GWD-C (draft-ggf-ogsa-usecase-1)
Open Grid Services Architecture Use Cases
http://www.ggf.org/ogsa-wg

**Editors:**
I. Foster, Argonne & U. Chicago
D. Gannon, Indiana U.
H. Kishimoto, Fujitsu Labs

June 6, 2003

# Open Grid Services Architecture Use Cases

## Status of this Memo

This document provides information to the community regarding the Grid use case scenarios used in the definition of Open Grid Services Architecture (OGSA) Platform components. Distribution of this document is unlimited. This is a DRAFT document and continues to be revised.

## Abstract

Successful realization of the Open Grid Services Architecture (OGSA) vision of a broadly applicable and adopted framework for distributed system integration requires definition of a wide variety of Grid use case scenarios of both e-science and e-business applications. Use cases described in this document cover infrastructure topics (Commercial Data Center, IT Infrastructure and Management, Workflow), as well as scientific or commercial applications (National Fusion Collaboratory, Severe Storm Prediction, Collaborative Grid, Service-Based Distributed Query Processing, Online Media and Entertainment). The list of Grid use cases presented here is necessarily preliminary and incomplete. Also use cases are not described at the detail required for formal requirements.

ogsa-wg@ggf.org

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

## Full Copyright Notice



## Intellectual Property Statement

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

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

## Contents

# 1 Introduction

One component of the OGSA-WG's charter is

> "To produce and document the use cases that drive the definition and prioritization of OGSA Platform components, as well as document the rationale for our choices."

This document is a collection of the use case scenarios contributed by OGSA-WG participants or solicited from others. It is a companion to "The Open Grid Services Architecture Platform."

Based on this document the OGSA-WG will (a) specify, in broad but somewhat detailed terms, the scope of important services required, (b) identify a core set of such services that are viewed as essential for many Grid systems and applications, and (c) specify at a high-level the functionalities required for these core services and the interrelationships among those core services.

While these use cases have certainly not been defined with a view to expressing formal requirements (and do not contain the level of detail that would be required for formal requirements), they have provided useful input to the definition process. We expect to expand the number of use cases in future revisions of this document.

Table 1: Use cases and contributors in this document

| Chapter | Title | Contributors |
|---|---|---|
| 2 | Commercial Data Center | Hiro Kishimoto, Andreas Savva, David Snelling |
| 3 | IT Infrastructure and Management | Ravi Subramaniam |
| 4 | National Fusion Collaboratory | Kate Keahey |
| 5 | Severe Storm Prediction | Dennis Gannon |
| 6 | Collaborative Grid Scenarios | Charles Severance |
| 7 | Online Media and Entertainment | Tan Lu, Boas Betzler |
| 8 | Service-Based Distributed Query Processing | Nedim Alpdemir, Norman Paton |
| 9 | Grid Workflow | Takuya Araki |
| 10 | Grid Resource Reseller | Jon MacLaren, William Lee |

# 2 Commercial Data Center

## 2.1 Summary

In these days, many enterprises consolidate a huge number of servers into Data Centers in order to reduce the total cost of ownership. In addition, sophisticated enterprises are moving to out-source their IT resource management so that they can focus on their own core business. Consequently, Data Centers should be able to manage several thousands of IT resources, that include servers, storage, and networks. In order to decrease the management complexity and to increase utilization of these resources, an innovative GRID based resource management software, a Commercial GRID system, is required. We use the term the Commercial GRID system and the GRID interchangeably.

In the old mainframe days, an IT system integrator could develop a controllable IT system on top of a single solid homogeneous platform. The current IT system integrator, however, must use tens of different APIs on different OSes and middleware platforms and has no consistent way to detect and respond to faults or identify underlying performance bottlenecks. GRID based meta-OS functionalities, provided by the GRID, could ease the burden of IT system integrators by enabling end-to-end QoS.

## 2.2 Customers

A **"GRID administrator"** is an important actor of the Commercial Data Center(CDC). Strictly speaking, the GRID administrator is not a customer but a provider. However, one of the significant benefits of the GRID is increased manageability of the IT infrastructure provided by the CDC. This is one of the key motivations to create the GRID. Since hardware and software management of the CDC is a lot of trouble and costs a lot of money, the administrator demands automation of key functionalities such as provisioning, monitoring, tuning, maintenance, error diagnosis, and repair of components of the IT infrastructure.

One requirement placed on the GRID administrator is to increase the utilization ratio of the IT infrastructure. According to several analysts' reports, actual utilization ratio is often less than 20%. Also some resources are reserved for failover and provisioning; in other words they are not put to productive use. It should be possible to share such resources among multiple systems, with physical location not being the single determining factor whether sharing is possible or not.

