GWD-I (draft-ggf-ogsa-spec-019)
Open Grid Services Architecture
http://forge.gridforum.org/projects/ogsa-wg

Authors:
I. Foster, Argonne & U.Chicago (Editor)
D. Berry, NeSC
A. Djaoui, CCLRC-RAL
A. Grimshaw, UVa
B. Horn, IBM
H. Kishimoto, Fujitsu (Editor)
F. Maciel, Hitachi
A. Savva, Fujitsu (Editor)
F. Siebenlist, ANL
R. Subramaniam, Intel
J. Treadwell, HP
J. Von Reich, HP

12 July 2004

The Open Grid Services Architecture, Version 1.0

Status of this Memo

Abstract

Successful realization of the Open Grid Services Architecture (OGSA) vision of a broadly applicable and adopted framework for distributed system integration, virtualization, and management requires the definition of a core set of interfaces, behaviors, resource models, and bindings. This document, produced by the OGSA working group within the Global Grid Forum (GGF), provides a first, and necessarily preliminary and incomplete, version of this OGSA definition. The document specifies requirements, the scope of important capabilities and services required to support Grid systems and applications in both e-science and e-business, identifies a core set of such services that are viewed as essential for many systems and applications, and specifies at a high-level the functionalities required for these core services and the interrelationships among those core services.

GLOBAL GRID FORUM
office@ggf.org
www.ggf.org

Contents

1 Introduction

Grid systems and applications aim to integrate, virtualize, and manage resources and services within distributed, heterogeneous, dynamic "virtual organizations" [Grid Anatomy] [Grid Physiology]. The realization of this goal requires the disintegration of the numerous barriers that normally separate different computing systems within and across organizations, so that computers, application services, data, and other resources can be accessed as and when required, regardless of physical location.

Key to the realization of this Grid vision is standardization, so that the diverse components that make up a modern computing environment can be discovered, accessed, allocated, monitored, accounted for, billed for, etc., and in general managed as a single virtual system—even when provided by different vendors and/or operated by different organizations. Standardization is critical if we are to create interoperable, portable, and reusable components and systems; it can also contribute to the development of secure, robust, and scalable Grid systems by facilitating the use of good practices.

In this document, we present an *Open Grid Services Architecture* (OGSA) that addresses this need for standardization by defining, within a service-oriented architecture, a set of core capabilities and behaviors that address key concerns in Grid systems. These concerns include such issues as: How do I establish identity and negotiate authentication? How is policy expressed and negotiated? How do I discover services? How do I negotiate and monitor service level agreements? How do I manage membership of, and communication within, virtual organizations? How do I organize service collections hierarchically so as to deliver reliable and scalable service semantics? How do I integrate data resources into computations? How do I monitor and manage collections of services?

In this document, we first identify requirements for Grid systems and then translate those requirements into a coherent set of capabilities that collectively define OGSA. First, in §2, we provide an abstract definition of the set of requirements that OGSA is intended to address. This analysis is based on requirements, technical challenges, use cases, previous experience, and the state of the art in related work. The abstract rendering is not constrained by any assumptions about the underlying infrastructure, but rather is intended to frame the "Grid" discussion and define solutions.

In §3, we first describe infrastructure services and assumptions that constrain our development of the OGSA design, in particular explaining how OGSA both builds on, and is contributing to the development of, the growing collection of technical specifications that form the emerging Web services (WS) architecture [WS-Architecture]. Then, we present a refinement of the required functionality into capabilities: Execution Management, Data, Resource Management, Security, Self-management and Information services.

In a later draft we will provide descriptions of specific services, making clear where existing service specifications can be used unchanged and where modified or new service specifications are needed. We will also describe the current state of any work known to be underway to define such extensions and/or definitions.

This informational document (GWD-I), a product of the Global Grid Forum's OGSA working group, defines OGSA version 1. The OGSA working group is committed to releasing a recommendation (GWD-R) version of this document in the future as our understanding of OGSA's purpose and form, and the details of specific components, evolves.

2 Requirements

This definition of OGSA version 1 is driven by a set of functional and non-functional requirements, which themselves are informed by the use cases listed in Table 1 and described in companion documents [OGSA Use Cases][OGSA Use Cases Tier 2]. These use cases do not constitute a formal requirements analysis, but have provided useful input to the architecture definition process.

Table 1: The OGSA use cases

Use case	Summary
Commercial Data Center (CDC)	Data centers will have to manage several thousands of IT resources, including servers, storage, and networks, while reducing management costs and increasing resource utilization.
Severe Storm Modeling	Enable accurate prediction of the exact location of severe storms based on a combination of real-time wide area weather instrumentation and large-scale simulation coupled with data modeling.
Online Media and Entertainment	Delivering an entertainment experience, either for consumption or interaction.
National Fusion Collaboratory (NFC)	Defines a virtual organization devoted to fusion research and addresses the needs of software developed and executed by this community based on the application service provider (ASP) model.
Service-Based Distributed Query Processing	A service-based distributed query processor supporting the evaluation of queries expressed in a declarative language over one or more existing services.
Grid Workflow	Workflow is a convenient way of constructing new services by composing existing services. A new service can be created and used by registering a workflow definition to a workflow engine.
Grid Resource Reseller	Inserting a supply chain between the Grid resource owners and end users will allow the resource owners to concentrate on their core competences, while end users can purchase resources bundled into attractive packages by the reseller.
Inter Grid	Extends the CDC use case by emphasizing the plethora of applications that are not Grid-enabled and are difficult to change: e.g. mixed Grid and non-Grid data centers, and Grid across multiple companies. Also brings into view generic concepts of utility computing.
Interactive Grids	Compared to the online media use case, this use case emphasizes a high granularity of distributed execution.
Grid Lite	Extends the use of Grids to small devices – PDAs, cell phones, firewalls, etc., and identifies a set of essential services that enable the device to be part of a Grid environment.
Virtual Organization (VO) Grid Portal	A VO gives its members access to various computational, instrument-based data and other types of resources. A Grid portal provides an end-user view of the collected resources available to the

	members of the VO.
Persistent Archive	Preservation environments handle technology evolution by providing appropriate abstraction layers to manage mappings between old and new protocols, software and hardware systems, while maintaining authentic records.
Mutual Authorization	Refines the CDC and NFC use cases by introducing the scenario of the job submitter authorizing the resource on which the job will eventually execute.
Resource Usage Service	Facilitates the mediation of resource usage metrics produced by applications, middleware, operating systems, and physical (compute and network) resources in a distributed, heterogeneous environment.
IT Infrastructure and Management*	Job execution, cycle sharing and provisioning scenarios.
Application Use Cases*	Peer-to-Peer PC Grid computing, file sharing and content delivery scenarios.
Reality Grid*	Distributed and collaborative exploration of parameter space through computational steering and on-line, high-end visualization.
The Learning GRID*	User-centered, contextualized and experiential-based approaches for ubiquitous learning in the framework of a Virtual Organization.
HLA-based Distributed Simulation*	A distributed collaborative environment for developing and running simulations across administrative domains.
GRID based ASP for Business*	An infrastructure for Application Service Provision (ASP) supporting different business models based on GRID technology.
Grid Monitoring Architecture*	Grid monitoring system scalable across wide-area networks and able to encompass a large number of dynamic and heterogeneous resources.

* *Use cases appearing in Tier 2 document*

Analysis of the use cases and other relevant input led us to identify both important and apparently broadly-relevant characteristics of Grid environments and applications, and functional and non-functional requirements that appear to have relevance to a variety of application scenarios. We summarize our findings in the following sections.

2.1 Dynamic and Heterogeneous Environment Support

Some use cases involve highly constrained or homogeneous environments that may well motivate specialized profiles. However, it is clear that, in general, Grid environments tend to be heterogeneous and distributed, encompassing a variety of operating systems (e.g., Unix, Linux, Windows, or embedded systems), hosting environments (e.g., J2EE, .NET), devices (e.g., computers, instruments, sensors, storage systems, databases, networks), and services, provided by various vendors. In addition, the environment is long-lived and dynamic, and may therefore evolve in ways not initially anticipated.

OGSA must enable interoperability between these diverse, heterogeneous, and distributed resources and services as well as reduce the complexity of administrating heterogeneous systems. Moreover, many functions required in distributed environments, such as security and resource management, may already be implemented by stable and reliable legacy systems, and thus it is required to be able to integrate such legacy systems into the Grid.

Requirements for heterogeneous systems support include:

- *Resource virtualization* – Essential to reduce the complexity of managing heterogeneous systems and to handle diverse resources in a unified way.
- *Common management capabilities* – Simplifying administration of a heterogeneous system requires mechanisms for uniform and consistent management of resources. A minimum set of common manageability capabilities is required.
- *Resource discovery and query* – Mechanisms are required for discovering resources with desired attributes and for retrieving their properties. Discovery and query should handle a highly dynamic system well.
- *Standard protocols and schemas* – Important for interoperability. In addition, standard protocols are also particularly important as their use can simplify the transition to using Grids.

2.2 Resource Sharing Across Organizations

The Grid is not a monolithic system but is composed of resources owned and controlled by various organizations. One major purpose of OGSA is to support resource sharing and utilization across administrative domains. Mechanisms are needed to provide a context that can be used to associate users, requests, resources, policies, and agreements across organizational boundaries. Sharing resources across organizations also implies various security requirements, a topic we address in §2.7.

Resource sharing requirements include:

- A *global name space* – To ease data and resource access. OGSA entities should be able to access other OGSA entities transparently, subject to security constraints, without regard to location or replication.
- *Metadata services* – Important for finding, invoking, and tracking entities. It should be possible to define views on entities that allow for access to and propagation, aggregation, and management of entity metadata across administrative domains.
- *Site autonomy* – Mechanisms are required for accessing resources across sites while respecting local control and policy (see Delegation in §2.7).
- Mechanisms and standard schemas for *collecting and exchanging usage information across organizations*, e.g., for the purpose of accounting, billing, etc.

2.3 Optimization

Optimization applies to both suppliers and consumers of resources and services. One common case of supply-side optimization is resource optimization. For example, resource allocation often provides for the worst case scenarios (e.g., highest expected load, backup against failures, etc.) and leads to resource underutilization as resources that are allocated to address such scenarios remain unused for long periods. Resource utilization can be improved by flexible resource allocation policies, including advance reservation of resources with a bounded time period, the pooling of backup resources, etc.

Mechanisms for monitoring resource utilization, including metering, monitoring and logging, changing resource allocation, and provisioning resources dynamically and on demand are the required foundation of demand-side optimization. Demand-side optimization must include different ways of managing various workloads, since Grid workloads can be unpredictable or chaotic. An important requirement for workload management is the ability to dynamically adjust priorities in order to meet service level objectives.

2.4 Quality of Service (QoS) Assurance

Services such as job execution and data services must provide the agreed-upon QoS. Key QoS dimensions include, but are not limited to, availability, security, and performance. QoS requirements should be expressed using measurable terms that can be captured in Service Level Agreements (SLAs).

QoS assurance requirements include:

- *Service level agreement* – QoS should be represented by agreements which are established by negotiating between service requester and provider prior to service execution. Standard mechanisms should be provided to create and manage agreements.
- *Service level attainment* – If the agreement requires attainment of Service Level, the resources used by the service should be adjusted so that the required QoS is maintained. Therefore, mechanisms for monitoring services quality, estimating resource utilization, and planning for and adjusting resource usage are required.
- *Migration* – It should be possible to migrate executing services or applications to adjust workloads for performance or availability.

2.5 Job Execution

OGSA must provide manageability for execution of user-defined tasks (jobs) throughout their lifetime. Functionalities such as scheduling, provisioning, job control and exception handling of jobs must be supported, even when the job is distributed over a great number of heterogeneous resources.

Job execution requirements include:

- *Support for various job types* – Execution of various types of jobs must be supported including simple jobs and complex jobs such as workflow and composite services.
- *Job management* – It is essential to be able to manage jobs during their entire lifetimes. Jobs must support manageability interfaces and these interfaces must work with various types of groupings of jobs (e.g., workflows, job arrays). Mechanisms are also required for controlling the execution of individual job steps as well as orchestration or choreography services.
- *Scheduling* – The ability to schedule and execute tasks based on such information as specified priority and current allocation of resources is required. It is also required to realize mechanisms for scheduling across administrative domains, using multiple schedulers.
- *Resource provisioning* – To automate the complicated process of *resource allocation, deployment, and configuration*. It must be possible to deploy the required applications and data to resources and configure them automatically, if necessary deploying and re-configuring hosting environments such as OS and middleware to prepare the environment needed for job execution. It must be possible to provision any type of resource not just compute resources, for example, network or data resources.

2.6 Data Services

Efficient access to and movement of huge quantities of data is required in more and more fields of science and technology. In addition, data sharing is important, for example enabling access to information stored in databases that are managed and administered independently. In business areas, archiving of data and data management are essential requirements.

Data services requirements include:

- *Data access* – Easy and efficient access to various types of data (such as database, files, and streams), independent of its physical location, by abstracting underlying data sources is required. Mechanisms are also required for controlling access rights at different levels of granularity.
- *Data consistency* – OGSA must ensure that consistency can be maintained when cached or replicated data is modified.
- *Data persistency* – Data and its association with its metadata should be maintained for their entire lifetime. It should be possible to use multiple persistency models.
- *Data integration* – OGSA should provide mechanisms for integrating heterogeneous, federated and distributed data. It is also required to be able to search data available in various formats in a uniform way.
- *Data location management* – The required data should be made available at the requested location. OGSA should allow for selection in various ways, such as transfer, copying, and caching, according to the nature of data.

2.7 Security

Safe administration requires controlling access to services through robust security protocols and according to provided security policy. For example, obtaining application programs and deploying them into a Grid system may require authentication and authorization. Also sharing of resources by users requires some kind of isolation mechanism. In addition, standard, secure mechanisms are required which can be deployed to protect Grid systems while supporting safe resource-sharing across administrative domains.

Security requirements include:

- *Authentication and authorization* – Authentication mechanisms are required so that the identity of individuals and services can be established. Service providers must implement authorization mechanisms to enforce policy over how each service can be used. The Grid system should follow each domain's security policies and also may have to identify users' security policies. Authorization should accommodate various access control models and implementations.
- *Multiple security infrastructures* – Distributed operation implies a need to integrate and interoperate with multiple security infrastructures. OGSA needs to integrate and interoperate with existing security architectures and models.
- *Perimeter security solutions* – Applications may have to be deployed outside the client domain. OGSA needs standard and secure mechanisms that can be deployed to protect institutions while also enabling cross-domain interaction without compromising local control of equipment for protecting domains, such as firewall policy and intrusion-detection policy.

- *Isolation* – Various kinds of isolation must be ensured, such as isolation of users, performance isolation, and isolation between content offerings within the same Grid system.
- *Delegation* – Mechanisms that allow for delegation of access rights from requestors to services are required. The rights transferred through delegation are scoped only to the intended job and should have a limited lifetime to minimize the risk of misuse of delegated rights.
- *Security policy exchange* – Service requestors and providers should be able to exchange dynamically security policy information to establish a negotiated security context between them.
- *Intrusion detection, protection, and secure logging* – Strong monitoring is required for intrusion detection and identification of misuses, malicious or otherwise, including virus or worm attacks. It should also be possible to protect critical areas or functions by migrating attacks away from them.

2.8 Administrative Cost Reduction

The complexity of administering large-scale distributed, heterogeneous systems increases administration costs and the risk of human errors. Support for administration tasks, by automating administrative operations and consistent management of virtualized resources, is needed.

Policy-based management is required to automate Grid system control, so that its operations conform to the goals of the organization that operates and utilizes the Grid system. From the low-level policies that govern how the resources are monitored and managed to high-level policies that govern how business processes such as billing are managed, there may be policies at every level of the system. Policies may include availability, performance, security, scheduling, and brokering.

Application contents management mechanisms can facilitate the deployment, configuration, and maintenance of complex systems, by allowing all application-related information to be specified and managed as a single logical unit. This approach allows administrators to maintain application components in a concise and reliable manner, even without expert knowledge about the applications.

Problem determination mechanisms are needed, so that administrators can recognize and cope quickly with emerging problems.

2.9 Scalability

A large-scale Grid system can create added value such as drastically reducing job turn around (or elapsed) time, allowing for utilizing huge number of resources, thereby enabling new services. However, the large scale of the system may present problems, since it places novel demands on the management infrastructure.

The *management architecture* needs to scale to potentially thousands of resources of a widely varied nature. Management needs to be done in a hierarchical or peer-to-peer (federated/collaborative) fashion.

High-throughput computing mechanisms are required for adjusting and optimizing parallel job execution in order to improve throughput of the entire computational process, as well as optimizing a single computation.

2.10 Availability

High availability is often realized by expensive fault-tolerant hardware or complex cluster systems. Because of the widespread use of IT systems to provide essential public infrastructure services, an increasing number of systems are required to operate at a high level of availability. Since Grid technologies enable transparent access to a wider resource pool, *across* organizations as well as *within* organizations, they can be used as one building block to realize stable, highly-reliable execution environments. But due to the heterogeneity of the Grid, components with longer or more unpredictable mean-time-to-repair (MTTR) characteristics than those generally used in existing high-reliability systems have to be used, presenting difficult problems.

In such a complex environment, policy-based autonomous control (see Policy-based Management in §2.8) and dynamic provisioning (see Provisioning in §2.5) are keys to realizing systems of high flexibility and recoverability.

Disaster recovery mechanisms are needed so that the operation of a Grid system can be recovered quickly and efficiently in case of natural or human-caused disaster, avoiding long-term service disruption. Remote backup and simplifying or automating recovery procedures is required.

Fault management mechanisms can be required so that running jobs are not lost because of resource faults. Mechanisms are required for fault detection, and diagnosis of causes or impacts on running jobs. In addition, if possible, automation of fault handling using techniques such as checkpoint recovery is required.

2.11 Ease of Use and Extensibility

The user should be able to use OGSA to mask the complexity of the environment if so required. As much as possible, tools, acting in concert with run-time facilities, must manage the environment for the user and provide useful abstractions at the desired level. Tempering this ease-of-use objective is the knowledge that there are “power users” with demanding applications that will require, and demand, the capability to make low-level decisions and to interface with low-level system mechanisms. Therefore it should be possible for end-users to choose the level at which they wish to interact with the system.

It is not possible to predict all of the many and varied needs that users will have. Therefore, mechanism and policy must be realized via extensible and replaceable components, to permit OGSA to evolve over time and allow users to construct their own mechanisms and policies to meet specific needs. Further, the core system components themselves must be extensible and replaceable. Such extensibility will allow third party (or site-local) implementations which provide value-added services to be developed and used. Extensibility and customization must be provided for in a way that does not compromise interoperability.

3 Capabilities

We now turn to the specific capabilities required to meet the requirements introduced above.

