B.C., Canada, 2004, Paper No. 1674.
75. Vahdat, A., Belani, E., Eastham, P., Yoshikawa, C., Anderson, T., Culler, D. and Dahlin, M., WebOS: Operating System Services For Wide Area Applications. *7th IEEE International Symposium on High Performance Distributed Computing*, 1998, IEEE Computer Society Press.
76. Welch, V., Barton, T., Keahey, K. and Siebenlist, F., Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration. 2004.
77. Welch, V., Siebenlist, F., Foster, I., Bresnahan, J., Czajkowski, K., Gawor, J., Kesselman, C., Meder, S., Pearlman, L. and Tuecke, S., Security for Grid Services. *12th IEEE International Symposium on High Performance Distributed Computing*, 2003.
78. Zhang, X., Freschl, J.L. and Schopf, J.M., A Performance Study of Monitoring and Information Services for Distributed Systems. *12th IEEE International Symposium on High Performance Distributed Computing*, 2003, IEEE Computer Society Press.

Index