The GRID increases infrastructure manageability thereby minimizing the number of GRID administrators, e.g. from a few dozens to less than ten.

An **"IT System Integrator**" is a customer of the Commercial Data Center. The "IT System Integrator" constructs heterogeneous systems, a very difficult task. Problems include making end-to-end performance predictions and guarantees, ensuring a required level of availability is achieved, e.g., 24/7 availability, provisioning so as to respond to unpredictable service demands, e.g., the internet spike problem, while at all time responding to frequent service specification changes.

The "IT System Integrator" expects to reduce the complexity of building distributed and heterogeneous systems by means of an OGSA based GRID, which provides standard and QoS enabled meta-OS functionalities.

The IT system integrator uses the Commercial Data Center for development.

An **"IT business activity manager"** is another **customer** of the Commercial Data Center. The IT business activity manager, for example, runs a "Ticketing service" which sells tickets to **"End**

**Users."** The end users are **actors** of the CDC but are not customers. They are customers of the "Ticketing service."

At the moment only a few IT business activity managers use the CDCs. We expect that in the future hundreds of managers would be using a single Data Center.

The following figure depicts the Data Centers ( = Real Organizations), the IT business activity (= Virtual Organizations), the IT business activity managers, and the GRID administrators. The IT business activity managers create VOs and run their services expecting that the VOs are reliable, scalable, secure, and deliver required QoS. On the other hand, the GRID administrators manage ROs and the GRID alleviates their work.

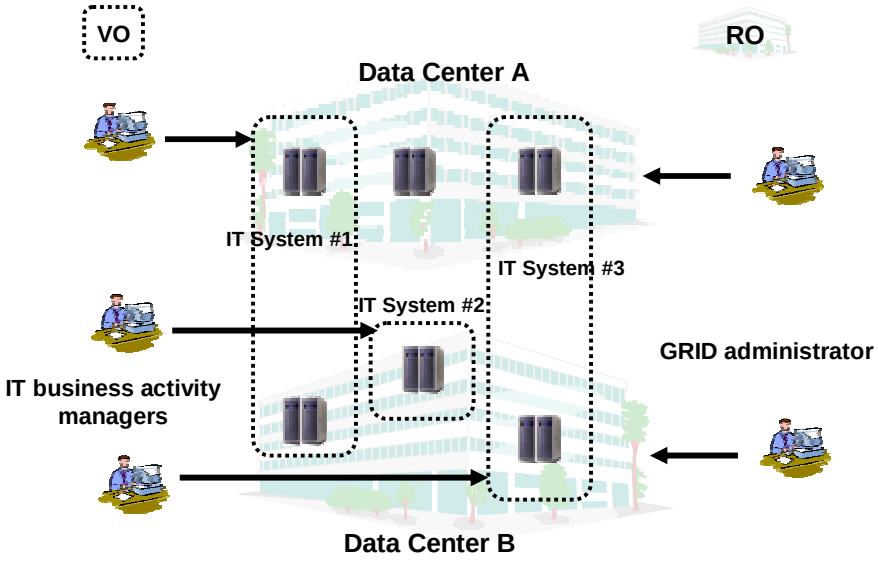



Figure 1. RO, VO, and customers for the CDC

## *2.3 Scenarios*

There are four scenarios for the Commercial Data Center.

### 2.3.1 Multiple in-house systems

Current in-house systems; e.g. personnel management system, finance and accounting system, order-receiving system, and CRM, are mostly isolated. Each in-house system runs on its own IT resources and also keeps extra IT resources for high availability or in preparation for increased workload. Since peak workloads are all different and do not necessarily occur at the same time, there are a lot of idle IT resources.

If the GRID could manage all IT resources in the enterprise and could provide necessary resources to each in-house system on demand, extra resources needed by each system could be shared among several systems leading to better IT resource utilization. Also more in-house system could run on less IT resources.

For each in-house system, the GRID makes reservations in advance, allocates hardware, deploys necessary software and data, and starts the needed applications. All these procedures are automated.

The GRID also provides autonomous management including failover and provisioning. Many defects are handled by the GRID autonomously.

Additionally, multiple remote Data Centers could work together to improve scalability and availability. Undisrupted operation must be ensured even in the event of disasters such as earthquakes, fires, or acts of terrorism. Independent, but networked, Data Centers can be used to provide the necessary physical infrastructure.

### 2.3.2 Limited time commercial campaign

Corporate marketing often plans limited time campaigns; e.g. concert ticket sales, international conference booking, or sales promotion campaigns. Current application systems for these campaigns require fixed IT resources. Thus they need high initial and maintenance costs.