3.1 Overview

OGSA is intended to facilitate the seamless use and management of distributed, heterogeneous resources. In this architecture, the terms “distributed,” “heterogeneous” and “resources” are used in their broad sense. For example: “distributed” could refer to a spectrum from geographically-contiguous resources linked to each other by some connection fabric to global, multi-domain, loosely- and intermittently-connected resources. “Resources” refers to any artifact, entity or knowledge required to complete an operation in or on the system. The utility provided by such an infrastructure is realized as a set of *capabilities*. Figure 1 shows the logical, abstract, semi-layered representation of some of these capabilities. Three major logical and abstract layers are envisioned in this representation.

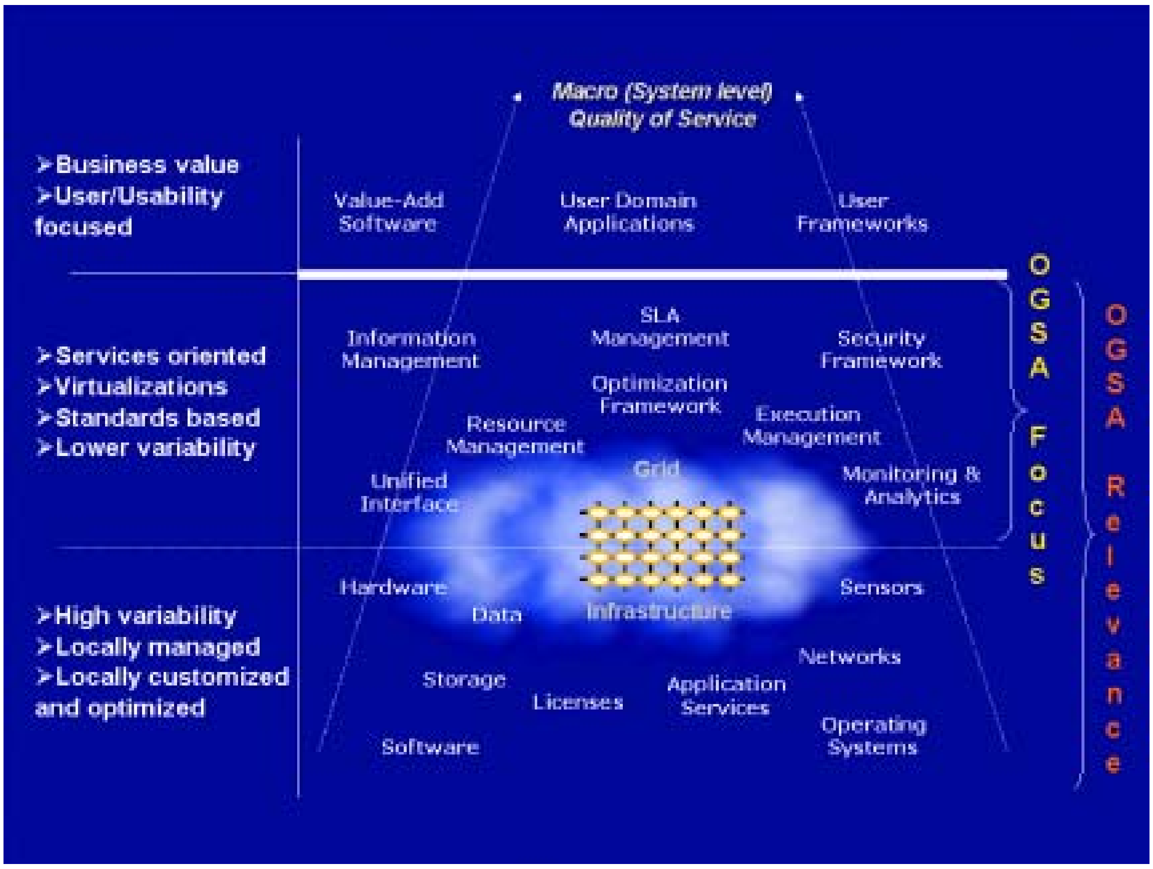

Figure 1: Conceptual view of Grid infrastructures

(*NOTE: The capabilities and resources depicted in the diagram are not an exhaustive list, and have been kept minimal for clarity.*)

The first (bottom) layer of the representation in Figure 1 depicts the *base resources*. Base resources are those resources that are supported by some underlying entities/artifacts that may be physical or logical, and that have relevance outside of the OGSA context. Examples of such physical entities include CPUs, memory, disks etc., and examples of such logical artifacts include licenses, contents, OS processes etc. The virtualizations are tightly-coupled with the entities that they virtualize, and hence the nomenclature used to name the underlying entities/artifacts is also used to name their virtualizations. These resources are usually locally owned and managed, but may be shared remotely. The configuration and customization is also done locally. Since the actual entities/artifacts can change rapidly, and can be from multiple sources, these resources can be highly variable in their characteristics, quality of service, version, availability etc.

Although in this discussion a distinction of base resources is made to tie OGSA concepts to traditional notions of resources, further discussion in this document makes no specific distinction between base resources and other services as resources, and only the generalized notion of resources is used.

The second (middle) layer represents a higher level of virtualization and logical abstraction. The virtualization and abstraction are directed toward defining a wide variety of *capabilities* that are relevant to OGSA Grids. These capabilities can be utilized individually or composed as appropriate to provide the infrastructure required to support higher-level applications or "user" domain processes. This set of capabilities, as defined in OGSA, is relatively invariant and standard. The manner in which these capabilities are realized/implemented and further composed and extended by user-domain applications determines the macro (system level) QoS of the larger infrastructure, as experienced by an end-user. *It should be noted that the capabilities shown in the diagram only represent a sample of the OGSA capabilities, and that a more complete set is discussed in the rest of this chapter.*

There is no clear boundary between the middle and bottom layers in the logical diagram. This is because OGSA has to comprehend the forms and types of resources, and use this understanding to frame the discussion of capabilities in the middle layer. In addition these resources (i.e., virtualizations) will be driven, used and/or managed in realizing many of the capabilities in middle layer. This close relationship in the OGSA discussion is indicated by the finer (thin line) demarcation of the bottom and middle layers in Figure 1. Hence the entities in the lower layer may be regarded as relevant to, and part of, the OGSA discussion.

At the third (top) layer in the logical representation are the applications and other entities that use the OGSA capabilities to realize user- and domain-oriented functions and process, such as business processes. These are, for the most part, outside the purview of OGSA, but they drive the definition of the architecture from the use cases that the infrastructure should support.

3.2 OGSA Framework

OGSA realizes the logical middle layer in Figure 1 in terms of *services*, the *interfaces* these services expose, the individual and collective *state* of resources belonging to these services, and the *interaction* between these services within a *service-oriented architecture* (SOA). The OGSA services framework is shown in Figure 2 and Figure 3. The services are built on Web service standards, with semantics, additions, extensions and modifications that are relevant to Grids.

A few important points are to be noted:

- An important motivation for OGSA is the *composition paradigm* or *building block* approach, where a set of capabilities or functions is built or adapted as required, from a minimalist set of initial capabilities, to meet a need. No prior knowledge of this need is assumed. This provides the adaptability, flexibility and robustness to change that is required in the architecture.
- OGSA represents the services, their interfaces, and the semantics/behavior and interaction of these services. It should be noted that the software architecture driving the implementation of the internals of these services is outside the OGSA working group's scope.
- In addition, the architecture is not *layered*, where the implementation of one service is built upon, and can only interact with, the layer upon which it is logically dependent; or *object-oriented*—though many of the concepts may seem to be object-based.

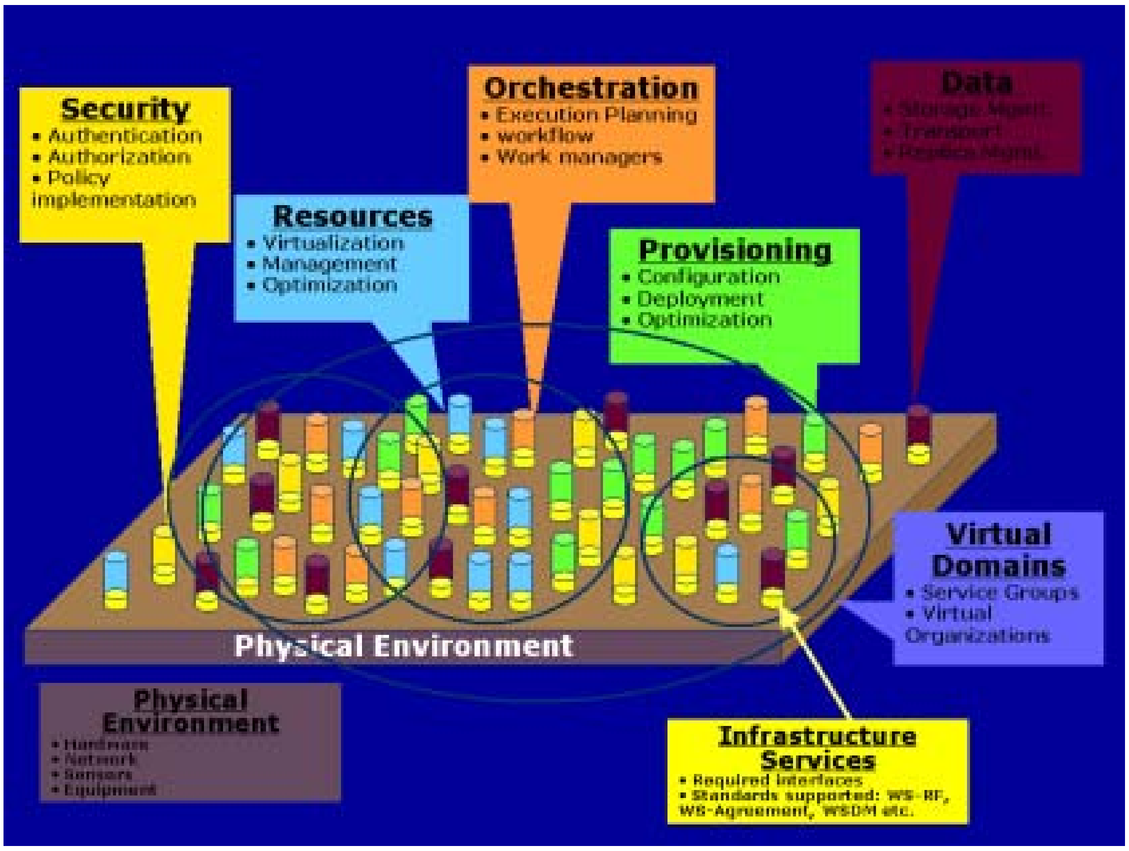

Figure 2: OGSA framework

(NOTE: The capabilities listed here are not exhaustive. See the rest of this chapter for more details)

Services are loosely coupled peers that, either singly or as part of an interacting group of services, realize the capabilities of OGSA through implementation, composition, or interaction with other services (see Figure 2). For example: to realize the “orchestration” capability a group of services might be structured such that one set of services in the group drives the orchestration (i.e., acts as the “orchestrator”), while other services in the group provide the interfaces and mechanisms to be orchestrated (i.e., be the “orchestratees”). A specific service may implement and/or participate in multiple collections and interactions to realize different capabilities. On the other hand, it is not necessary that *all* services participate to realize a particular capability.

The services may be part of, or participate in, virtual collections called *virtual domains* (see Figure 2) to realize a capability, as in *service groups*, or to share a collective context or manageability framework, as in *virtual organizations*.

OGSA services require and assume a *physical environment* that may include well-known physical components and interconnects such as computing hardware and networks, and perhaps even physical equipment such as telescopes.

It is expected that there may be a core set of non-null interfaces, standards and common knowledge/bootstrap that services must implement to be part of an OGSA Grid. This set of common implementations and manifestations to support OGSA is referred to as the *infrastructure services* or the *Grid fabric*. As noted in the next section (§3.3), we assume that the Web Services

Resource Framework (WS-RF) specification, currently under development, will be part of the initial Grid fabric. We describe additional standards that could be a part of the Grid fabric in §3.3.

- **Uses**: Capabilities from the target are used by the source to provide a service to a client
- **Composes**: Service combines the capabilities of the underlying services to provide a higher level capability
- **Delegates**: Service passes on the request for “service” to related service for fulfillment
- **Refers**: Service contacts the related service for substantiating a request (validation, concretization)
- **Extends**: Service subsumes and augments the capabilities of the related service

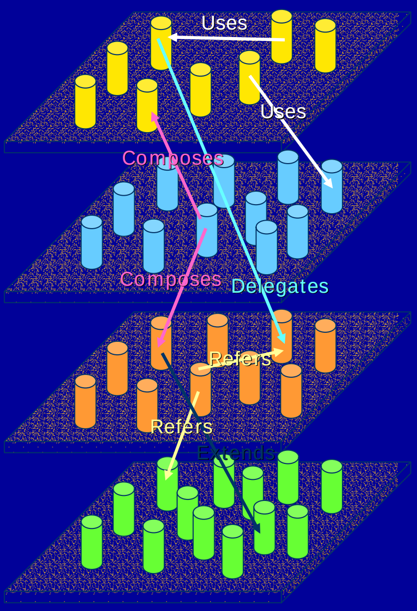

Figure 3: Service Relationships

There are a number of possible relationships and interactions between OGSA services. Some examples of the types of relationships that services may exhibit are shown in Figure 3. Aspects of service interactions for other purposes such as management/manageability, declarative specification of service profiles etc. can also be modeled with relationships.

3.3 Infrastructure Services

Our goal in defining OGSA is to define a coherent and integrated set of components that collectively address the requirements identified in §2, within the context of a service-oriented architecture. We must necessarily make assumptions about the infrastructure on which we build if we are to make concrete statements about higher-level services. There are many examples of specifications that tried to be too abstract and did not achieve their goals. For example, CORBA 1.1 failed to achieve interoperability because service names depended on implementation. Thus, just as the designs of TCP, DNS, and other higher-level TCP protocols and services are informed by the properties of the IP substrate on which they build, so our design of OGSA is influenced by our choice of underlying mechanisms. Here we list, and to some extent justify, those choices.

The primary assumption is that work on OGSA both builds on, and is contributing to the development of, the growing collection of technical specifications that form the emerging Web services (WS) architecture [WS-Architecture]. Indeed, OGSA can be viewed as a particular profile for the application of core WS standards. We made this choice because of a strong belief in the merits of a service-oriented architecture and our belief that the Web services architecture is the most effective route to follow to achieve a broadly-adopted, industry-standard service-oriented rendering of the functionality required for Grid systems.

This choice of Web services as an infrastructure and framework means that we assume that OGSA systems and applications are structured according to service-oriented architecture principles, and that service interfaces are defined by the Web Services Description Languages (WSDL). For now, we assume WSDL 1.1, with a move to WSDL 2.0 planned once that latter specification is finalized. We also assume XML as the lingua franca for description and representation (although recognizing that other representations may be required in some contexts, for example when performance is critical) and SOAP as the primary transport binding for OGSA services. In addition, we seek to develop service definitions that are consistent with the interoperability profiles defined through the WS Interoperability (WS-I) process.

While thus working within a Web services framework, we do not take it as a given that Web services standards as currently defined meet all Grid requirements. In some cases, existing specifications may require modification or extension. Thus, OGSA architects have been involved in the definition of WSDL 2.0 and in the review of WS-Security and related specifications, and we identify in this document other areas in which extensions to existing specifications are desirable. In other cases, Grid requirements motivate the introduction of entirely new service definitions.

One key area in which Grid requirements motivate extensions to existing specifications is security. Security issues arise at various layers of the OGSA infrastructure. We use WS-Security standard protocols to permit OGSA service requests to carry appropriate tokens securely for purposes of authentication, authorization, and message protection. End-to-end message protection is required by some scenarios addressed by the OGSA infrastructure, and thus OGSA must also provide for higher-level protection mechanisms such as XML encryption and digital signatures in addition to, or in place of, point-to-point transport-level security, such as TLS and IPSec. In addition to message-level security, an interoperable and composable infrastructure needs security components to be themselves rendered as services. There are various efforts underway to specify service definitions for these security services. For example, an OGSA authorization service may use the proposed WS-Agreement standard along with evolving OASIS [OASIS] standards including SAML and XACML to express security assertions and access control descriptions. When and where appropriate, OGSA will adopt, or define, those security services.

A key area in which Grid requirements have motivated new specifications—specifications that have relevance beyond Grid scenarios—is in the area of state representation and manipulation. In particular, we assume that we have available as building blocks the interfaces and behaviors defined by the WS Resource Framework (WSRF) [WS-RF], the recently-proposed refactoring of the Open Grid Services Infrastructure (OGSI) [OGSI]. WSRF defines an approach to modeling, accessing, and managing state; to grouping services; and to expressing faults. These mechanisms (defined after an 18-month design process within the GGF OGSI-WG) all have a fundamental role to play in constructing Grid systems. In addition, WSRF is being strongly considered for use in a variety of resource modeling and systems management standards efforts, such as the OASIS WSDM Technical Committee. Thus, OGSA-related standards that build on WSRF are likely to be well-positioned for composition with efforts from these standards bodies.

Finally, we assume notification or eventing capabilities such as those defined within the recently-proposed WS-Notification specifications [WS-N]. These specifications, derived like WSRF from OGSI, define notification mechanisms that build on WSRF mechanisms to support subscription to, and subsequent notification of changes to, state components. (The WS-Eventing specification provides similar mechanisms, but does not yet connect to WSRF.)

3.4 Execution Management Services

Execution Management Services (OGSA-EMS) are concerned with the problems of instantiating and managing tasks. Within OGSA-EMS a task refers to a single unit of work to be managed – for example a legacy batch job, a database server, a servlet running in a Java application server container, etc.

3.4.1 Objectives

The following example illustrates some of the issues to be addressed by EMS. An application needs a cache service. Should it use an existing service or create a new one? If it creates a new service, where should it be placed? How will it be configured? How will adequate resources (memory, disk, CPU) be provided for the cache service? What sort of service agreements can it (the cache service) make? What sort of agreements does it require? Similarly, a user wants to run a legacy program, e.g., BLAST. Where will it run? How are the data files and executables staged to the execution location? What if it fails? Will it be restarted, and if so how?

More concretely EMS addresses problems of placing, "provisioning," and lifetime management of tasks. These include, but are not limited to:

- Where can a task execute? What are the locations at which it can execute because they satisfy *resource* restrictions such as memory, CPU and binary type, available libraries, and available licenses? Given the above, what *policy* restrictions are in place that may further limit the candidate set of execution locations?
- Where should the task execute? Once it is known where the task *can* execute, the question is where *should* it execute? Answering the question may involve different selection algorithms that optimize different objective functions or attempt to enforce different policies or service level agreements.
- Prepare the task to execute. Just because a task can execute somewhere does not necessarily mean it can execute there without some setup. Setup could include deployment and configuration of binaries and libraries, staging data, or other operations to prepare the local execution environment to execute the service.
- Get the task executing. Once everything is ready, actually start the task and register it in the appropriate places.
- Manage (monitor, restart, move, etc.). Once the task is started in must be managed and monitored. What if it fails? Or fails to meet its agreements. Should it be restarted in another location? What about state? Should the state be "checkpointed" periodically to ensure restartability? Is the task participating in some sort of fault-detection and recovery scheme?