The GRID could provide necessary IT resources on demand and charge based on usage.

IT business activity managers can also chose the most inexpensive Data Center or use multiple Data Centers for scalability and availability.

### 2.3.3 Disaster recovery

IT systems providing essential public infrastructure services, including banking system and air traffic control system, require disaster recovery. They are, however, very high cost and require very high level technical expertise to build and operate. Popularization of the Internet also makes many application systems, e.g, popular web pages like Google, indispensable.

The GRID could provide a standard disaster recovery framework across remote CDCs to these IT business activities at low cost.

### 2.3.4 Global load balancing

Geographically separated CDCs can share high workload and provide scalability for application system.

## *2.4 Involved resources*

A CDC is equipped with all sorts of IT resources including servers, storage, data, and networks.

The GRID should manage at least several thousands of resources.

## *2.5 Functional requirements for OGSA platform*

For the scenario described in section 2.3.1, the following functions are required:

1. **Discovery**

   At first, a customer should pick out a reference to a CDC, which the customer will use. One or more well-known discovery services are used as a first step.

2. **Authentication, Authorization, and Accounting** [1]

   When the customer submits a job request, the CDC authenticates the customer and authorizes the submitted request. The CDC also identifies his/her policies (including but not limited to SLA, security, scheduling, and brokering policies).

[1] This function should be added to OGSA platform functionality.

3. **Advanced Reservation** [2]

   Based on the customer's request the GRID decides when to start the request processing. [3] The GRID interprets the job specification description language in which the request is written. The GRID checks if the customer has the right to perform the request.

4. **Brokering**

   The GRID match-makes the most suitable resources for the requested time period (assuming a request for advanced reservation). Access-control to the resources and quotas are also applied. The reservation is made and its reference is returned to the customer.

5. **Data Sharing**

   Required user data (databases and/or files) is also specified in the job request. Data accessibility should be considered during match-making.

6. **Provisioning**

   Some time before the reserved time, the GRID begins application and user data deployment. In the case of a Java program, the GRID discovers designated java program (jar file) and deploys it into the reserved resource. The deployment feature for Java is already well-defined and supported on most hosting environments.

7. **Scheduling** [4]

   When the reserved time comes, the GRID starts the task.

8. **Metering and Accounting**

   During job execution, the metering service keeps track of resource usage. The information is passed to the accounting service.

9. **Fault Handling** [5]

   Assume that the customer only needs "failure notification" in case his/her job encounters any error and cannot complete successfully (required fault handling is designated as fault management policy).

10. **Policy**

    Several attributes should be handled as policy. A brokering policy defines resource usage quotas per customer. An error and event policy guides autonomous management including provisioning and failover.

11. **Security**

    Isolation of customers in the same Data Center is a crucial requirement. The GRID should provide not only access control but also performance isolation.

For the scenario described in section 2.3.2 "Limited time commercial campaign," the following functions are required in addition to the above:

12. **Virtual Organization**

---

[2] This function should be added to OGSA platform functionality.
[3] "Request processing" and "job processing" are different. In case of advanced reservation, the request processing books resources for future use and job processing is actual job execution at the reserved time.
[4] This function should be added to OGSA platform functionality.
[5] This function is called "Fault Tolerant" in [1]. In order to cover more generic functionality, the function is renamed to "Fault Handling" in this document.

Upon the customer job request the GRID creates a VO in a Data Center which provides IT resources to the job. Depending on the customer's request, the GRID will negotiate with another GRID on remote CDC and create a VO across the CDCs. Such a VO can be used to achieve the necessary scalability and availability.

13. **Monitoring**

    The customer wants to monitor his/her application running on a remote Data Center.

14. **Load balancing**[6]

    The GRID monitors the job performance and adjusts allocated resources to match the load and fairly distributes end users' requests to all the resources.

For the scenario described in section 2.3.3 "Disaster recovery," the following functions are required in addition to the above:

15. **Disaster Recovery**

    In case of the Data Center becoming unavailable due to a disaster such as an earthquake or fire, the remote backup Data Center takes over the application systems.

For the scenario described in section 2.3.4 "Global load balancing," no additional function is required.

## *2.6 OGSA platform services utilization*

The following services are necessary to provide functions in the previous section.

1. **Name resolution and discovery service**

   This service is used for the GRID as discovery function.

2. **Security service**

   This service is necessary for OGSA AAA functionality. Resource access control also need security service.

3. **Reservation service** [7]

   This service is used for advanced reservation.

4. **Brokering service** [8]

   This service is used for resource brokering.

5. **Data management service**

   This service is used for data sharing within Data Center and across them. It is also used for disaster recovery.

6. **Provisioning and resource management service**

   This service is used for provisioning and also creating a VO on a remote site.

7. **Scheduling service** [9]

   This service is used for priority job scheduling.

[6] This function should be added to OGSA platform service.
[7] This function should be added to OGSA platform service.
[8] This function should be added to OGSA platform service.
[9] This function should be added to OGSA platform service.

8. **Metering and accounting service**

   This service is used for metering and accounting.

9. **Fault handling service** [10]

   This service is used for fault handling. It is a part of autonomous management. In case of disaster recovery, affected IT business activities are relocated to other Data Center(s).

10. **Policy service** [11]

    This service is used for policy function.

11. **Monitoring service** [12]

    This service is used for monitoring function.

12. **Deployment service** [13]

    This service is used for provisioning function.

## *2.7 Security considerations*

Each Commercial IT system (corresponding to a VO) should be securely isolated from each other since competing companies may be hosted in the same Data Center (RO). Before starting commercial systems, VOs should be divided using Virtual LAN or equivalent technology. When workload increases, IT resources (e.g. servers) will be reallocated to another system by rearranging the network configuration but no information should leak out.

WS-security is the starting point and some extension may be necessary for the Commercial Grid.

A VO may sit in a single Data Center or across multiple Data Centers. For disaster recovery and wide area load balancing, VOs should use multiple Data Centers.

## *2.8 Performance considerations*

In contrast to the Science GRID, execution speed is not the highest priority requirement for the Commercial GRID. Instead, several Quality of Service matrixes should be considered. A best effort scheme cannot satisfy the Commercial GRID requirements. Since each job request should complete by the specified date and time, deadline scheduling by means of advanced resource reservation is the base-line assumption. Typically, jobs are expected to run for a certain predefined period and provide a certain level of performance.

To avoid the Internet spike problem, adaptive resource allocation (i.e., provisioning) enables scalability of the requests throughput.

Each IT system administrator expresses their requirements in a Service Level Agreement (SLA). Based on the SLA, each job demands additional resources under heavy load or substitute resources when a failure occurs. In case all requests cannot be satisfied, low priority ones, based on SLA, are rejected.

## *2.9 Use case situation analysis*

Several cutting-edge technologies[14] [15] and products[16] already in the market attempt to solve one or more issues described above. Such attempts take a proprietary approach and have limited scope.

[10] This function should be added to OGSA platform service.
[11] The explanation of policy service in [1] is very vague and is not clear what it is.
[12] This function should be added to OGSA platform service.
[13] This function should be added to OGSA platform service.

OGSA, however, is an open, extensible, and comprehensive architecture, which can be used to address these problems.

We are now in research phase [5]. After research completion, we would like to prototype OGSA base GRID.

### 2.10 References

1. Foster, I and Gannon, D. The Open Grid Services Architecture Platform, 2003. www-unix.gridforum.org/mail_archive/ogsa-wg/doc00016.doc
2. Kishimoto, H., Savva, A., Snelling, D. OGSA Fundamental Services: Requirements for Commercial GRID Systems, OGSA-WG document, 14 October 2002 www-unix.gridforum.org/mail_archive/ogsa-wg/pdf00002.pdf

[14] Océano Project, IBM. www.research.ibm.com/oceanoproject
[15] N1, Sun Microsystems. wwws.sun.com/software/solutions/n1
[16] Jareva, http://www.jareva.com

# 3 IT Infrastructure and Management

## *3.1 Two Scenarios*

Since the scenarios for IT infrastructure and management are many, a couple of possible scenarios are chosen for a first/initial set of capabilities determination. More such scenarios can be added in future. The two chosen are:

1. Cycle sharing and job execution (batch and interactive)
2. Provisioning (OS selection, software distribution (on-demand) to satisfy license requirements, apps availability, create Web service example, convert the capabilities and primary focus, limits that are triggered). Will focus specifically on software distribution