These are the major issues to be addressed by EMS. As one can see, it covers the gamut of tasks, and will involve interactions with many other OGSA services that will not be defined in EMS (e.g., provisioning, logging, registries, security, etc.).

Why is EMS important? Why not just assume a static task set and use registries such as UDDI? We expect Grids to be used in a large number of settings where the set of available resources, and the load presented to those resources, are highly variable and require high levels of dependability. For example, in any dynamically provisioned computing environment, the set of resources in use by an application may vary over time, and satisfying the application requirements and service level agreements may require temporarily acquiring the use of remote resources. Similarly, to respond to unexpected failures and meet service level guarantees may require finding available resources and restarting tasks on those resources. The common theme is the need to dynamically instantiate new task instances in response to application needs, and to monitor them throughout their lifetimes.

3.4.2 Approach

The solution consists of a set of services that decompose the EMS problem into multiple, replaceable plug points. Different use cases may use different subsets of these services to realize their objectives. In general, though, one can think of EMS as consisting of a supply side and a demand side. Suppliers *provide* resources – CPU, disk, data, memory, and services. Consumers *use* those resouces. On the demand side are tools such as workload management systems, workflow systems, and so on. On the supply side are services that manage and supply resources – and enforce policy and service agreements. Figure 4 illustrates a generic framework.

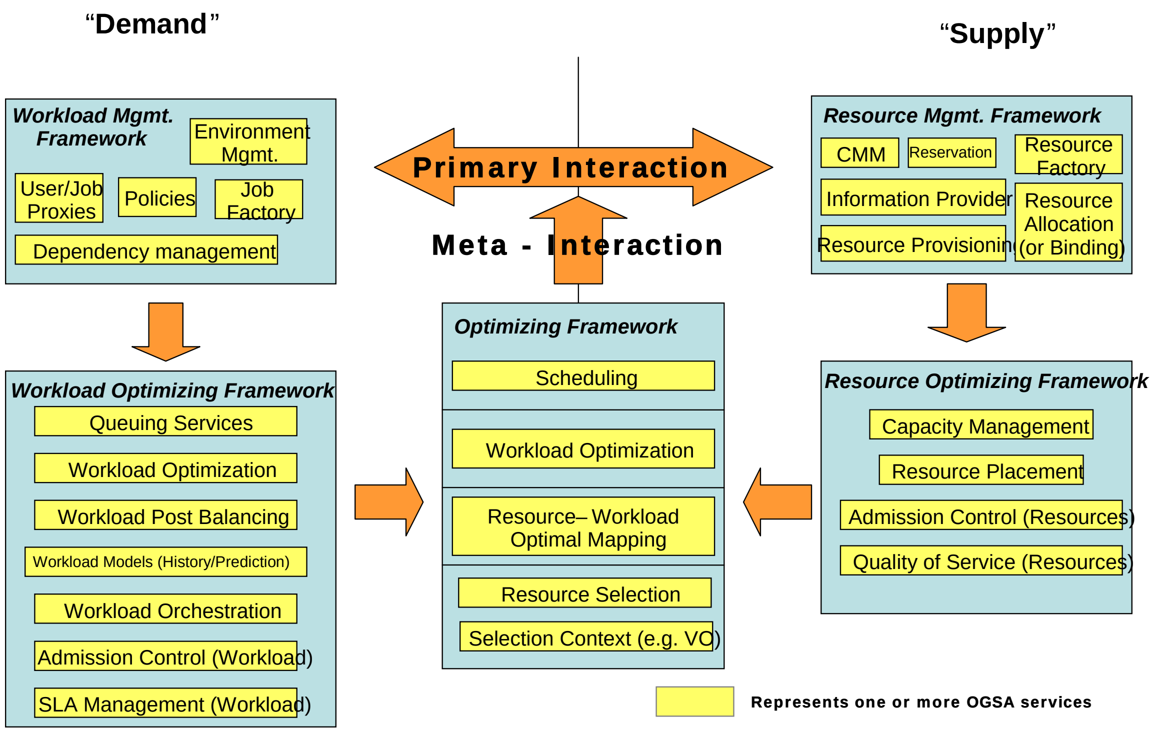

Figure 4: Notional meta-model for EMS divides the world into a supply side and a demand side.

EMS services enable applications to have *coordinated* access to underlying resources, regardless of their physical locations or access mechanisms. EMS services are the key to making resources easily accessible to end-users, by automatically matching the requirements of a Grid application with the available resources.

EMS consists of a number of services working together. Below we describe these services. Before we proceed, though, a few caveats and comments are in order.

First, not all services will be used all of the time. Some Grid implementations will not need some of the services – or may encapsulate some services within other services and not make them directly available. In general, we have tried to factor the services around those functions that we have seen again and again in different Grid implementations – in other words the common functions that not only does one normally need to use, but also that are plug-points where one or more different implementations may be desirable.

Second, this is the first pass at the definitions. It is not our objective in this document to completely define the services. Rather it was our intention to identify key components and their higher level interactions.

Third, we want to emphasize that these definitions and services will be applicable to general Web service execution – not just to the execution of legacy "jobs."

One final assumption: In this document we assume the existence of a "resource handle." A resource handle is an *abstract name* (see §3.9.4.1 and §3.4.7.2) for a resource and its associated state, if any. We also assume that a mechanism exists (defined outside the scope of this document) that binds a resource handle to a "resource address," where a resource address contains protocol-specific information needed to communicate with the resource. We will use *RH* to denote a resource handle, and *RA* to denote a resource address.

3.4.3 EMS Services

There are three broad classes of EMS services:

- Resources that model processing, storage, executables, resource management, and provisioning;
- Job management and monitoring services; and
- Resource selection services that collectively decide where to execute a task.

We also assume the availability of data management services (§3.4); security services (§3.6); and logging services (§3.8.3.3).

3.4.4 Resources

3.4.4.1 Service Container

A service container, hereafter just a container, "contains" running tasks, whether they are "jobs" (described later) or running Web services. A container may, for example, be a queuing service, a Unix host, a J2EE hosting environment, or a collection of containers (a façade or a VO of job containers). Containers have attributes (metadata) that describe both static information such as what kind of executables they can take, OS version, libraries installed, policies in place, security environment/QoS, etc., as well as dynamic information such as load, QoS issues, etc.

A container implements some subset of the manageability interfaces of a WSDM managed resource. Extended interfaces that provide additional services beyond the basic service container are expected.

Containers will have various relationships to other resources that will be exposed to clients. For example, a container may have a "compatibility" relationship with data containers that indicates that tasks running "in" a container can access persistent data "in" a particular data container. Similarly other managed resources might be a deployed operating system, a physical network, etc.

The relationships with other resources are critical. We expect that sets of managed resources will be composed into higher-level services – for example a "container" may be extended to a host-container, for example, that includes a "container", a persistent state handle service (see below), an OS, etc.

Similarly we expect containers to use reservation services, logging services, information services, job management services, and provisioning services.

3.4.4.2 Persistent State Handle Service (PSHS)

A Persistent State Handle Service (PSHS) keeps track of the "location" of persistent state for tasks. It may be implemented many different ways: by a file system, by a database, by a hierarchical storage system, etc. A PSHS has methods to get a "resource handle" (RH) to persistent state that it is managing. The form of the RH depends on how the state is actually stored. A persistent state "resource address" (RA) may be a path name in a file system or a primary key value in a database. The important notion is that the RA can be used to directly access the data.

A PSHS implements the manageability interfaces of a WSDM managed resource. Extended interfaces that provide additional services beyond the basic data container are expected. PSHSs also have methods for managing their contained RHs, including passing it to other PSHSs. This facilitates both migration and replication.

Another way to think about a PSHS is that it is a metadata repository that provides information on how to get to the data efficiently using native mechanisms, e.g., a mount point, a database key, or a path.

Note that a PSHS is *not* a data service. Rather it is a means of keeping track of where the state of a task is kept so that it can be accessed quickly if necessary.

3.4.5 Job Management

3.4.5.1 Job

A job is a resource (named by a distinct resource handle (RH)) – it is created at the instant that it is requested, even though at that point no resources have been committed. The job encapsulates all there is to know about a particular instance of a running application (such as BLAST) or service. Jobs are *not* workflows or array jobs. A job is the smallest unit that is managed. The job represents the manageability aspect of a task and is not the same as the actual running application, or the execution aspect of the task. The job keeps track of job state (started, suspended, restarted, terminated, completed, etc.), resource commitments and agreements, job requirements, and so on. Many of these are stored in a *job document.*

A *job document* describes the *state* of the job – e.g., the submission description (JSDL [JSDL]), the agreements that have been acquired, its job status, metadata about the user (credentials etc.), and how many times the job has been started. We do *not* include in "state" application-specific details such as the internal memory of the executing application program.

The job document is exposed as a resource property of the job. The logical view is of one large document that consists of one or more – possibly many – subdocuments. These subdocuments can be retrieved independently. The organization of the subdocuments will be subject to further specification.

3.4.5.2 Job Manager

The Job Manager (JM) is a higher-level service that encapsulates all of the aspects of executing a job, or a set of jobs, from start to finish. A set of job instances (complex job) may be structured (e.g., a workflow or dependence graph) or unstructured (e.g., an array of non-interacting jobs). Similarly the JM may be a portal that interacts with users and manages jobs on their behalf.

The JM will likely interact with an Execution Planning Services (see §3.4.6.1), the deployment and configuration system, containers, and monitoring services. Further, it may deal with failures and restarts, it may schedule them to resources, and it may collect agreements and reservations.

The JM is likely to implement the manageability interfaces of a WSDM collection, which is a collection of manageable entities. A WSDM collection can expose as its methods some of the methods exposed by the members of its collection.

The JM is responsible for orchestrating the set of services to start a job or set of jobs, e.g., negotiating agreements, interacting with containers, monitoring and logging services, etc. It may also aggregate job resource properties from underlying related job instances.

Examples of JMs are:

- A "queue" that accepts "jobs", prioritizes them, and distributes them to different resources for computation. (Similar to JobQueue [JobQueue] or Condor [Condor]). The JM would track jobs, may prioritize jobs, and may have QoS facilities, a maximum number of outstanding jobs, and a set of service containers in which it places jobs.
- A portal that interacts with end-users to collect job data and requirements, schedule those jobs, and return the results.
- A workflow manager that receives a set of job descriptions, QoS requirements, their dependence relationships, and initial data sets (think of it as a data flow graph with an initial marking), and schedules and manages the workflow to completion – perhaps even through a number of failures. (In this case a node could be another workflow job manager). (Similar in concept to parts of DAGman [DAGman]).
- An array job manager that takes a set of identical jobs with slightly different parameters and manages them through completion. (e.g., Nimrod).

3.4.6 Selection Services

3.4.6.1 Execution Planning Services (EPS)

An Execution Planning Service (EPS) is a service that builds mappings called "schedules" between jobs and resources. A schedule is a mapping (relation) between services and resources, possibly with time constraints. A schedule can be extended with a list of alternative "schedule deltas" that basically say "if this part of the schedule fails, try this one instead."

An EPS will typically attempt to optimize some objective function such as execution time, cost, reliability, etc. An EPS will not enact the schedule; it will simply generate it. The enactment of a schedule is typically done by the JM. An EPS will likely use information services and Candidate Set Generators (CSG, see below). For example, first call a CSG to get a set of resources, then get more current information on those resources from an information service, then execute the optimization function to build the schedule.

3.4.6.2 Candidate Set Generator (CSG)

The basic idea is quite simple: determine the set of resources on which a task can execute – i.e., "where is it *possible* to execute?", rather than "where *will* it execute?" This may involve issues such as what binaries are available, special application requirements (e.g., 4GB memory and 40GB temporary disk space, *xyz* library installed), and security and trust issues ("I won't let my job run on a resource unless it is certified Grade A+ by the Pure Computing Association," or "they won't let me run there until my binary is certified safe," or "will they accept my credit card?").

A Candidate Set Generator (CSG) generates a set of containers (really their RHs) in which it is possible to run a job named by a RH. The set of resources to search over may either be a default for the particular service or be passed in as a parameter.

We expect CSGs to be primarily called by EPSs, or by other services such as JMs that are performing EPS functions. We expect CSGs to use information services, to access jobs to acquire appropriate pieces of the job document, and to interact with provisioning and container services to determine if it is possible to configure a container to execute a particular task.

3.4.6.3 Reservation services

Reservation services manage reservations of resources, interact with accounting services (there may be a charge for making a reservation), revoke reservations, etc. This may not be a separate service, rather an interface to get and manage reservations from containers and other resources. The reservation itself is likely to be an agreement document that is signed.

A reservation service presents a common interface to all varieties of reservable resources on the Grid. Reservable resources could include (but are not limited to) computing resources such as CPUs and memory, graphics pipes for visualization, storage space, network bandwidth, special-purpose instruments (e.g., radio telescope), etc.

A reservation could also be an aggregation of a group of lower-level reservations, as might be negotiated and “resold” by a broker of resources.

Reservation services will generally be used by many different services: a JM might create reservations for the groups of jobs which are being managed, or an EPS might use reservations in order to guarantee the execution plan for a particular job. It could also be the case that the creation of reservations will be associated with the provisioning step for a job.

3.4.7 Interactions with the rest of OGSA

This section details the interactions between the EMS and other parts of OGSA.

3.4.7.1 Deployment & Configuration Service

Often before a task can execute in a container the service container and/or data container must be configured or provisioned with additional resources. For example, before running BLAST on a host, a user must ensure that the BLAST executable and its configuration files are accessible to the host. A more in-depth example is the configuration of a complex application and installation of appropriate databases, or installing Linux on a host as a first step to using the host as a compute resource.

3.4.7.2 Naming

OGSA-EMS uses OGSA-naming, see §3.9.4.1. For example, in a sophisticated job queuing system which has checkpoint and restart feature for availability or load balancing purpose, an ***address*** of the job may specify the location of a job on a particular machine. The ***abstract name*** will identify the job in a location-independent but universal way, for example the ***abstract name*** should be the same before and after the job migration. The ***human-oriented name*** may be a user-friendly short job name that can be disambiguated by referring to the context in which it is used.

3.4.7.3 Information Service

The basic idea is simple: information services are databases of attribute metadata about resources. Within EMS, information services are used by many of the different services: for example, containers need to publish information about their attributes so that CSG services can evaluate the suitability of a container for a job; an EPS might read policy information for a VO from an information service; and the PSHS itself could be implemented using information services. How the information service gets its information is unspecified, although we expect “freshness” to be an attribute on data. In this sense, OGSA information services are similar to MDS services in Globus [Globus MDS] and collections in Legion [Legion].

3.4.7.4 Monitoring

Simply starting something up is often insufficient. Applications (which may include many different services/components) often need to be continuously monitored, for both fault-tolerance reasons and QoS reasons. For example, the conditions on a given host that originally caused the scheduler to select it may have changed, possibly indicating that the task needs to be rescheduled.

3.4.7.5 Fault-Detection and Recovery Services

Fault-detection and recovery services may or may not be a part of monitoring, and may include support for managing simple schemes for stateless functions that allow trading off performance and resource usage; slightly more complex schemes that manage checkpointing and recovery of single-threaded (process) jobs; and still more complex schemes that manage applications with distributed state, such as MPI jobs.

3.4.7.6 Auditing, billing and logging services

Auditing, logging, and billing services are critical for success of OGSA outside of academia and government. This will include the ability for schedulers to interact with resources to establish prices, as well as for resources to interact with accounting and billing services. Logging is the basis of the whole chain.

- *Metering* is using the log to keep track of resource usage.
- *Auditing* is using the log in persistent fashion, possibly non-repudiation as well.
- *Billing* is yet another service, not defined by OGSA, that may use auditing and/or metering logs and other data to generate bills, or chargeback.

3.4.7.7 Accounting

Like a credit card, some resources need to see if the user has enough credit to pay. The scheduler may need to interact with the accounting services, as may certain resources such as containers.

3.4.8 Example Scenarios

The best way to understand these services is to see how they are used to realize concrete use cases. We have selected three use cases to demonstrate: a system patch tool, deploying a data caching service, and a legacy application execution.

3.4.8.1 Case 1 – System Patch Tool

Often operating system patches or library updates need to be applied to a large number of hosts. This can be done in several ways. One commonly-used technique is to run a script on each host in a system that copies the appropriate files. These scripts are often initiated using tools such as “rsh” or “ssh,” and are called from shell scripts that iterate over a set of hosts to be patched. Alternatively, hosts may periodically check if they need an update, and if so, run some script to update the OS.

Using EMS this can be done in many ways. Suppose the OS version number is a piece of metadata maintained by containers and collected by an information service, and that the objective is to patch all operating systems that don’t have all the patches. Perhaps the simplest way to approach this problem is to first query information services for a list of containers whose OS version number is below some threshold. Then, instruct a Job Manager (JM) to run the patching service on each container in the list. In this case the JM does not need to interact with execution planning services (EPS) because it knows where it wants to run the service. Instead, the JM interacts directly with each container – and possibly a deployment and configuration service, to execute the patching service on the container. (The deployment and configuration service may be needed to install the patch service first.)

3.4.8.2 Case 2 – A Data Cache Service

Imagine a data cache service that caches data (files, executables, database views, etc.) on behalf of a number of clients, and maintains some notion of coherence with the primary copies. When a client requests a cache service, one can either deliver a handle to an existing cache service or create a new cache service, depending on the location of the client, the location of the existing caches, the load on existing caches, etc.

Once a decision has been made to instantiate a new cache service, an EPS is invoked to determine where to place the data cache. The EPS uses CSG to determine where it is possible to run the service – constrained by a notion of locality to the client. Once a location has been selected, the service is instantiated on a container.

3.4.8.3 Case 3 – A Legacy Application

Our third example illustrates a rather typical scenario. Suppose a user wants to run a legacy BLAST job. Further, suppose the user is interacting with a portal or queue job manager. There are four basic phases in getting the BLAST job started:

1. Job definition phase. What are the input files? What are the service level requirements? E.g., job must complete by noon tomorrow. What account will be billed for the job? Etc.
2. Discover the resources available and select the resources required to execute the job.
3. Enact the schedule and all that may be involved, e.g., provisioning of resources, accounting, etc.
4. Monitor the job through its lifetime(s). Depending on the service level agreements the job may need to be restarted if it fails to complete for any reason.

To realize this case using EMS the JM creates a new legacy job with the appropriate job description (written in JSDL [JSDL]). The JM then calls an EPS to get a schedule. The EPS in turn calls a CSG, which calls information services to determine where the job can be executed based on binary availability and policy settings. The EPS selects a service container, after first checking with the service container that the information is accurate. The EPS returns the schedule to the JM. The JM then interacts (if necessary) with reservation and deployment and configuration services to set up the job execution environment. This may involve interaction with the data container as well. The service container is invoked to start the job. Logging services are used for accounting and audit trail. When the job terminates the job manager is notified by the container. If the job terminates abnormally, the whole cycle may repeat again (see Figure 5).