Table 1: Typical scenario for cycle sharing in the grid

| Seq. No. | Operations | *3.2 Key Capabilities* | OGSA Service Mapping |
|---|---|---|---|
| 1. | User submits job; job specifies a type and set of resources | *Discovery*: Discover the queue to submit to<br>*Authentication*: System authenticates the user<br>*Authorization*: Queue manager validate authority of user to queue job<br>*Resource specification Language*: to specify the resources<br>*Fault Tolerance*: Faults in locating queue, transmission errors, automated routing.<br>*Encryption*: Encrypt the communication at the very least the payload | **Services**<br>• Registry<br>• Authentication<br>• Authorization<br>**Schemas, Protocols**<br>• Resource Specification Language or Framework<br>**Horizontal capabilities (required in all participating services)**<br>• Fault tolerance<br>• Encryption |
| 2. | Job is queued | *Queue*: To store the job<br>*Notification/Messaging*: Status of submission communicated to "user"<br>*Logging*: log status | **Services**<br>• Generalized Queuing<br>• Notification<br>• Messaging services (reliable delivery)<br>• Logging<br>**Schemas, Protocols**<br>• Event schema<br>• Message schema<br>**Horizontal capabilities (required in all participating services)**<br>• Fault tolerance<br>• Encryption |
| 3. | Job is scheduled | *Resource selection*: Resources matching requirements are determined<br>*Brokering and arbitration*: Multiple requests for resources and managed and conflicts resolved using defined *policy*.<br>*Scheduler*: matches job to resources<br>*Reservation*: All resource determined are reserved<br>*Data staging and provisioning*: Ensure that data required for computation is available in the highest performance repository. Data storage space for temporary of storage of intermediate computation made available.<br>*Logging*: log status | **Services**<br>• Registry<br>• Resource selector<br>• Broker<br>o Reservation<br>• Scheduler<br>• Common Resource Model services<br>• Data management services<br>• Logging services<br>**Schemas, Protocols**<br>• Policy schema<br>• Resource specification<br>• Resource description<br>• Reservation<br>**Horizontal capabilities (required in all participating services)** |

| | | | |
|---|---|---|---|
| | | | • Policy framework<br>• Fault tolerance<br>• Encryption |
| 4. | Job dispatched to "consume" the resources and begins execution | *Hosting environment*: Ensure that hosting environment is available and initialized.<br>*Validate user and enable execution environment*:<br>*Staging*: Make sure that all entities for job execution available (app, data, credentials, authority)<br>*Data Migration*: Data is migrated to the hosting environment that will execute the job (or made available in or via a high performance infrastructure)<br>*Fault tolerance*: Ensure that collaterals available. Handle exceptions.<br>*Monitoring*: Reporting job status and resource consumption<br>*Quota management*: Watch and manage the resource consumption of job. Enforces SLA or contract between consumer and provider. | **Services**<br>• Registry<br>• Authentication<br>• Authorization<br>• Caching services<br>• Data services<br>• Hosting services (or provisioning services)<br>• SLA manager<br>**Schemas, Protocol*s***<br>• SLA schema<br>**Horizontal capabilities (required in all participating services)**<br>• Fault tolerance<br>• |
| 5. | Job executes | *Hosting environment:* Job executes in environment<br>*Single application:* Standard execution profile<br>*Flow i.e. multiple applications:* Many applications can be wrapped in scripts or process that requires licenses and other resource from multiple sources. Applications co-ordinate using the file system or based on events. | **Services**<br>• Services co-ordination<br>• Data grids<br>**Schemas, Protocol*s***<br>• Event schema<br>• Logging schema<br>**Horizontal capabilities (required in all participating services)**<br>• Fault tolerance |
| 6. | Job completes | *Logging:* Log completion/error status | **Services**<br>• Logging<br>**Schemas, Protocol*s***<br>• Event schema<br>• Logging schema<br>**Horizontal capabilities (required in all participating services)**<br>• Fault tolerance<br>• Encryption |

Table 2: Typical scenario for software provisioning

| Seq. No. | Operations | *3.3 Key Capabilities* | OGSA Service Mapping |
|---|---|---|---|
| 1. | User needs to execute application (could be a need for a proxy i.e. a batch job or done interactively) | *Authentication*: System authenticates the user<br>*Authorization*: User obtains the credentials in the local VO and possibly remote VO if the application is remote.<br>*Fault Tolerance*: Faults in locating queue, transmission errors, automated routing. | **Services**<br>• Registry<br>• Authentication<br>• Authorization<br>**Schemas, Protocols**<br><br>**Horizontal capabilities (required in all participating services)**<br>• Fault tolerance |
| 2. | A set of servers that can serve the applications is found; server can be a peer that is near in network segment or a dedicated server. | *Registry*: Lookup the registry to determine location and handle to available application. Determine if it is local or remote<br>*Resource specification*: To provide inventory of applications on machine and record in the registry<br>*Notification/Messaging*: Registry notified of application inventory and changes to inventory<br>*Fault Tolerance*: Manage alternative application servers or . | **Services**<br>• Registry<br>• Notification<br>• Messaging services (reliable delivery)<br>**Schemas, Protocols**<br>• Resource specification language<br>• Message schema<br>**Horizontal capabilities (required in all participating services)**<br>• Fault tolerance |
| 3. | License requirements are evaluated | *Resource selection*: Resources matching requirements are determined; available licenses are determined<br>*Policy schema*: Specify policy<br>*Brokering and arbitration*: License scheme is evaluated against policy (this is only if there are multiple license types are supported for the same application)<br>*Reservation*: All licenses required by application are reserved<br>*Logging*: log status | **Services**<br>• Resource selector<br>• Broker<br>o Reservation<br>• Model services<br>• Logging services<br>**Schemas, Protocols**<br>• Policy schema<br>• Resource specification<br>• Resource description<br>• Reservation<br>**Horizontal capabilities (required in all participating services)**<br>• Policy framework<br>• Fault tolerance |
| 4. | Application is copied to required workstation and installed | *Hosting environment*: Ensure that hosting environment is available and initialized.<br>*Data Migration*: Application is migrated to the computer that will execute the application<br>*Fault tolerance*: Ensure that collaterals available. Handle exceptions.<br>*Monitoring*: Reporting installation resource. | **Services**<br>• Registry<br>• Authentication<br>• Authorization<br>• Data services<br>• Hosting services (or provisioning services) |

| | | | |
|---|---|---|---|
| | | | **Schemas, Protocol*s***<br>• SLA schema<br>**Horizontal capabilities (required in all participating services)**<br>• Fault tolerance |
| 5. | Monitor application usage for auditing and billing | *Monitoring*: Reporting job status and resource consumption<br>*Metering*: Record the usage and duration; especially meter the usage of licenses.<br>*Auditing*: Audit usage and application profile on machine<br>*Billing*: Based on metering bill the user. | **Services**<br>• Monitoring and Logging<br>• Metering<br>• Billing<br>**Schemas, Protocol*s***<br>•<br>**Horizontal capabilities (required in all participating services)**<br>• Fault tolerance |

### *3.4 Resources and Services*

Infrastructure should support:

- Geographically distributed environment with both varied usage, management and administration policies across organizations and government/civil policies that need to be honored and managed.
- A high level of security and management of multiple security infrastructures.
- Heterogeneous platforms and varied hosting environments
- A global, cross-organizational and consistent view of resources and assets for project and fiscal planning
- Usage models that provide batch and interactive access to resources.
- Applications that can be single process, multi-process (local and distributed) and flows (i.e. multiple applications stitched together with intermediate processing and automated provisioning for data and/or resources).
- Support to enable, manage and monitor data usage (spatially, temporally and quantity (for example: disk space management), email management).
- Automated provisioning to meet peak demand. For example: bring more Web servers on line as demand exceeds a threshold. Permit multi-use environment that can be flexibly transitioned to the different tasks as required.
- Need to optimize resource usage while meeting cost targets (i.e. deal with finite resources). Mechanism to manage conflicting demands from various organizations, groups, projects and users and implementing a fair sharing of resource and access to grid.
- Need an environment that can represent policy at multiple stages in the hierarchies to automate the policies that are implemented as organizational processes or managed manually.
- Need a high degree of fault-tolerant environment (fail-over, redistribution of load).

- The self-healing capabilities of resources, services and systems are required. Significant manual effort should not be required to monitor, diagnose and repair faults. Ability to integrate intelligent self-aware hardware such as disks, networking devices etc.
- Need strong monitoring the environment for defects and ability to identify misuses including virus/worm attacks. Ability to migrate attacks away from critical areas.
- Mechanisms to self-organize and self-describe so that configuration of the environment is manageable. (System should automatically manage low level configuration based on administrator set higher-level configurations and management. Reduce personnel headcount)
- Be able to “codify” and “automate” the normal practices used to manage the environment.
- A requirements definition language or schema to specify and identify resources.
- SLA or contract violations by all available parties should be tracked and flagged.
- Applications and schemas for metering, auditing and billing.