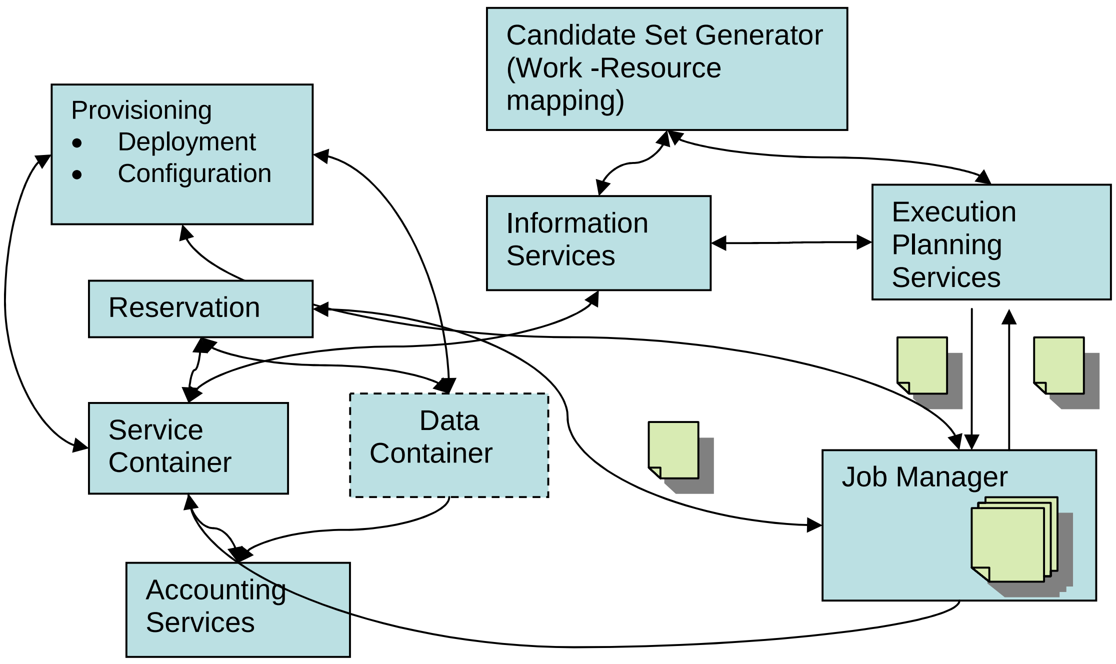

Figure 5: Interactions of EMS services to execute a legacy BLAST job

3.5 Data Services

OGSA data services are concerned with the movement, access and update of data resources.

3.5.1 Objectives

Data services are used to move data to where it is needed, manage replicated copies, run queries and updates, and transform data into new formats. They also provide the capabilities necessary to manage the metadata that describes OGSA data services or other data, in particular the provenance of the data itself.

For example, suppose an Execution Management Service needs to access data that is stored elsewhere. Does it use data services to access the data remotely or to stage a copy to the local machine? Does it cache some of the data locally? Is the data available in multiple locations? What policy limitations or service guarantees does the data service provide?

Conversely, suppose that a data service wishes to provide a virtual schema of data stored in several different places. How should queries against the virtual schema be mapped to the underlying resources? Where should any joins or data transformations be executed? To where should the data be delivered? What quality of service can be guaranteed?

With the exception of certain classes of metadata, the data capabilities do not specify the meaning of any particular data. Other OGSA services (e.g., information services) that use data services may add meaning of their own.

3.5.2 Models

3.5.2.1 Types of Data Resource

A data resource is any entity that can act as a source or sink of data. The heterogeneous nature of the Grid means that many different types of data must be supported. These include, but are not limited to:

- *Flat Files*. The simplest form of data is a file with application-specific structure. For OGSA these files are opaque. Some file formats support database-like queries. Examples include comma-separated values, which can be queried like relational tables, and XML files, which can be queried using XML Query [XQuery]. The data access services support these and can also be extended to support specialised queries over new file formats.
- *Streams*. Potentially-infinite sequences of data values are called streams. The data access services support queries and transformations over streams.
- *DBMS*. Several kinds of database management systems may be part of Grids. These include relational, XML, and object-oriented databases, among others.
- *Catalogues*. A catalogue structures and tracks other data services. A simple example of a catalogue is a directory, which lists a set of files. Nested directories are equivalent to a hierarchic namespace.
- *Derivations*. Some data is the result of asynchronous queries or transformations on other data. These derivations are often managed like finite streams rather than single items.
- Data Services themselves can be data resources for other services, as can be sensor devices or programs that generate data.

3.5.2.2 Example Scenarios

Data service capabilities are fundamental to the Grid. This section describes a few examples of how they can be used to support a range of activities.

- *Remote access.* The simplest use of the OGSA data services is to access remote data resources across the Grid. The services hide the communication mechanism from the client; if necessary they can also hide the exact location of the remote data. An optimisation of this is to cache some of the data in a local resource.
- *Staging*. When jobs are executed on a remote resource, the data services are often used to move input data to that resource ready for the job to run, and then to move the result to an appropriate place.
- *Replication*. To improve availability and to reduce access times, the same data can be stored in multiple locations across the Grid.
- *Federation*. The OGSA data services allow the creation of a virtual data resource that incorporates data from multiple data sources that are created and maintained separately. When a client queries the virtual resource, the query is compiled into sub-queries and operations that extract the appropriate information from the underlying federated resources and return it in the appropriate format.
- *Derivation.* The OGSA data services support the automatic generation of one data resource from another.
- *Metadata.* Data that describes OGSA data services or other data is fundamental to the Grid. In the simplest form, this is just another use of the OGSA data services. In addition, OGSA provides support for maintaining links between data and metadata.

3.5.2.3 Supply & Demand Meta Model

As with other parts of OGSA, the data service architecture can be seen as composed of suppliers and consumers. Suppliers provide data resources – files, databases, streams and services. On the demand side are services that virtualize data resources, manage distributed queries, manage service level agreements, and so on. On the supply side are services that cache, replicate and manage data, and enforce policy and service agreements. This can be seen in Figure 6 below, which illustrates a generic framework.

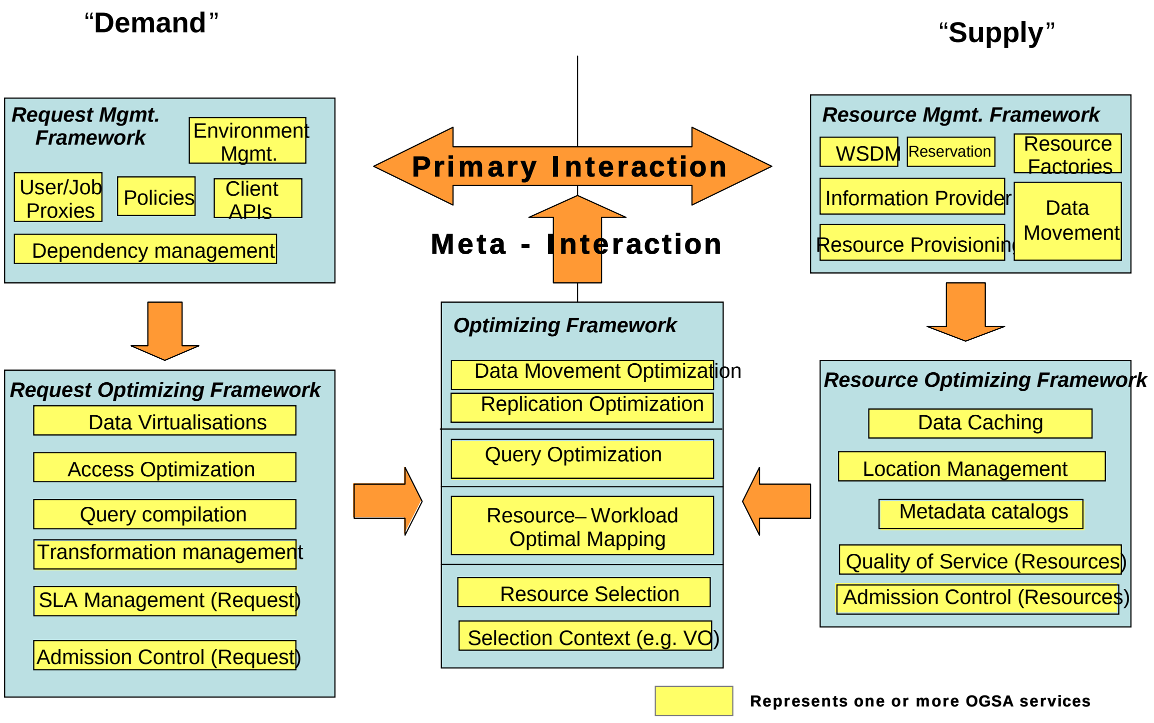

Figure 6: Notional meta-model for Data divides the world into a supply side and demand side.

3.5.3 Functional Capabilities

This section describes the functional capabilities provided by the OGSA data services. Different subsets of the services are needed to implement the different capabilities. Implementations are free to provide only some of these capabilities.

3.5.3.1 Transparency, Virtualization and Layered Architecture

A distributed system may contain a variety of data resources. These resources may use different models to structure the data, different physical media to store it, different software systems to manage it, different schema to describe it, and different protocols and interfaces to access it. The data may be stored locally or remotely; may be unique or replicated; may be materialized or derived on demand. The OGSA data services can define virtualisations over these data resources. Virtualisations are abstract views that hide these distinctions and allow the data resources to be manipulated without regard for them.

Conversely, although the data services allow the clients to ignore these distinctions, some clients may prefer to exploit them. For example, a client may wish to make use of a particular query language for a given database, or to specify the location of the particular data resource to use. One client may require native access to the data, while another needs to tune the performance parameters of the data resource. To support such clients, the OGSA data services allow clients to bypass the virtualisation interfaces and access the resource-specific interfaces directly. This layered architecture allows clients to choose the combination of power and abstraction that suits them best.

3.5.3.2 Client APIs

Many users will wish to use the OGSA data services with legacy applications that use existing APIs such as NFS, CIFS, JDBC, ODBC, ADO, POSIX IO or XQuery. Often it will be too expensive or impractical to rewrite the clients to use new interfaces. However, the OGSA services can easily be wrapped with a veneer that emulates these existing APIs, although OGSA itself will not define such wrappers.

Typically these wrappers will map the operations of the existing APIs to the corresponding messages to send to the OGSA data services. Therefore they may provide access to only a subset of the full OGSA functionality. The extent of the OGSA functionality that they make available will depend on the scope of the API concerned.

3.5.3.3 Extensible data type support and operation

It is not possible to predict all the data resources with which the OGSA data services will be used. In any case, new data resources will be created in the future. The layered architecture described above allows the addition of new data services that access new types of resources.

Similarly, it is not possible to anticipate all the operations that may be needed on a given resource. The layered architecture supports services that provide additional operations beyond those specified in the OGSA document. The virtualisation layer gives clients the option of ignoring these extensions, while those clients that require them can bypass the virtualisation layer as necessary.

3.5.3.4 Data Location Management

The OGSA data services offer reliable data transfer from one location to another. This may be to create a copy of the original or to migrate the original completely. Data may be cached at a given location to avoid unnecessary additional transfers. Data caches can be configured in terms of, for example, lifetime and update consistency. Data may also be replicated between multiple copies, thus increasing availability through redundancy.

Services for individual users allow them to upload and manage their own data with the above facilities. The security settings control the access permitted to other users.

3.5.3.5 Queries

The OGSA data access services provide mechanisms for querying data resources. In simple cases these may run an SQL query over a relational database, an XML query over an XML database, or a regular expression over a text file. Other services may implement text mining over a set of documents, or distributed queries over federated databases.

Synchronous queries return the data in the response to a request, while asynchronous queries expose the derived data as new resources. Services may also deliver the results of a query to a specified set of other services.

A distributed query processor will analyse each query and create sub-queries to be run on distributed data resources. It may also determine where intermediate processing is done in order to minimise network traffic. As such distributed query processors provide all necessary information to workflow enactment engines to work efficiently.

3.5.3.6 Transformation

Data services may themselves transform data. For example, they may convert data from one format to another, or filter it, before moving it or updating it. They may support stored procedures that execute within the service, making the service a form of container. These transformations may be instigated explicitly by certain operations, or they may be programmed to be triggered automatically in response to certain conditions.

3.5.3.7 Data Update

OGSA data services provide a range of mechanisms for updating data resources, depending on the semantics of the data resource. For catalogues, the operations include creation, renaming and deletion. For structured files and databases the operations include the update of entries. For streams and other files the operations are largely limited to appending new data.

When a data resource has replicated versions or is the source for derived data services, the updates may be propagated to the replicated or derived versions. In this case, and in the case where several clients are updating the same data resource, the services may implement various forms of consistency maintenance, e.g., to ensure that a client always sees the results of its own updates in any queries that it issues itself.

Update operations may be part of transactions. Transaction support is provided by other OGSA services.

3.5.3.8 Security Mapping Extensions

Database management systems often implement sophisticated security mechanisms. Some of these provide a large range of possible operations and access control at the level of individual tuples. Therefore the OGSA data services support extensions to the standard OGSA security infrastructure to allow users, operators and applications to access the greater control provided by such systems.

3.5.3.9 Data Resource Configuration

Data resources often provide sophisticated configuration options. These can be made available to clients via the OGSA data services. In addition, the services may provide additional operations for configuring the virtualisation of the resource provided by the service. For example, a relational data service might allow for specific tables from an underlying database resource to be made part of the data service's data virtualization, or for views on the underlying database to be made available as tables within that virtualization.

3.5.3.10 Metadata

Metadata services are data services that store metadata about OGSA data services or other data. OGSA provides support for maintaining links between OGSA services and the metadata that describes them. This support includes provision for maintaining the consistency of the metadata.

The metadata for OGSA data services may include information about the structure of the data, including references to the schemas that describe the data. For some services this is not practical, as the data resources include many schemas that are modified frequently, and in these cases schema information will be provided by the services themselves.

3.5.3.11 Provenance

Metadata for data services may also include information about the provenance and quality of the data. This may be at the level of the whole resource or of its component parts, sometimes to the level of individual elements. This in turn requires the services or other processes that generate the data to also maintain the consistency of the metadata. Complete provenance information can allow the data to be reconstructed by following the workflow that originally created it.

3.5.4 Properties

Properties are non-functional capabilities. They are aspects of the architecture that apply across a range of services. Whereas functional capabilities are defined by entries in the service interfaces, non-functional properties have a more global, semantic, role.

3.5.4.1 Scalability

The OGSA data services handle large scale in several dimensions, including size of data sets, number of data sets, size of data flows, and number of sites.

3.5.4.2 Quality of Service

The OGSA data services can implement various levels of QoS, such as guaranteed delivery and referential integrity.

3.5.4.3 Coherency

When data is replicated, cached or derived, a range of coherency operations are available. Similarly, a range of different mastering and peering strategies are supported for the deployment of metadata catalogues, and the resolution of conflicting updates.

3.5.4.4 Performance

The OGSA data services are designed to minimize the copying and movement of data to the minimum. This is a key factor in the overall performance of the Grid.

The data transfer services use monitoring information such as bandwidth, utilisation patterns and packet size. This enables them to choose the best approach for moving a given data set to suit the agreed quality of service.

3.5.4.5 Availability

The OGSA data services provide features for graceful degradation in the event of network or other failures. For example, query services may be configured to return partial results when only a subset of sources is available.

3.5.4.6 Legal and Ethical Restrictions

The OGSA data services may have to operate within an environment where a variety of legal and ethical policies affect their operation. For example, some policies may restrict the entities that can access personal data and limit the operations that they can perform (confidentiality). Privacy concerns may limit the queries that can be made about individuals. In some cases the policies may permit other queries that return results about a group as a whole, such as average income or total salary.

Much data is covered by copyright limitations. This restricts the copies that can be created of the data. In the European Union, the similar “Database right” applies specifically to databases. The security mechanisms provided by OGSA allow these restrictions to be specified, but care must be taken when using the data services.

3.5.5 Interactions with the rest of OGSA

This section summarizes the interactions between the data services and the other parts of OGSA.

3.5.5.1 Transactions

Data services are the classic example for transactions, and many data resources provide independent transactional support. OGSA allows data services to be part of transactions. Support is provided for conventional ACID transactions and also for two-phase commit which implements synchronization for distributed databases. In addition, "time warp" co-ordination, in which services can execute speculatively and roll-back their execution in the event of a transaction being aborted, can also be supported.

3.5.5.2 Logging

Similarly to other OGSA services, data services use logging services, for example for audit. Conversely, the logging services themselves may use the data services to store the logs.

3.5.5.3 Execution Management Services

Data services have a close relationship with Execution Management Services (OGSA-EMS). OGSA-EMS uses data services in order to stage data where it is needed. The OGSA-EMS Execution Planning Service has to take into account the costs of accessing the data from candidate computational resources.

3.5.5.4 Workflow

Workflows can instruct data access services to send query results to third parties and to apply transformations as the data is moved. This task requires that the workflow services have intimate knowledge of the capabilities of the data access services. Also, a call to a distributed query service may cause a variety of data movements and operations on other machines. For these reasons a data service can provide information to the workflow manager to ensure that the workflow enactment takes full account of the effect of distributed queries on the network and other resources.

3.5.5.5 Provisioning

In addition to the provisioning of storage space and of services themselves, data services also require provisioning support for uploading data sets to data resources. Some may also require support for uploading filters and cutters to a given data service.

3.5.5.6 Resource Reservation

Data services may need to reserve certain resources in order to operate. For example, a file transfer will require storage space and network bandwidth, while a distributed query system may additionally require compute power in order to perform join operations.

3.5.5.7 Discovery

The data services may use the discovery services not just for registering services themselves, but also for registering the data sets that are stored by those services. They may also register the locations of schema definitions.

3.5.5.8 Security

Security services support the mapping of OGSA identities and roles to resource-specific identities and roles. VO management services similarly provide control of these mappings. Security services also provide support for checking the integrity and encryption of data transfers.

3.5.5.9 Network management

Network management can be crucial when planning the transfer of large amounts of data. It may be necessary to configure the network parameters to suit the amount of data to be transferred and the time constraints specified on those transfers.

3.5.5.10 Naming

The OGSA data services use OGSA-naming (§3.9.4.1) for naming data sets. For example, in a replicated file system, an ***address*** may specify the location of a file on a particular machine. The ***abstract name*** will identify the file in a location-independent manner, perhaps using a directory service. The ***human-oriented name*** may be a user-friendly short file name that can be disambiguated by referring to the context in which it is used.

3.5.5.11 Notification

We can use the notification service to externalise database triggers. Database management systems often provide a mechanism that specifies actions to be taken when certain conditions (triggers) are met. OGSA can easily cause these triggers to invoke the notification services.