# 4 National Fusion Collaboratory

## 4.1 Summary

The National Fusion Collaboratory (NFC) project [2] defines a virtual organization devoted to fusion research and addresses the needs of codes developed and executed by this community. Up to now, these developers of these codes would typically port their software to a standard set of platforms; community users would then install and use this software on their machines. This process was found to be complex from the provider's as well as user's viewpoint. The user would have to go through the (usually complex) process of installing the code and its dependencies, would then have to maintain that code and also update the installation whenever a new version comes out. This process is made especially difficult by the fact that scientific codes are typically developed and refined over decades and result in very complex systems which need to be updated frequently in order to reflect the latest improvements in modeling and simulation techniques. From the provider's point of view, the necessity of supporting the code on even a limited set of platforms can require significant cost and effort. In addition, maintaining and debugging a community code on an unfamiliar platform can mean that a significant amount of effort is spent simply in reproducing, let alone fixing, a problem.
Due to these problems, the fusion community recently decided to adopt the application service provider (ASP) model, also known as the "network services model". In the "network services" model, a code, as well as a set of familiar platforms is provided or contracted by a service provider and made accessible remotely to clients. The service provider undertakes not only to maintain a reasonable set of versions of the code, but also to debug and otherwise manage client's runs to ensure that they achieve their objective. This might include executing the code as efficiently as possible, executing it within a certain time bound, or producing results of certain accuracy (see next section for details). The clients specify those objectives and execute the codes remotely thus avoiding maintenance costs. This sharing paradigm is new to the Fusion community, but is rapidly gaining acceptance as it encourages sharing of software and hardware resources and frees the researcher from needing to know about software implementation details and allowing a sharper focus on the physics.

## 4.2 Customers

The customers of this use case are fusion scientists. Of those, service providers defined above seek to reduce maintenance costs by providing a service on a familiar set of platforms, while service clients seek to obtain remote execution of a code satisfying certain objectives, specifically capable of executing within certain timebounds during fusion experiments. Two principal issues arise in this environment: (1) issues of trust and (2) issues of control. Issues of trust address questions such as: will my code run get priority when I need it? How do I enter into contract with code/hardware resource provider? What guarantees do I have that this contract will be observed? And, on the provider's side: how can I ensure that my deployment is secure and yet deal with a dynamically changing community of users? The issues of control deal with questions like: how do I provide reliable execution in this environment? How can I meet client's demands? All of these issues need to be addressed in wide-area national, and eventually international, deployment comprising hundreds and later potentially thousands of users.

Below we summarize in detail needs of the clients as well as servers.

*QoS-based execution during fusion experiments.* Magnetic fusion experiments operate in a pulsed mode producing plasmas of up to 10 seconds duration every 15 to 20 minutes, with multiple pulses per experiment. Decisions for changes to the next plasma pulse are made by analyzing

measurements from the previous plasma pulse (hundreds of megabytes of data) within roughly 15 minutes between pulses. This mode of operation could be made more efficient by the ability to do more analysis and simulation in a short time using codes running on remote resources *only if* their execution time could be guaranteed. Given the present capabilities, the decision to include a new code in the "between pulse" analysis usually involves buying a new cluster that will be run on-site and dedicated during the experiment; obviously this mode of operation does not scale in the long run. The ability to run codes on remote resources would be helpful, on the condition that end-to-end quality of service (QoS) guaranteeing the execution within certain timebounds could be provided. For example, end-to-end quality of service should combine input and output data transfer with execution time and ensure the execution of this QoS-based workflow in such a way as to meet the user's overall QoS requirement.

*Availability contracts.* Like in many other scientific communities, much of the work in the Fusion community is driven by the need to make results available in time for major conferences. Although the current deployment has not yet been found lacking in this respect, we anticipate that resource utilization before such events will grow to the point where some users's requests will not be fulfilled due to high demand. The resolution of this problem could be provided by a contract mechanism whereby the user contracts for the availability of a service ahead of time, and claims it when the need arises.

*Use policies.* Both of the client needs described above require mechanisms for use policy specification and enforcement on the part of service/resource provider as well as the virtual organization. The service provider for example has the need to assert who (which groups or users) have the right to run certain codes, how many hardware resources they can use, what availability contracts they can enter into and under what circumstances, how service execution can be managed etc. Such use policies also have to be suitably enforced by the underlying resource management system.

*Flexible delegation of rights*. Providing seamless maintenance of a client's run requires flexible rights delegation policies for the server. For example, if a run is found to experience an unexpected failure, the service provider may want to diagnostic the run, debug and restart it. Since the run may involve access to secure databases, in order to perform these actions, the service provider will need to acquire rights that allow it to reproduce this usage pattern. Impersonating the client is not necessarily a reasonable option as that may give the service provider too many rights, and the client may be unwilling to do this.

*Community accreditation.* The clients would like to be able to use community services by getting accredited with the community rather than each individual service provider. For example, code execution on a hardware resource (which may not even be known to the client) should not be associated with the need to obtain an account on that resource. Instead, a mechanism is needed whereby it is sufficient for the client to present community credentials in order to initiate the run.