In addition, some implementations of the notification services may use the data services to store the notification messages and support client queries.

3.6 Resource Management Services

3.6.1 Objectives

Resource management performs several forms of management on resources in a Grid. In an OGSA Grid there are three types of management [OGSA RM] that involve resources:

- Management of the resources themselves (e.g., rebooting a host, or setting VLANs on a network switch)
- Management of the resources on Grid (e.g., resource reservation, monitoring and control)
- Management of the OGSA infrastructure, which is itself composed of resources (e.g., monitoring a registry service)

3.6.2 Model

Different types of interfaces realize the different forms of management in an OGSA Grid. These interfaces can be categorized into three levels, shown in the middle column of Table 2, and also on the right in Figure 7.

Table 2: Relationships between types of management and interfaces

Type of management	Level of interface	Interface
Management of the resources themselves	Resource level	CIM, SNMP, etc.
	Infrastructure level	WSRF, WSDM, etc.
Resource management on the Grid	OGSA functions level	Functional interfaces
Management of OGSA infrastructure		Specific manageability interfaces

We provide below a detailed description of each level and its interfaces. Note that the descriptions focus on the manageability interfaces, *not* on the implementation (e.g., on the services that implement them). Also note that a service may implement multiple interfaces (which are possibly unrelated in terms of functionality), and that a service may be separated from the functionality that it represents (e.g., a manageability provider for a resource that is separate from this resource). Therefore a description based on services would be imprecise, and a description based on interfaces is chosen instead.

In Figure 7, the OGSA capabilities cover all levels, extending to capabilities in the resources that are needed to implement these OGSA capabilities. The interfaces are shown as small circles.

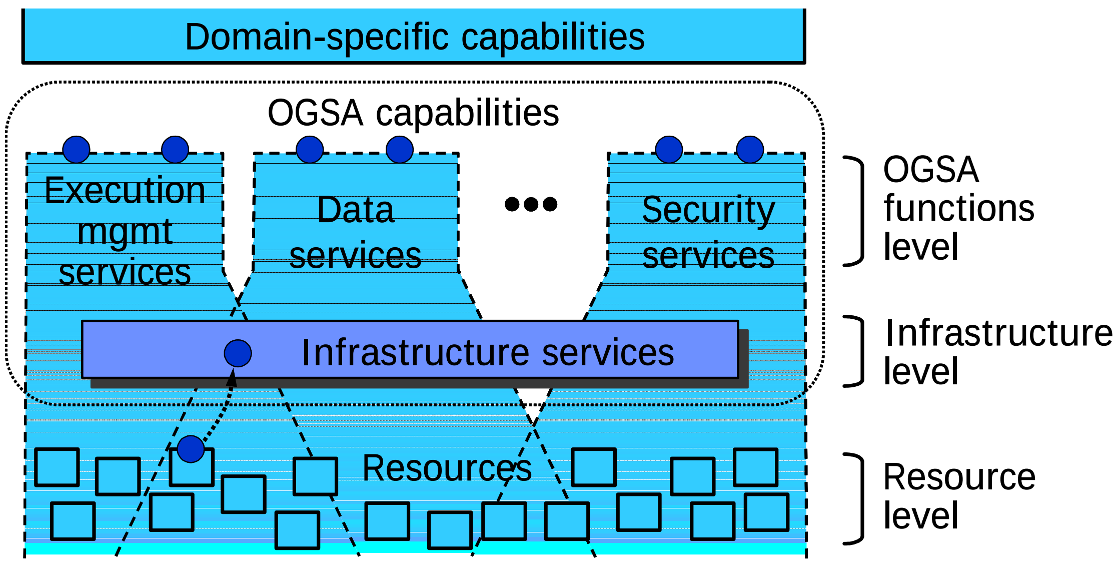

Figure 7: Levels of management in OGSA

At the resource level, resources are managed directly through their native manageability interfaces (for discrete resources, these are usually SNMP, CIM/WBEM, JMX, or proprietary interfaces). Management at this level involves *monitoring* (i.e., obtaining the state of the resource, which includes events), *setup and control* (i.e., setting the state of the resource), and *discovery*.

The infrastructure level provides the base management behavior of resources, forming the basis for both manageability and management in an OGSA environment. Standardization of this base management behavior is required in order to integrate the vast number and types of resources—and the more limited set of resource managers—that are introduced by multiple suppliers. The infrastructure level provides:

- A *base manageability model*, which represents resources as WS-resources [WS-RF] and allows resources in OGSA to be manipulated through the standard Web services means for discovery, access, etc. This model allows the resources to become manageable, at least to a minimum degree, by enabling discovery, termination, introspection, monitoring, etc.
- A *generic manageability interface* that is common to all services implementing OGSA capabilities. This manageability interface has functionality such as introspection, monitoring, and creation and destruction of WS-resources.

At the OGSA functions level, there are two types of management interfaces, denoted by the two circles on the top of each of the capabilities shown in Figure 7:

- *Functional interface*: Some common OGSA capabilities (e.g., OGSA EMS) are a form of resource management. Services that provide these capabilities expose them through functional interfaces (e.g., create and destroy a job).
- *Manageability interface*: Each capability has a specific manageability interface through which the capability is managed (e.g., monitoring of registries or monitoring of a job manager). This interface could extend the generic manageability interface, adding any manageability interfaces that are specific to the management of this capability.

3.6.3 Management Capabilities

Management functionality at the infrastructure level is provided by OASIS WSDM, which is expected to become a key standard for manageability across the IT landscape. WSDM is developing separate documents to address management *of* Web services (MOWS) and Management *using* Web services (MUWS). MUWS provides the base manageability model, i.e., how to represent resources, plus basic manageability functions that are common to the OGSA, such as state representation and operations, and relationships among resources. MOWS provides the generic manageability interface to manage the services in an OGSA Grid.

At the OGSA functions level, the resource management capability includes (but is not limited to) typical distributed resource management activities and IT systems management activities. Functionalities included in the OGSA resource management capability include resource reservation, monitoring and control, VO management, security management, problem determination and fault management, service groups and discovery services, metering, deployment, and discovery.

3.6.4 Properties

3.6.4.1 Scalability

Management architecture needs to scale to potentially thousands of resources. Management needs to be done in a hierarchical and/or peer-to-peer (federated/collaborative) fashion to achieve this scalability, so OGSA should allow these forms of management.

3.6.4.2 Interoperability

Management architecture must be able to span software, hardware and service boundaries, e.g., across the boundaries between different products, so standardized and broad interoperability is essential to avoid “stovepipes.”

3.6.4.3 Security

There are two security aspects in management. *Management of security* concerns the management of the security infrastructure, including the management of authentication, authorization, access control, VOs and access policies. *Secure management* refers to using the security mechanisms on management tasks. Management should be able to ensure its own integrity and to follow access control policies of the owners of resources and VOs.

3.6.4.4 Reliability

A management architecture should not force a single point of failure. For purposes of reliability, a resource may be virtualized by multiple services each exposing a single URL as the management endpoint. In such situations, the system that provides manageability capabilities must be aware that for certain queries, such as metrics, the manageability provider must aggregate the results from the multiple services that virtualize that single resource.

3.6.4.5 Policy

Management must be able to enforce policy assertions that are put in place to support requirements and capabilities such as authentication scheme, transport protocol selection, QoS metrics, privacy policy, etc.

3.6.5 Interactions with other OGSA services

The services in the resource management capability are an enabler for other OGSA capabilities. Thus services in most OGSA capabilities, especially execution services, data services, and self-management services will use functionality provided by the resource management capability.

The resource management capability uses basic services in other OGSA capabilities, such as registries, which are used for resource discovery.

3.7 Security Services

This section describes objectives, models, functional capabilities, and properties of the security services, and discusses the interactions with the rest of OGSA.

3.7.1 Objectives

OGSA security services are to facilitate the enforcement of the security-related policy within a (virtual) organization [Grid Anatomy][Grid Physiology][VO Security].

In general, the purpose of the enforcement of security policy is to ensure that the higher-level business objectives can be met. This is often a delicate balance, as there is an increased cost associated with increased policy enforcement, and an increased loss exposure with less stringent enforcement, while potential profits or rewards may be adversely affected by the increased complexity and inflexibility of stricter policy requirements. Note that in some cases, legislative rules and regulations mandate conformance to associated security policy.

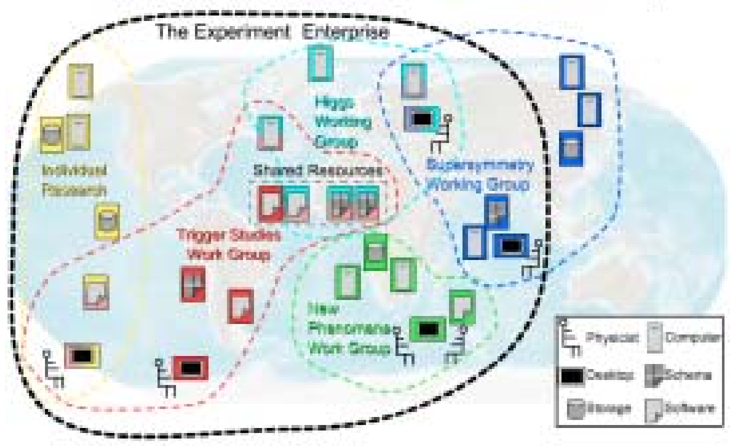

Figure 8: Cross-Organizational Collaborations

The property that Grid-specific applications may span multiple administrative domains (see Figure 8) implies that each of these domains will have its own business objectives to meet, which translates to the individual domains separately establishing and enforcing their own policies, which can differ greatly in complexity and strictness. Note that all interactions associated with a thread of work in a Grid application must adhere to the domain-locally-enforced policies as well as to the policies established for the VO—i.e. the cross-organizational (business) agreement.

To meet the business and associated security objectives, the security policy can, for example, specify the enforcement of:

- Message integrity and confidentiality
- Authentication of interacting entities
- Minimum authentication strength
- Secure logging and audit
- Separation of responsibilities
- Intrusion and extrusion detection
- Authorization policy checks
- Least privilege operations
- Mandatory access control mechanisms
- Discretionary access control mechanisms
- Trust and Assurance level of the environment
- Application Isolation
- Avoidance of DoS attacks
- Redundancy
- Training
- …

OGSA security architectural components must support, integrate, and unify popular security models, mechanisms, protocols, platforms, and technologies in a way that enables a variety of systems to interoperate securely. The components must be able to support integrating with existing security architectures and models across platforms and hosting environments. This means that the architecture must be *implementation-agnostic*, so that it can be instantiated in terms of any existing security mechanisms (e.g., Kerberos [Kerberos V5], PKI [PKI]); *extensible*, so that it can incorporate new security services as they become available [RFC2903][WS Security Whitepaper][Liberty]; and *integratable* with existing security services. Also, services that traverse multiple domains and hosting environments need to be able to interact with each other, thus introducing the need for interoperability at multiple levels: protocol, policies and identity. In addition, certain situations can make it impossible to establish trust relationships among sites prior to application execution. Given that the participating domains may have different security infrastructures (e.g. Kerberos or PKI) it is necessary to realize the required trust relationships through some form of federation among the security mechanisms.

3.7.2 Model

This section describes a model to facilitate the reasoning about, and the specification of, the OGSA security services. The presented model is not a formal model in the mathematical sense, but is meant to provide a language to describe, and to obtain common understanding about, the security policies that are to be enforced.

In general, one can make the observation that entities are interacting through mechanisms within a context. Entities are things such as users, subjects or services. Interaction mechanisms span all the different communication methods, such as mail, telephone, HTTP, SOAP, SSL/TLS, etc. A context puts the interaction in perspective, and could localize the interaction to a single machine, or could be associated with a service invocation within a VO, and/or with an established secure association, and/or with a distributed transaction.

All these entities, mechanisms and contexts can be described by sets of attributes or properties. Some of these attribute types and values may be used for unique identification, others are used for classification or grouping. Furthermore, some of these attributes are *inescapable*. An inescapable attribute can identify an entity by itself without any reference to an outside authority. Examples of inescapable attributes are a shared secret, a private/public key pair, fingerprints, etc. All other attribute values are essentially bound to an entity's inescapable attribute by an issuer or attribute authority.

Security policies are statements about these different entities, interaction mechanisms and contexts, and specify restrictions on the associated attribute values, properties and their relationships. The policy statements (or rules) will be able to be expressed in terms of these entities (e.g. identities, user attributes), resources (i.e. end points), and environment characteristics (e.g. time, location, purpose, or the trust level of the requester or request path). These policies will be about various aspects including authorization, authentication, trust, identity mapping, delegation, assurance levels, etc.

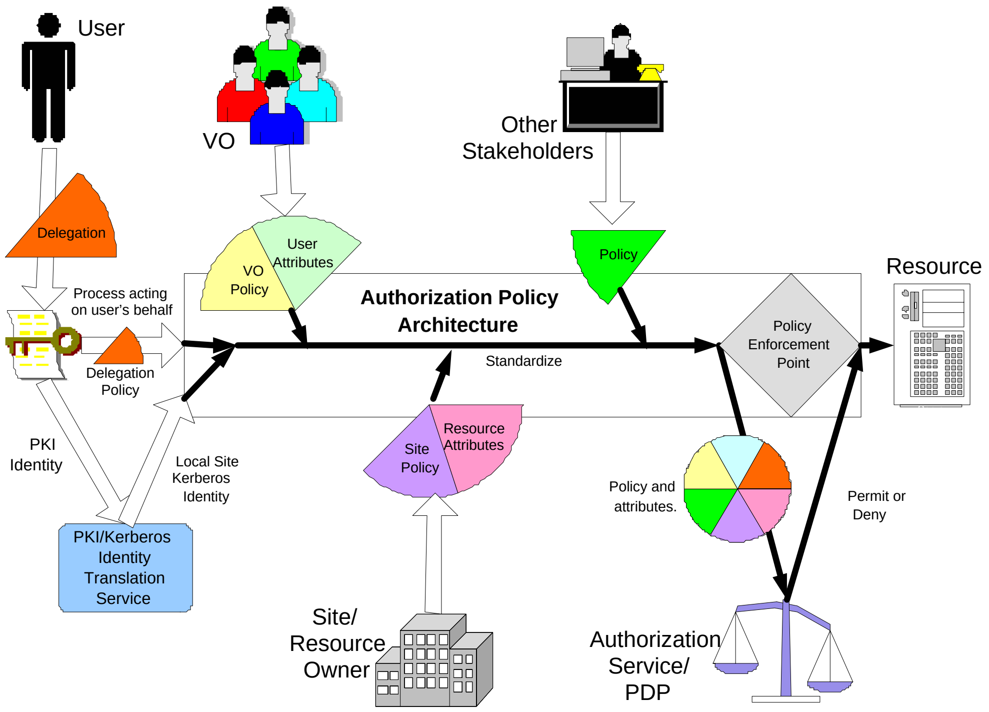

Figure 9: Example input for the policy decision and enforcement

The model defines the security services as entities with interaction patterns that facilitate the administration, expression, publishing, discovery, communication, verification, enforcement and reconciliation of the security policy. In other words, the security policy enforcement is the ultimate goal, and the security services are designed and deployed to support that goal.

To make this model more concrete, we will identity a number of these entities, interaction mechanisms and contexts, and discuss some of their attributes and common relationships. Figure 9 shows an example of the policy enforcement to protect the use of a resource. In order to understand the model, it helps to walk through the example in Figure 9 from the two opposite ends: on one side the initial user authentication, and on the other the enforced authorization, and to realize that a user will only be allowed to access the resource if the authorization policy evaluated at the enforcement point will yield permit. The resource's authorization policy will be expressed as rules for the attribute values of the relevant entities, such as, for example, the user's name, her role, her VO-membership, etc. With the initial authentication of the user, only inescapable attribute values are presented and verified, such as the possession of the private key associated with the presented public key. Normally, there is a mismatch between the fact that the policy is expressed in derived attributes and the initial authentication only yields a key, and this can be resolved by finding the matching attribute assertions that bind the key to the attributes used in the policy expression. Several security services are depicted in Figure 9; these are used to federate the user's public key credentials to Kerberos ones, and to obtain attributes from different sources, such as the VO or the local site, that assert specific attribute bindings used in the enforced policy statements.

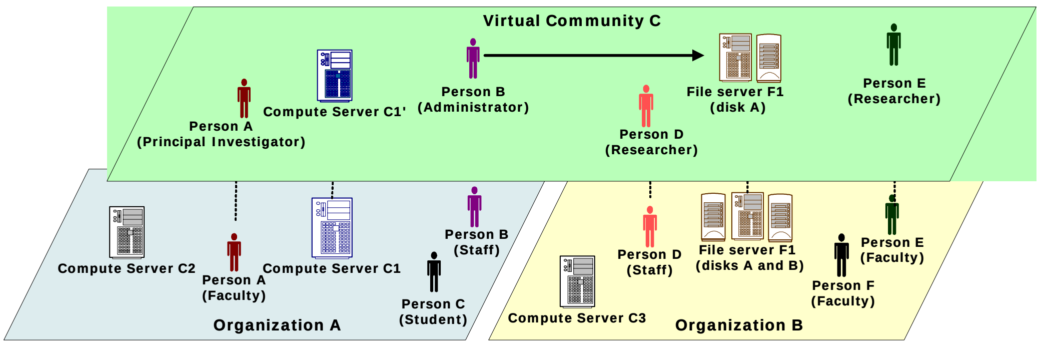

Figure 10: The Virtual Community Concept

Conceptually, the presented model is no different from the general Web services security model and other distributed computing architectures [RFC2903][WS Security Whitepaper][Liberty]. The Grid applications, however, put a distinct focus on the entities and patterns related to cross-organizational settings, and the security services that enable those interactions [VO Security][Role-Based VO][CAS][EU DataGrid][Fine-Grain Auth][Fine-Grain Auth RM][Dynamic Access Control][PRIMA][Grid Auth Framework] [Grid AAA Req] [CAS2][SAZ][GS Security]. It can be noted that such interactions are also prevalent in business applications where business-to-business interactions (or organizations boundaries within an enterprise) bring about similar requirements and settings. For example, Figure 10 shows two organizations and an overlaid virtual community that is governed by its own policy. Note that the entities within this virtual community context have their own specific attributes and properties, which are different from, and have no relation to, those in their home domain. This highlights the fact that our security services model has to support the concurrent enforcement of multiple policies, which each has to be evaluated within their own proper context.

3.7.3 Example Scenarios

3.7.3.1 Digital Library

A digital library program for educational material is operated by a public organization. A number of schools and public libraries in the nation participate in the library program. The program provides teachers and students of the participating schools or libraries with a means of sharing educational materials such as digital books, videos, photos and all other digital materials for education that would originally have been stored by individual schools or libraries. Each of the participating schools and libraries is responsible for its users and resource (educational materials) management, such as registration or removal of users and resources. There is also a case where some schools, for example universities, consist of several sub-organizations. In such a case, each sub-organization in a university could form a VO by itself and the university VO would then become an aggregation of these school VOs. This scenario is depicted in Figure 11.

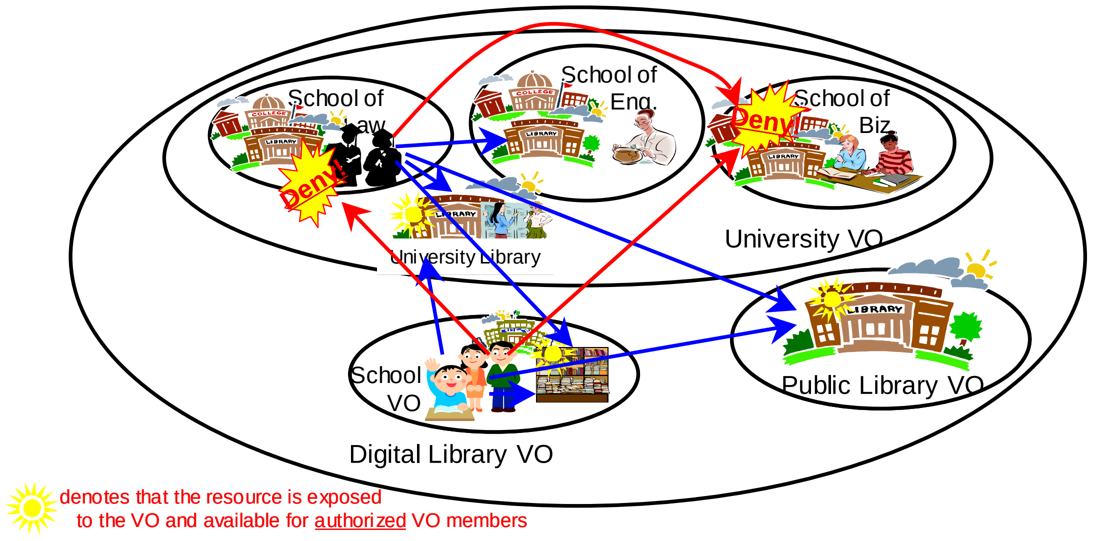

Figure 11: VO example: Digital Library

The following is a list of example policies of how users of the library program have access to the shared materials.

- All the students enrolled in a participant school can have read access to the materials for students.
- All the teachers can have read access to materials both for students and faculties.
- Some teachers who are certified by a participant school can also register new materials.

The school libraries in the university may allow access from inside the university, but will not allow accesses from the Digital Library VO.

3.7.3.2 Least Privilege Delegation

The delegation of rights is a fundamental capability needed to let services work on behalf of other entities. With this rights-delegation comes the associated risk that any of these services may be compromised and use those rights in inappropriate ways. To limit the exposure, one would like to limit the delegated rights to only those rights truly needed by the service. This *least-privilege delegation model* requires that one is able to match the invoked service operations with the exact “amount” of rights, which is a non-trivial requirement. Many Grid applications use the concept of jobs, in which job directives are specified in their own language. The job requirements are then matched with the capabilities and availability of resources by discovery, brokers, and scheduler services. The language used for the expression of these job directives and resource capabilities should be able to match up with the directives used to express the equivalent rights needed. Any mismatch is likely to result in a deployment where essentially too many rights will have to be given to services to ensure that the job directives can be executed. This issue is illustrated in Figure 12.

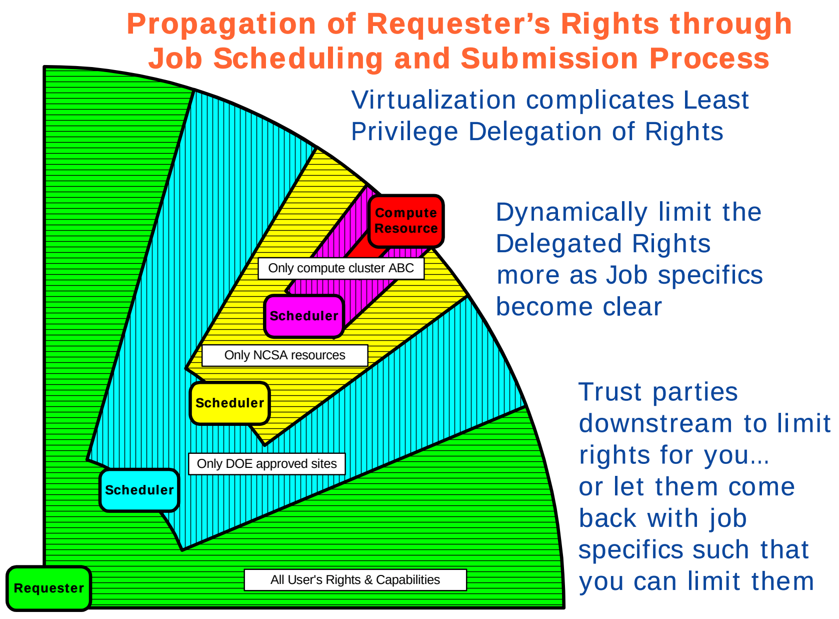
Propagation of Requester's Rights through Job Scheduling and Submission Process
Virtualization complicates Least Privilege Delegation of Rights
Dynamically limit the Delegated Rights more as Job specifics become clear
Trust parties downstream to limit rights for you… or let them come back with job specifics such that you can limit them
Compute Resource
Only compute cluster ABC
Scheduler
Only NCSA resources
Scheduler
Only DOE approved sites
Scheduler
All User's Rights & Capabilities
Requester

Figure 12: Least-Privilege Delegation and Job Description

3.7.3.3 *Secure logging in a distributed environment*

Logging services, and secure access to the logs for reconciliation purposes, becomes a much harder problem in a distributed setting, where the services and associated logs may reside in different administrative domains. This scenario is depicted in Figure 13.

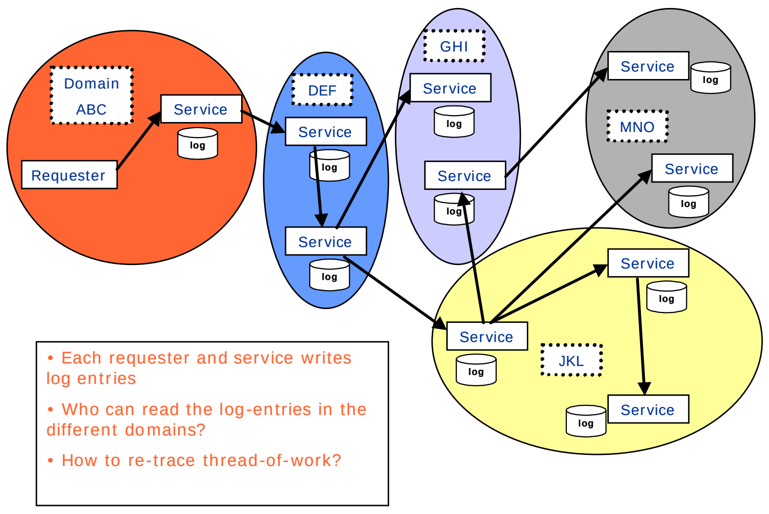
Domain ABC
Requester
Service
log
DEF
Service
log
Service
log
GHI
Service
log
Service
log
MNO
Service
log
Service
log
JKL
Service
log
Service
log
Service
log
• Each requester and service writes log entries
• Who can read the log-entries in the different domains?
• How to re-trace thread-of-work?

Figure 13: Secure logging in a distributed environment

Logging services should have security characteristics so that logs are secured, are capable of being tamper-proof, and ensure integrity of the messages. When we get to that level of sophistication, it leads to the notion of auditing – where events are recorded in a secure fashion. These can be any events, including security events, business events, transaction events, etc. Security services and infrastructure will need to deal with generating security events that can be consumed by the event infrastructure so that they can be audited, or acted upon (e.g. an intrusion defense system may react to a set of DoS events).

3.7.4 Functional Capabilities

In the OGSA model, the functional capabilities are translated into service descriptions and usage patterns.

As an example, Figure 14 illustrates how the requestor and service provider both call-out to different infrastructure security services to ensure policy compliance. Note that in the picture the call-outs are made from within the stubs, and outside of and transparently to the application. The expectation is that most of the policy enforcement could be taken care of this way, which has the desirable benefit of keeping the security-specific code to a minimum for the application developers.

Furthermore, Figure 14 clearly shows that in order for service invocations to comply with the requestor's, the service provider's and the VO's policy, call-outs are made to different security service instances that are managed in those different organizations.

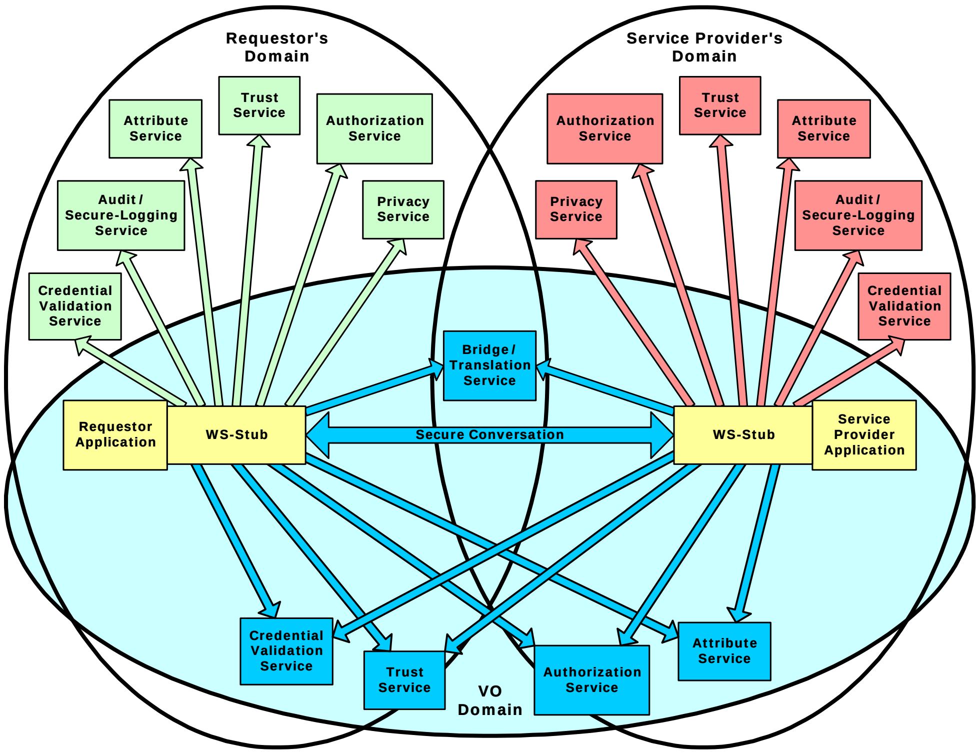

Figure 14: Security services in a virtual organization setting

The functional capabilities and corresponding security services are enumerated below:

- Authentication: Authentication is concerned with verifying proof of an asserted identity. This functionality is part of the Credential Validation and Trust Services in Figure 14. One example is the evaluation of a user-id and password combination, in which a service requestor supplies the appropriate password for an asserted user-id. Another example involves a service requestor authenticating through a Kerberos mechanism, and a ticket being passed to the service provider's hosting environment, which determines the authenticity of the ticket before the service is instantiated.
- Identity mapping: The Trust, Attribute and Bridge/Translation Services in Figure 14 provide the capability of transforming an identity that exists in one identity domain into an identity within another identity domain. As an example, consider an identity in the form of an X.500 Distinguished Name (DN), which is carried within a X.509 V3 digital certificate. The combination of the subject DN, issuer DN and certificate serial number may be considered to carry the subject's or service requestor's identity. The scope of the identity domain in this example is considered to be the set of certificates that are issued by the certificate authority. Assuming that the certificate is used to convey the service requestor's identity, the identity mapping service via policy may map the service requestor's identity to an identity that has meaning (for instance) to the hosting environment's local platform registry. The identity mapping service is not concerned with the authentication of the service requestor; rather it is strictly a policy-driven name-mapping service
- Authorization: The authorization service is concerned with resolving a policy-based access-control decision. The authorization service consumes as input a credential that embodies the identity of an authenticated service requestor and, for the resource that the service requestor requests, resolves, based on policy, whether or not the service requestor is authorized to access the resource. It is expected that the hosting environment for OGSA-compliant services will provide access control functions, and it is appropriate to further expose an abstract authorization service depending on the granularity of the access-control policy that is being enforced.
- Credential Conversion: The Trust, Attribute and Bridge/Translation Services in Figure 14 provide credential conversion from one type of credential to another type or form of credential. This may include such tasks as reconciling group membership, privileges, attributes and assertions associated with entities (service requestors and service providers). For example, the credential conversion service may convert a Kerberos credential to a form that is required by the authorization service. The policy-driven credential conversion service facilitates the interoperability of differing credential types, which may be consumed by services. It is expected that the credential conversion service would use the identity mapping service.
- Audit/Secure-Logging: The audit service, similarly to the identity mapping and authorization services, is policy-driven. The audit service is responsible for producing records that track security-relevant events. The resulting audit records may be reduced and examined so as to determine whether the desired security policy is being enforced. Auditing and subsequently reduction tooling are used by the security administrators within a VO to determine the VO's adherence to the stated access-control and authentication policies.
- Privacy: The privacy service is primarily concerned with the policy-driven classification of personally identifiable information (PII). Service providers and service requestors may

store personally identifiable information using the privacy service. Such a service can be used to articulate and enforce a VO's privacy policy.

A slightly different view of the interdependencies between the relevant security components is shown in Figure 15 as a layered stack of related services.

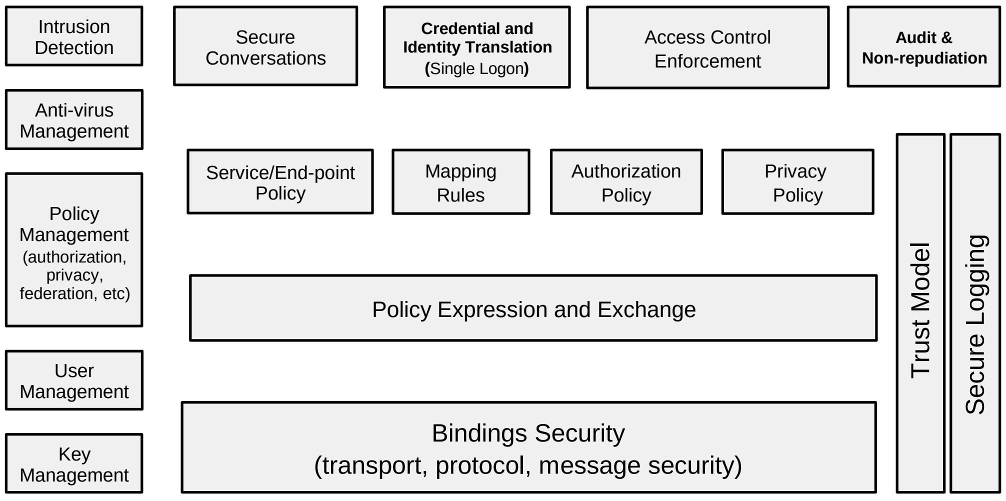
Intrusion Detection
Secure Conversations
Credential and Identity Translation (Single Logon)
Access Control Enforcement
Audit & Non-repudiation
Anti-virus Management
Service/End-point Policy
Mapping Rules
Authorization Policy
Privacy Policy
Trust Model
Secure Logging
Policy Management (authorization, privacy, federation, etc)
Policy Expression and Exchange
User Management
Key Management
Bindings Security (transport, protocol, message security)

Figure 15: Components of Grid Security Model

All security interfaces used by a service requestor and service provider need to be standardized within OGSA. Compliant implementations will be able to make use of existing services and defined policies through configuration. Compliant implementations of a particular security-related interface would be able to provide the associated and possibly alternative security services.

3.7.5 Properties

In general, the properties of the security services depend on the technological requirements that follow from the policy that has to be enforced. For example, in order to be able to enforce a stated policy, certain service levels have to be met by the security services, which translates in properties like:

- Maximum latency
- Response-time
- Availability
- Recovery

In many cases, the implementation of the security services will be able to obtain the desired properties through the use of other services. For example, the attribute information service could use the data services to access the assertion information in LDAP or RDBMS, and it could make use of the data mirroring features of those services to achieve the desired availability property needed to meet the enforcement of the stated security policy.

3.7.6 Interactions with other OGSA services

In general, the invocation of any OGSA service is subject to enforcement of all relevant security policies. In some cases, this enforcement is implicit and hard-coded; in other cases the ability to plug-in or call-out to external security infrastructure services is essential for deployment. In this view, all OGSA services depend on and are layered above the security services.

In some cases, the security services and requirements are intimately connected to other OGSA services on a higher level. For example, an attribute service implementation can make use of OGSA data services to retrieve policy related information from a registry or database. In this view, security services can be consumers of other OGSA services.

3.8 Self-Management Services

3.8.1 Objectives

Self-management was conceived as a way to help reduce the cost and complexity of owning and operating an IT infrastructure. In such an environment, system components—from hardware such as desktop computers and mainframes to software such as operating systems and business applications—are self-configuring, self-healing and self-optimizing.

These self-managing attributes, described further in §3.8.4, suggest that the tasks involved in configuring, healing and optimizing the IT system can be initiated based on situations that the components themselves detect, driven by business needs, and that these tasks are performed by those same technologies. Collectively, these intuitive and collaborative characteristics enable enterprises to operate efficiently with fewer human resources, while decreasing costs and enhancing the organizations' ability to react to change. For instance, in a self-managing system, a new resource is simply deployed and then optimization occurs. This is a significant shift from traditional implementations, in which a significant amount of analysis is required before deployment, to ensure that the resource will run effectively.

Although it is expected that the self-managing attributes will be pervasive in OGSA, it is not the case that every single service will demonstrate all, or even a subset, of these attributes. Rather, these attributes are part of the autonomous nature of the system as a whole.

One of the main objectives of self-management is to support service-level attainment for a set of services (or resources, depending on the taxonomy) – with as much automation as possible, to reduce the costs and complexity of managing the system. In an operational environment, it is often necessary to control various aspects of the behavior of a solution component in a manner that cannot be determined a priori by the component developer. This is achieved in a self-managing system through the deployment of policies to govern the behavior of system components derived from business objectives; a role called *Service Level Manager*. Service level managers (SLMs) are responsible for setting and adjusting policies, and then changing the behavior of the managed resource or service in response to observed conditions in the system to ensure overall compliance with business objectives. SLMs are themselves managed by policies that are either embedded in their implementation or retrieved from other SLMs.

Thus composition and hierarchy are expected between different SLMs, thereby significantly reducing the complexity of operation of the system and initial system design, since complex SLMs can be built by involving simple SLMs in the process.

While the self-management set of services/mechanisms are a significant part of the OGSA, this work is still at a preliminary stage and hence only some aspects of self-management are described here. More detailed analysis and architecture will be addressed in later versions of this document.

3.8.2 Basic Attributes

The collection of attributes collectively needed for various stages of self-management are given below. Their existence at a conceptual level is discussed; no assumptions are made about their implementation.

3.8.2.1 Service Level Agreement

Service level agreements (SLAs) include business or IT agreements between the service provider and the users of the service. SLAs provide guidance, expressed in terms of measurable intent, as to the purpose and delivery objective of the service or resource that is being managed.

3.8.2.2 Policy

A policy is used to govern the behavior of an SLM and the manageable resources under its control. Policies governing the behavior of self-managing entities are derived from service level agreements (SLAs), and deployed as part of the management context under which the manageable resources are used. SLMs orchestrate real-time changes in the dynamic management infrastructure, based upon policy which governs their behavior.

3.8.2.3 Service Level Manager Model

This model provides the framework to instantiate and work with an SLM such that various SLMs and human operators can interact and understand each other without having knowledge about each other built in at design time. There is an interface component to the SLM model, as well as a representation of an SLM and its component management services (not the managed services) and the control loops that they form, and instantiation and maintenance of SLMs.

SLMs are typically modeled after a *generic control loop pattern* that may be used to control and adjust various service activities. This pattern is a cycle consisting of Monitoring, Analysis and Projection, and Action phases. SLMs carry out *service level attainment* activities. Service Level Management is further described in §3.8.4.1.

3.8.3 Example Scenarios

Self-management capabilities are fundamental to the Grid. This section describes two examples of real-world usage: Job-level management and Grid system-level management. Similar concepts can be derived by analyzing how self-management works for other scenarios, e.g., security and so on. In the following scenarios the terminology used is that of the OGSA Execution Management Services capability and the Commercial Data Center use case.

3.8.3.1 Job Level management

The IT business activity manager submits a job and negotiates the job’s SLA. The job must be executed so that it satisfies the agreement. At a later stage, the IT business activity manager may wish to renegotiate the agreement to address new business requirements. It does so with the job manager and updates the existing agreement. When the agreement is updated the resource requirements are recalculated in the service level attainment loop (analysis and projection, see Figure 16), and the provisioning steps (including resource allocation and deployment) are triggered (Action) as a result of the changing conditions. After the provisioning steps, the resources are in a ready state for the required components of the job to start, including starting executable resources such as application server or DBMS.

The IT business activity manager can monitor the load and resource utilization of the provisioned resources using the Monitoring Service or the Metering Service. The IT business activity manager also can obtain the state of the running job from the job manager.

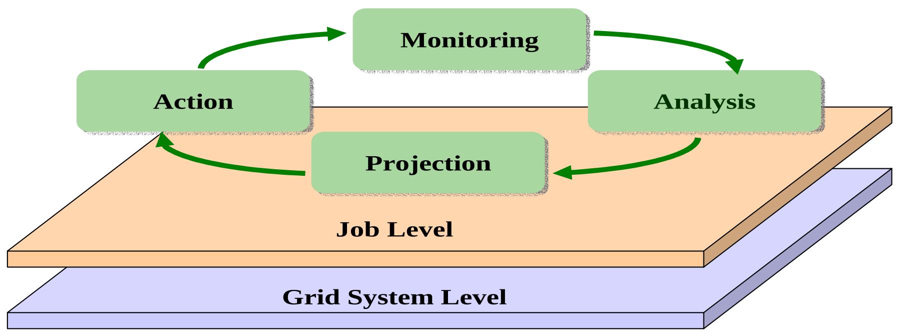

Figure 16: Example scenarios: Grid System Level and Job Level

3.8.3.2 Grid System Level management

At the Grid system level, one aim is to improve resource utilization while maintaining the SLAs of running jobs. The Grid system service level manager may have to add new resources to prepare for expected load increases, and to release surplus resources in order to reduce costs. The resources allocated to a job may also have to be adjusted based on policy – e.g., the priority of a job relative to other jobs in the Grid system.

In the Analysis and Projection phase, information about the available resources and current load, and estimates of the expected future utilization and load are evaluated. An expected increase in utilization may trigger (Action) provisioning steps to add resources to the Grid system's available pool, e.g., by arranging to shift resources from other systems or by releasing resources that are currently being used by lower-priority jobs.

3.8.4 Functional Capabilities

Self-management is essentially self-configuration, self-healing, and self-optimization. This shows the essential difference of self-management from other OGSA categories: namely, it is not just about the components that are involved in doing self-management, but the method by which it is done – the *sauce*, or the way in which the components interact, control loops are formed and systems behave intelligently, based on environmental changes. The mechanisms that bring self-management about can be described as follows:

- Self-configuring mechanisms adapt dynamically to changes in the IT system, using policies provided by the IT professional. Such changes could trigger provisioning requests leading to, for example, the deployment of new components or the removal of existing ones, maybe due to a significant increase or decrease in the workload.
- Self-healing mechanisms can detect improper operations of and by the resources and services, and initiate policy-based corrective action without disrupting the IT environment. Self-healing has an element of self-protection included in it as well. This means that components can detect hostile behaviors as they occur, and take corrective actions to make themselves

less vulnerable. The hostile behaviors could include unauthorized access and usage, virus infection and proliferation, and denial-of-service attacks.

- Self-optimizing mechanisms are able to tune themselves to the best efficiency to meet end-user or business needs. The tuning actions could mean reallocating resources to improve overall utilization or optimization by enforcing an SLA. Self-optimization makes use of self-configuration in its implementation.

It is important to note that the self-configuring, self-healing, self-optimizing attributes are not independent of one another. They work together to allow changes to be made to the configuration of one or more aspects of the IT system.

All of the OGSA service categories are utilized in achieving self-management. In addition, the following sections highlight some special functional requirements that are important for self-management, but that may not be reflected in the same way in other OGSA service categories.

3.8.4.1 Service level management

Service level management ensures that the desired QoS is maintained. This is done through the activities of the SLMs as described in §3.8.1. These *service level attainment* activities are described in more detail here

- **Monitoring**: The SLM receives and processes resource instrumentation through a monitoring component. Service execution and monitoring of resource utilization—for example, monitoring the load and utilization of resources and the running states of service components, and detecting faults, may be made possible by using the monitoring services from the general Resource Management capability in OGSA. Such monitoring information is used as input to the Analysis phase.
- **Analysis and Projection:** Analysis is performed against the instrumentation to evaluate and determine compliance with the established policy and SLAs. The manager gets to know if resources are meeting QoS objectives and operating within defined policies. Analysis can also predict future resource behavior based on history and projected requirements. When system behavior is not consistent with overall goals, the manager evaluates alternative courses of action to effect changes in the set of configured resources in its sphere of influence, and selects a plan of action in accordance with its configured policies.
- **Action:** The manager then executes the plan, either by interacting with underlying managed resources or by communicating with other managers responsible for other aspects of the system. For example a workload manager may adjust priorities and process shares of tasks executing within a cluster of processors to meet service level objectives and policies. Or, in cases where local action against a pool of resource is either impossible (in violation of policy or constrained) or ineffective, a resource manager may “appeal” to other management functions to remedy the situation. For example, a workload manager might make a provisioning request to add additional processors to a cluster.

This control loop executes continuously, assessing the current state of the system against the expressed service level objectives and making adjustments as necessary to bring the system into compliance. Other services are needed for these activities to be successful—for example *capacity planning, entitlement of resources, problem and root cause analysis, predictive analysis,* etc.

A number of management components may be involved in service level attainment activities. The QoS and SLA requirements addressed by these management activities can be extremely varied. For example, they may include performance attainment objectives such as processor capacity or utilization (workload management), or more qualitative objectives such as security levels. Service level attainment activities affect each other directly or indirectly. Therefore the relationship and order of execution between the different management components in the control loops and service level managers should be fairly loose.

3.8.4.2 Policy and Model based Management

Service level managers orchestrate real-time changes in the dynamic management infrastructure based upon policy that governs their behavior. The intelligence in the self management is basically directed by the policies and models about the SLMs and the services that are being managed.

3.8.4.3 Entitlement

Entitlement is about negotiating with other resource holders (other SLMs, human operators, resource pools etc.) to obtain the right resources. During this process different SLAs are compared to find out which resource need is more important. Entitlement is an earlier phase to resource reservation and provides a looser binding – an option to reserve. In contrast, a resource reservation guarantees access to the resource.

3.8.4.4 Planning

Planning refers to calculating the optimum requirements of a service in terms of the resources it needs. This can be at an initiation stage, due to some problem/root cause being identified, or due to a command from a higher level.

3.8.4.5 Capacity Management

Capacity management includes the actual actions relating to updating the current state and requirement of resources. It includes things such as linking with inventory, asset management, moving the resources from the current location into the service domain, moving the resources from the service domain into a storage area or to another domain, etc.

3.8.4.6 Provisioning

Provisioning, including its sub-activities of deployment and configuration, is an important activity that is done in support of self-managing actions as resources are prepared for their expected use. Provisioning is supported by a number of other OGSA services, such as the Application Contents Service that maintains the deployment contents and configuration descriptions.

3.8.4.7 Analytics

- **Problem and root cause analysis** – Analyze monitored data to find out the express issue and the root cause of detected problems. Includes filtering capabilities, correlation etc.
- **Predictive analysis** – Analyze monitored data to predict future behavior – for example to plan for future resource needs, or to predict future problems.

3.8.5 Properties

Service Level Management components implement self-managing, policy-based capabilities across multiple QoS dimensions of the management infrastructure. Key QoS dimensions include, but are not limited to, availability, security, and performance. These cornerstone QoS dimensions are expressed in terms of measurable intent captured in SLAs. Availability might be measured in terms of “minutes of outage.” Desired security capabilities might be described as privileges provided by a class of service for associated users. Performance characteristics could be specified in terms of well-known throughput or response-time objectives.

3.8.5.1 Performance

Gives measurements and metrics – for example, quantifiable data about how the system is performing, such as cumulative CPU load factor, that go into computing the performance metrics of a specified service.

3.8.5.2 Availability

Specifically denotes the metrics that show the failure rate of the service – of all types, including the macro cases such as system crash, and other, less observable cases, such as the failure of a service to deliver due to throughput drop and buffer flushing.

3.8.5.3 Security

Security is an extremely important aspect of the IT environment. These measures may include security violations, probability of security violations, and identity management metrics such as the time to set up or delete a user, etc.

3.8.6 Interactions with the rest of OGSA

Self-management interacts with almost all other aspects of OGSA. As the required interactions are still being worked out, only some services are listed here as an example of what interactions are expected, with no attempt at being exhaustive.

- Discovery – to find and integrate new resources and services.
- Logging and Monitoring – to provide the information needed to determine the state of the system.
- Resource Reservation – to facilitate more predictable resource usage.
- Workflow – to automate the actions that the SLMs have to carry out when addressing abnormal conditions.
- Composition – to construct complex or higher level SLMs from simple ones.
- Security, and in particular Authentication and Authorization, are essential as different system components may be involved in management actions.
- Resource management, and in particular service or resource manageability models, are required to provide the representation of the service or resource that is being managed. The SLMs can read this information, and can act on the intelligence in it, in response to a changing environment or a command from a higher level. Further work to determine how best to model a service or resource needs to be done.

3.9 Information Services

3.9.1 Objectives

The ability to efficiently access and manipulate information about applications, resources and services in the Grid environment is an important OGSA capability. In this section, the term *information* refers to: dynamic data or events used for status monitoring; relatively static data used for discovery; and any data that is logged. In practice, an information service needs to support a variety of QoS requirements for reliability, security, and performance. The scope of the OGSA information service covers publication through consumption. Activities prior to publication (e.g. sniffing network packets) or subsequent to consumption are out of scope.

While it may be possible to design one single information service that deals with all information delivery patterns and QoS, OGSA is best served by multiple information services, some general, and some optimized to meet specific use cases. However, caution should be exercised so as not to end up with a number of fractured information services, each capable of answering a limited number of use cases. The challenge is to balance the desire for generality and requirements on domain-specific semantics and QoS. In theory, generality can be achieved by abstracting common (high-level) functionality and features covering the requirements of as many use cases as possible. While common semantics might encourage interoperable implementations they can also be too high-level and not fully expose desired behavior. The question of how far the level of abstraction can be raised, without compromising the usability of the services, is a topic that needs further investigation.

Clients of the OGSA information services include, but are not limited to, execution management services, accounting services, problem determination services, resource reservation services, resource usage services, and application monitoring. To facilitate interoperability and reuse, the information services themselves should be built on top of OGSA infrastructure capabilities such as notification (e.g., WS-Notification). Information services could also make use of other OGSA capabilities such as data access and distributed query processing.

3.9.2 Models

The characterization of an information service depends greatly on factors such as the demand placed on the source of information (e.g., static versus dynamic, publication rate), its purpose (e.g., discovery, logging, monitoring) and QoS requirements. However, we see similar, recurring structures in information services. Information is made available for consumption, either from the originating producer, or through an intermediary (e.g. logging service, notification broker) acting on behalf of the originating producer. Either one or more consumers wish to obtain information from one or more producers, or one or more producers wish to send information to one or more consumers. Producers and consumers should be decoupled and not be required to have any prior knowledge of each other. Consumers may contact a producer (or intermediary) and pull information in one call or they may use a subscription mechanism to receive information as it becomes available.

OGSA is not prescriptive on the data model used to implement an information service or the language used to query for information. Current systems broadly fall into those that are based on XML and Xpath/Xquery query languages (e.g., Globus MDS) and those that use the relational model and the SQL query language (e.g., R-GMA).

Metadata is associated with information (e.g., events or messages) for describing its structure, properties and usage. For interoperability, a standard event scheme for OGSA information services is desirable. In some cases, such as when performance is paramount and interoperability is not a concern; user-defined, optimized events may be more appropriate.

An information service might allow producers and consumers to discover each other by making detailed descriptions about themselves available for querying. A special distributed registry or point-to-point scenario could be used for that purpose. The descriptions could include, for example, the type of producer or consumer, what information they produce or consume, and their endpoint URLs.

3.9.3 Example scenarios

3.9.3.1 Directory scenario

A user needs to locate a service description that meets some desired functional and other criteria. He queries a central directory service where service descriptions are published, and receives an answer. The directory owner decides who can publish information into the directory and who is authorized to query it. UDDI is an example legacy directory service. OGSA directory services could be based on WSRF service group concepts.

3.9.3.2 Logging scenario

A number of distributed computing usage scenarios require logging services. These scenarios include those based on problem determination, usage metering, failure recovery, transaction processing, and security.

A problem determination scenario might proceed as follows. A developer is writing code for a Disaster Recovery application. Following corporate programming guidelines for logging, he inserts log statements into his code to provide information regarding any unexpected situations. He is responsible for coding a section of the application that accesses an RDBMS to obtain attributes for candidate fail-over resources. He codes this section of the application such that RDBMS failures are trapped and log records are generated to reference the RDBMS failure. After development has been completed, the application is deployed into an operating environment where log records are processed by a handler chain. In this environment, based on the severity level (e.g., fatal, warning, informational), records exceeding an operator-specified level are stored in a log by a file handler. Due to the deployment of an erroneous new security policy, the code reaches an abnormal state and logs a record indicating that the underlying RDBMS has denied access to the application. The operator had configured the operating environment to store log records of this severity. After the failure, an analyst, in the role of a log consumer, uses a console application to peruse the logs. He finds the log statement indicating the RDBMS problem and directs the database administrator to correct the security policy.

3.9.3.2.1 Grid Monitoring Architecture (GMA) scenario

A scientist wants to run an interactive simulation/rendering job that must be finished within the next half hour. He must find 100 computing elements that are fast enough, have enough memory between them, and are linked by fast network connections. He needs up-to-date information and submits a complex query to the OGSA information service. After a short time he receives the information on the computing resources that satisfy his query. He selects the nearest ones and proceeds with his job.

In answering the query, the information service first discovers the information producers for the computing, storage and network resources that are relevant and then gets the necessary information before performing a “join” to identify the resources that satisfy the query. The scientist then needs to monitor his job and respond to partial failures in the execution environment, possibly using the same GMA-based service (provided that the information has an associated timestamp). GMA is an example of a multi-purpose information services architecture [GMA].

3.9.3.3 Producer/Consumer patterns

The basic pattern of decoupling producers from consumers using an intermediary is widely used in computing. The Producer-Intermediary-Consumer pattern is shown in Figure 17. Producers put data *into* an intermediary, and consumers extract data *from* it. In a general sense, this is the pattern followed by any data store. That is, producers write to and consumers read from a file, RDBMS, ODBMS etc.

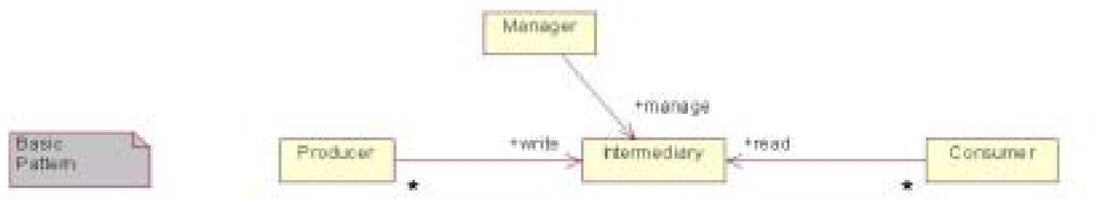

Figure 17: The Producer - Intermediary - Consumer Pattern.

Figure 17 also shows that, in addition to supporting producer and consumer interfaces, an intermediary may also support a management interface. The management interface controls those functions that are not directly associated with reading or writing to the data store. Operations controlling retention policy, backup policy, etc. would be accessed through a management interface.

In distributed computing, the above pattern is followed by several infrastructure services whose task it is to provide information to consumers. We broadly refer to these services as information services. They include: name services, logging services, and notification services. We assume that the reader is generally familiar with these concepts and we defer OGSA-specific discussions on each service.

Each of these services has evolved to support a distinct set of domain-specific semantics and qualities of service. While they each could be implemented using a general-purpose RDBMS for an underlying data store, this would most likely result in trading off some desirable, specialized qualities of service for unneeded functionality. For example, a log service that must support fast writes would most likely find the performance and footprint costs of a full-function RDBMS to be prohibitive. Similar observations could be made about name services and notification services.

Figure 18 attempts to characterize each of these information services. Each service is labeled on the left. Service semantics and data store requirements are depicted using UML, and examples of existing technologies, along with a brief summary, are shown to the right of each service.

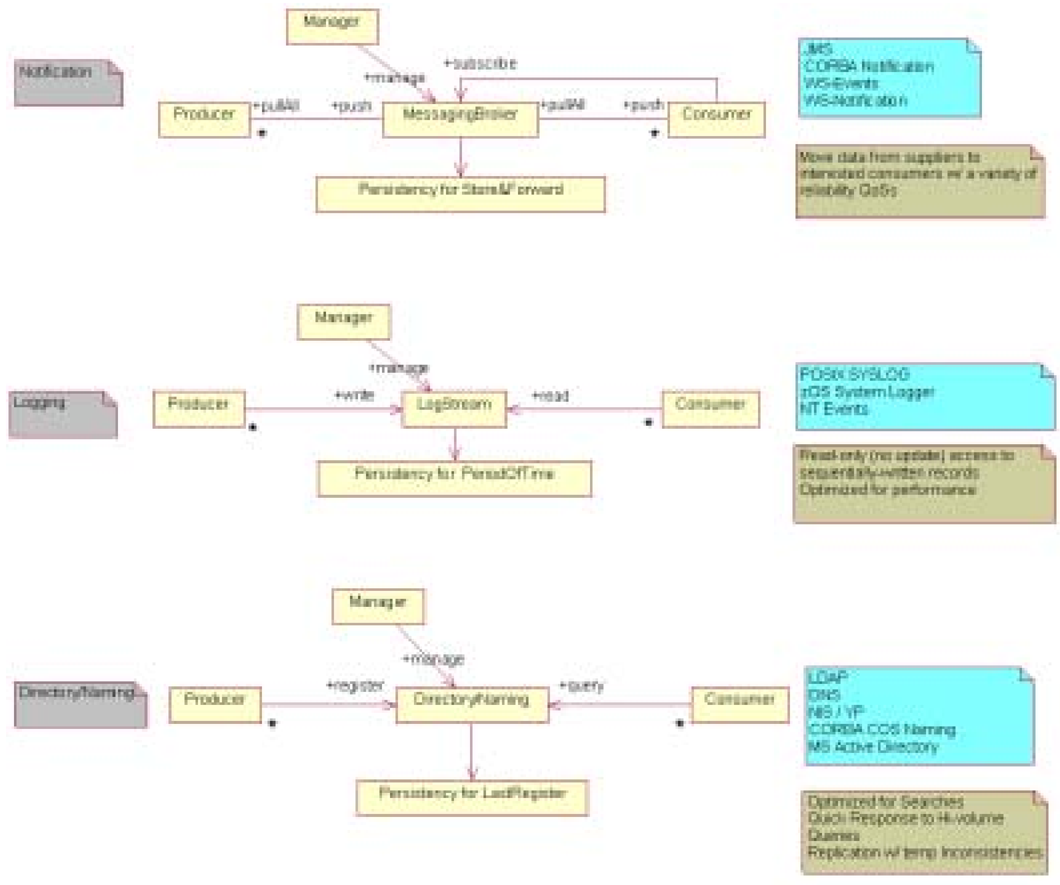

Figure 18: Basic OGSA Information Services.

A more general information service could support a combination of these basic capabilities. Adhering to a single capability can result in a very specialized service, applicable to only a subset of current use cases. This approach might be desirable when other requirements (for example performance or specific semantics) are paramount. Analysis of the OGSA use cases reveals that a service that supports a combination of capabilities is often desirable. For example, GMA requires both directory and notification capabilities.

3.9.4 Functional capabilities

We define naming, discovery, message delivery, logging, and monitoring capabilities.

3.9.4.1 Naming scheme

Traditional distributed systems usually support a two- or three-layer naming scheme. In OGSA, the naming service, ***OGSA-naming***, uses a three-level convention. Every named OGSA entity is associated with: a (optional) ***human-oriented name***, an ***abstract name***, and an ***address***.

The ***human-oriented name*** is usually human-readable and may belong to a name space. Name spaces are usually hierarchic and usually have syntactic restrictions. Hierarchic name spaces permit each part of a name to map to a particular context. For example, in the Unix file "`node1:/var/log/error`," the context for `var` is the root directory of a machine named "`node1`," the context for "`log`" directory is "`var`," and the context for "`error`" file is "`log`." OGSA-naming does not require human-oriented names to be unique. Many different naming schemes exist and this document does not attempt to prescribe the set of all supported schemes.

The ***abstract name*** is a persistent name that does not specify a particular location. The ***abstract name*** may not be globally unique. A WSRF endpoint reference [WS-RF] with renewable references and the Legion Object Identifier are examples of abstract names. A mechanism, outside the scope of this document, is required to map ***human-oriented names*** to ***abstract names***.

The ***address*** is a concrete name that contains the location of an entity. The combination of the endpoint address and reference properties in a WSRF endpoint reference [WS-RF], a memory address, and an IP address/port pair are examples of ***addresses***. A mechanism, outside the scope of this document, is required to bind ***abstract names*** to ***addresses***.

In OGSA we assume the existence of a ***resource handle***. A resource handle is an abstract name of a resource and its associated state (if any).

3.9.4.2 Discovery

One universally needed capability is service and resource discovery. A directory (or registry) is an obvious solution, but not the only one. A directory is distinguished from other possible solutions in that it has persistent storage for the "latest" information and is optimized for searches. Low latency response to a high volume of queries is required. The directory may be replicated for scalability.

Alternatively, a compilation or guide of information can be stored in an *index* (such as Google). Unlike a registry, which tends to be centrally controlled, anyone can create an index.

Another alternative is peer-to-peer discovery, where a Web service is a node in a network of peers and dynamically queries its neighbors in search of a suitable match. The query propagates through the network from one node to another until a match is found, a particular hop count is reached, or some other termination criterion is satisfied. Yet another alternative for a discovery service is a general GMA-based service (see below).

3.9.4.3 Message delivery

Producers and consumers interact by exchanging messages, and this can be handled by a common messaging infrastructure. This infrastructure is only concerned with how to distribute copies of messages to interested parties, not how these messages are constructed in the first place. Producers either send messages directly to relevant consumers or make use of an intermediary (message broker) that decouples producers from consumers. In the latter case the producers publish their messages to the broker, which takes the responsibility for forwarding the message to interested parties. A producer (or the intermediary) may provide notification capabilities and additional function such as a finder service (allowing its producers and consumers to find each other). Message brokers may store and forward messages in stable storage – not necessarily for persistency, but for reliable message delivery purposes.

3.9.4.4 Logging

An OGSA logging service acts as an intermediary between log artifact producers and consumers. Producers write log artifacts sequentially, and consumers may read (but not update) the log records. To ensure the general acceptance of the basic log semantics and to enable the exploitation of existing implementations, OGSA logging services should support key features found in existing logging implementations. There may be multiple producers and consumers for a given logging intermediary, and both may set filters for records. In the logger service the message exchange is optimized for performance and the records are kept in a persistent store for a period of time.

3.9.4.5 Monitoring

Information that carries a field for ordering purposes (e.g., a time stamp and sequence number) can be used for monitoring. An OGSA monitoring service could be equally used for applications or resources. Some situations (e.g., real-time applications) might impose strict requirement on the monitoring service (e.g., high update rates and high performance). In such a case a special-purpose service might be needed.

3.9.4.6 General Information and Monitoring service

A general OGSA service that provides a combination of the above capabilities can provide more flexibility to the end user. For Grid resources in general (including services and applications), the amount of available information about resources could be large, dispersed across the network, and updated frequently. Searches in this space may have unacceptable latencies. In order to manage such information in a controllable way, it is important to separate information source discovery from information delivery. Searches should only be used to locate information sources or sinks. A special-purpose directory is used to hold metadata about the resources. In this case the directory must cope with high rates of updates that are expected in the dynamic OGSA environment. Individual producer/consumer pairs can limit the amount of data flowing between them to that satisfying the consumer query. This model differs from a message broker that combines the mechanisms for finding sources and sinks of information and its delivery into a single searchable channel. The merits of this approach are described in the GMA document [GMA].

A user of such a general service should be able to put any information, irrespective of its intended use (e.g., discovery, monitoring), into it without needing to understand the complexity of the system. He first must specify what information is to be made available in OGSA, where the type and structure of that information should be well-defined. The user might also wish to specify certain properties and policies. This could include how the information is held, retention period, guarantee of delivery, persistency, and access control. A consumer could filter information of interest using a subscription topic, for example, and it could support a query to further refine the events delivered to it through predicates defined in the query expression.

More advanced users may require a deeper understanding of the internal workings of the service. Following a request for information in a general (GMA-based) service, expected behavior is that a “mediator” capability is used to perform a registry/schema lookup and locate suitable sources of information. For long-term queries the mediator ensures that, as sources are dropped or new relevant sources come online, the subscribed consumers are updated. Mediation is also concerned with planning the distributed queries to the relevant producers, as well as merging the results.

3.9.5 Properties

3.9.5.1 Security

Authentication and authorization rules for consumers and producers allow them to exchange information in a secure fashion. Discovering metadata information about producers and consumers (e.g., their existence or the type of information they produce or consume) could also be subjected to security rules. Some services (e.g., metering, authentication, authorization) require that messages be made secure (e.g., encrypted) for delivery.

3.9.5.2 Quality of service

Various levels of QoS can be provided by OGSA information services, such as reliable, guaranteed delivery. WS-Reliability, defined by the OASIS WS-RM TC, provides these reliable message delivery properties [WS-Reliability].

3.9.5.3 Availability/Performance/Scalability

Information systems play a critical role in OGSA. Since almost every other capability in OGSA makes use of them, they need to be available at all times and to be especially tolerant of partial failure. Many clients of the information systems expect to receive information at high rates and cannot afford to wait long periods of time. High-performance systems are needed in this case. Because a large number of resources, services and applications may wish to produce and consume information, the system must also be scalable across wide-area networks.

3.9.6 Interactions with the rest of OGSA

Standard event data models facilitate the transfer of information from producers to consumers. The use of recognized standards event schema permits the discovery and processing of events from a large variety of potentially different sources across a VO. To accommodate domain-specific events, the schema should be extensible. The WSDM event schema from the WSDM-TC in OASIS may be used as the basis for the OGSA event data model.

Notification mechanisms, or an extension of them, could be used to deliver events from producers to consumers once they have discovered each other, and between other components of the system.

Security services are needed for authentication/authorization between different components.

Distributed query processing is required in the query-planning phase of the mediation between producers and consumers of information in GMA.

Replication mechanisms are needed so that repositories of metadata (directories or registries) can be distributed and replicated to avoid single points of failure and improve scalability. Some temporary inconsistence between replicated copies might be acceptable in some situations; in this case the information system needs to be robust in the event of inconsistent or out-of-date metadata.

4 Security Considerations

Security considerations are discussed in the Security services section (§3.7).

5 Editor Information

Ian Foster
Distributed Systems Laboratory
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
Phone: 630-252-4619
Email: foster@mcs.anl.gov

Hiro Kishimoto
Grid Computing and Bioinformatics Laboratory
Fujitsu Laboratories
4-1-1, Kamikodanaka, Nakahara, Kawasaki City, Japan

Phone: +81-44-754-2628
Email: hiro.kishimoto@jp.fujitsu.com

Andreas Savva
Grid Computing and Bioinformatics Laboratory
Fujitsu Laboratories
4-1-1, Kamikodanaka, Nakahara, Kawasaki City, Japan
Phone: +81-44-754-2628
Email: andreas.savva@jp.fujitsu.com

6 Contributors

We gratefully acknowledge the contributions made to this specification by Jeffrey Frey, Takuya Mori, Jeffrey Nick, Chris Smith, David Snelling, Latha Srinivasan, Jay Unger.

7 Acknowledgements

We are grateful to numerous colleagues for discussions on the topics covered in this document, in particular (in alphabetical order, with apologies to anybody we've missed) Mario Antonioletti, Takuya Araki, Jamie Bernardin, Shel Finkelstein, Steve Fisher, Dennis Gannon, Kate Keahey, Takashi Kojo, Peter Kunszt, Simon Laws, Tan Lu, James Magowan, Susan Malaika, David Martin, Nataraj Nagaratnam, Dave Pearson, Benny Rochwerger, John Rofrano, Ellen Stokes, Paul Taylor, Sachiko Wada, James Warnes, and Ming Xu.

References

[CAS] Foster, I., Kesselman, C., Pearlman, L., Tuecke, S., Welch, V. The Community Authorization Service: Status and future. *Proceedings of Computing in High Energy Physics (CHEP '03)*, 2003.

[CAS2] Pearlman, L., Welch, V., Foster, I., Kesselman, C., Tuecke, S. A Community Authorization Service for Group Collaboration. *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2001.

[Condor] http://www.cs.wisc.edu/condor

[DAGman] http://www.cs.wisc.edu/condor/dagman/

[Dynamic Access Control] Lepro, R., Cardea: Dynamic Access Control in Distributed Systems. *NAS Technical Report NAS-03-020*, 2003.

[EU DataGrid] EU DataGrid, VOMS Architecture v1.1. 2003. http://grid-auth.infn.it/docs/VOMS-v1_1.pdf.

[Fine-Grain Auth] Keahey, K., Welch, V., Lang, S., Liu, B., Meder, S. Fine-Grain Authorization Policies in the GRID: Design and Implementation. *1st International Workshop on Middleware for Grid Computing*, 2003.

[Fine-Grain Auth RM] Keahey, K., Welch, V. Fine-Grain Authorization for Resource Management in the Grid Environment. *Proceedings of Grid2002 Workshop*, 2002.

[Globus MDS] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman. Grid Information Services for Distributed Resource Sharing. *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, IEEE Press, August 2001.

[Grid Auth Framework] Lorch, M., Cowles, B. (eds.) Conceptual Grid Authorization Framework and Classification. GGF Working Group on Authorization Frameworks and Mechanisms, 2004.

[Grid AAA Req] Mullen, S. Crawford, M., Lorch, M, Skow, D. Grid Authentication Authorization and Accounting Requirements. GGF Site Authentication, Authorization, and Accounting working group. January, 2004.

[Grid Anatomy] 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.

[Grid Physiology] Foster, I., Kesselman, C., Nick, J. and Tuecke, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Globus Project, 2002. www.globus.org/research/papers/ogsa.pdf.

[GS Security] Welch, V., et. al. Security for Grid Services, *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, IEEE Press, to appear June 2003.

[GMA] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor and R. Wolski, *A Grid Monitoring Architecture*, Global Grid Forum, Final Document Series, March 2000.

[JobQueue] Katramatos, D., Humphrey, M., Grimshaw, G., and Chapin, S. J. JobQueue: A Computational Grid-Wide Queueing System. Proceedings of the Second International Workshop on Grid Computing. Lecture Notes in CS, 2001.

[JSDL] Anjomshoaa, A., Brisard, F., Ly, A., McGough, S., Pulsipher, D., Savva, A. (ed.) Job Submission Description Language (JSDL) Specification, Global Grid Forum JSDL-WG, Draft, June 2004. https://forge.gridforum.org/projects/jsdl-wg/document/draft-ggf-jsdl-spec/en

[Kerberos V5] Kohl, J., and Neuman, C. The Kerberos Network Authentication Service (V5), RFC 1510, IETF, 1993.

[Legion] S.J. Chapin, D. Katramatos, J.F. Karpovich, and A.S. Grimshaw, "Resource Management in Legion," *Journal of Future Generation Computing Systems*, vol. 15, pp. 583-594, 1999.

[Liberty] Liberty Alliance Project, http://www.projectliberty.org/, 2004.

[OASIS] http://www.oasis-open.org/home/index.php

[OGSA Use Cases] Foster, I., Gannon, D., Kishimoto, H and J. Von Reich, Jeffrin. (eds.) Open Grid Services Architecture Use Cases. Global Grid Forum OGSA-WG, Draft, March 2004. https://forge.gridforum.org/docman2/ViewProperties.php?group_id=42&category_id=0&document_content_id=1889

[OGSA Use Cases Tier 2] J. Von Reich, Jeffrin. (ed.) Open Grid Services Architecture: Second Tier Use Cases. Global Grid Forum OGSA-WG, Draft, March 2004. https://forge.gridforum.org/docman2/ViewProperties.php?group_id=42&category_id=0&document_content_id=1886

[OGSI] Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., Maquire, T., Sandholm, T., Snelling, D. and Vanderbilt, P. Open Grid Service Infrastructure (OGSI), Version 1.0, 2003.

[PKI] Housley, R., Polk, W., Ford, W., and Solo, D., Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. *RFC 3280*, IETF, April 2002.

[RFC2903] de Latt, C., Gross, G., Gommans, L., Vollbrecht, J., Spence, D. Generic AAA Architecture, RFC 2903, IETF, 2000.

[PRIMA] Lorch, M., Adams, D., Kafura, D., Koneni, M., Rathi, A., and Shah, S., The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments, *4th Int. Workshop on Grid Computing - Grid 2003*, 17 November 2003 in Phoenix, AR, US

[Role-Based VO] Canon, S., Chan, S., Olson, D., Tull, C., Welch, V. Using CAS to Manage Role-Based VO Sub-Groups. *Proceedings of Computing in High Energy Physics (CHEP '03)*, 2003.

[OGSA RM] Maciel, F. B. (ed.) Resource Management in OGSA. Global Grid Forum CMM-WG, Draft, June 2004. http://forge.gridforum.org/projects/cmm-wg/document/CMM_Gap_Analysis/en/

[SAZ] Sekhri, V. and Mandrichenko, I. Site Authorization Service (SAZ) at Fermilab. *Proceedings of Computing in High Energy Physics (CHEP '03)*, 2003.

[WS Security Whitepaper] Security in a Web Services World: A Proposed Architecture and Roadmap, Version 1.0, A joint security whitepaper from IBM Corporation and Microsoft Corporation. April 7, 2002, http://www-106.ibm.com/developerworks/library/ws-secmap/

[WS-N] Graham, S. (ed), Niblett, P. (ed), Chappell, D., Lewis, A., Nagaratnam, N., Parikh, J., Patil, S., Samdarshi, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W., Weihl, B. Publish-Subscribe Notification for Web services, Version 1.0, May 3, 2004. http://www.oasis-open.org/committees/download.php/6661/WSNpubsub-1-0.pdf

[WS-RF] Czajkowski, K., Ferguson, F. D., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S. and Vambenepe, W. The WS-Resource Framework, Version 1.0, March, 2004. http://www.oasis-open.org/committees/download.php/6796/ws-wsrf.pdf

[WS-Agreement] Czajkowski, K., Dan, A., Rofrano, J., Tuecke, S. and Xu, M. Agreement-Based Grid Service Management (WS-Agreement). Global Grid Forum, Draft, 2003.

[WS-Architecture] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C. and Orchard, D. Web Services Architecture. W3C, Working Draft, 2003. http://www.w3.org/TR/2003/WD-ws-arch-20030808/

[WS-Reliability] Kazunori Iwasa, WS-Reliability 1.1, Working Draft 1.05, 7 July 2004. http://www.oasis-open.org/committees/download.php/7598/WS-Reliability-2004-07-07.pdf

[XQuery] Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J., Siméon J. XQuery 1.0: An XML Query Language, Working Draft, Nov. 2003. http://www.w3.org/TR/xquery/

[VO Security] Siebenlist, F., Nagaratnam, N., Welch, V., Neuman, B.C. Security for Virtual Organizations: Federating Trust and Policy Domains. In *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*, 2004.