### *4.3 Scenarios*

In the experimental scenario described above, a scientist at one of the NFC sites (a client site) needs to remotely run code installed and maintained at another NFC site (a service provider site) during an experiment within time bound *T* (typically on the order of 10 minutes). For a very simple execution, the following would be available on the service provider's side: a script that will download experimental data for the application input once that data becomes available, a suitable "short-running" configuration of an application, capable of executing in less than *T* (some application may be available in multiple configurations reflecting accuracy/time trade-offs), a script delivering results to the client, as well as an execution plan, or a workflow, describing the

sequence of these actions and their QoS dependencies. To ensure that the code executes with the required QoS (in this case: within time *T*), the scientist at the client site enters into a contract with the application server and as a result is guaranteed code execution within *T* any time it is requested during the experimental availability window (typically a day). Since only a few such executions may be requested during that day, and the service provider resources have to be shared with other clients, it is essential that resource allocations not be overgenerous and that other codes can share the resource with the time-critical application, getting preempted whenever the situation requires.

When the client claims the execution based on the contract, the service provider initiates and monitors the run, adaptively recovering from failure of specific actions if needed. Depending on the importance of the run the service provider may have overprovisioned, or replicated the run.

This scenario may be more sophisticated depending on the service in question. It is essential that the execution time or other QoS aspect experienced by the client is end-to-end, in other words accounts not only for application execution but also allows for database access, data transfer, and other activities. It is important to note that data availability before transfer time (replication) cannot be leveraged in this case as it becomes available dynamically. Similarly, in national (and potentially international) deployment data transfer will become a significant factor which cannot currently be reliably managed. Also, it is important that the QoS-base execution is available to small fusion labs in small centers as well as large fusion labs in large centers.

Apart from the time, fusion codes can also require non time-critical mode of execution but one that provides accurate results, or the time requirement can be relaxed to complete by a certain deadline rather than in a specific amount of time. More details of the scenario are described in [3].

### *4.4 Involved resources*

The primary resources involved include

1. The hardware resource at service provider site; these can range from supercomputers to single workstations.
2. The machines running the client's sites.
3. Networks between Fusion sites (the service provider sites and the client site), they are widely distributed, potentially internationally distributed.

The services to be delivered primarily relate to service executions, however may involve hardware services in the future.

### *4.5 Functional requirements for OGSA platform*

This use case uses the following OGSA functionalities as described in [1]:

1. **Discovery.** The clients need to discover network services before they are used. Service brokers need to discover hardware and software availability.
2. **Workflow management**. A fusion grid network service is a workflow of multiple components (remote execution, input and output data transfer, etc.).
3. **Scheduling of service tasks**. The service provider (or broker acting on service provider's behalf needs to schedule resource in order to meet the execution constraints requested by the client. The scheduling can take the form of advance reservation.
4. **Disaster Recovery**. As the service provider (or broker acting on itse behalf) strives to meet the client's end-to-end constraints, some degree of adaptation may have to be used to prevent failure.

5. **Provisioning**. Provisioning is required to reserve obtain CPU time, applications, licenses, storage, and potentially networks.
6. **Brokering**. The service broker identifies codes and platforms suitable for execution requested by the client.
7. **Load Balancing**. Some load balancing may be required to use service provider resource more efficiently.
8. **Fault Tolerance.** A reliable solution is needed in order to provide the time-critical execution capability.
9. **Transport Management**. Reliable transport management is essential to obtain the end-to-end QoS required by this application.
10. **Legacy Application Management**. Realizing the Grid potential to deal with legacy issues was the one of the foremost motivation for this project.
11. **Services Facilitating Brokering**. This capability is essential for the service broker to compose and later execute an workflow meeting the requested constraints.
12. **Application and Network-level Firewalls**. This is a long-standing problem in the fusion use case. It is made particularly difficult by the many different policies we are dealing with and particularly harsh restrictions at international sites.
13. **Agreement-based interation**. This project requires agreement-based interaction capable of specifying and enacting agreements between clients and servers (not necessarily human) and then composing those agreements into higher-level end-user structures.
14. **Authorization and use policies**. We also require use policy specification and enforcement mechanisms as described above.

### *4.6 OGSA platform services utilization*

The following services are necessary to provide functions in the previous section.

1. Name resolution and discovery service
2. Security service
3. Provisioning and resource management service
4. Metering and accounting service
5. Policy service
6. Messaging and logging
7. Monitoring service
8. Metering and Accounting
9. Administration
10. Service Orchestration

### *4.7 Security Considerations*

The server sites need the ability to provide authorization on the usage of certain codes (or application services) as well as on the usage of resources. The VO-specific authorization policies need to be maintained centrally, while resource-specific policies need to be maintained by resource owners.

In addition, application service providers need to be able to assume a subset of user's rights needed to debug an application that has gone astray. This is needed because applications access the experimental database based on the rights of the user that started the run. Frequently, the application provider is able to debug and resubmit the user's program in a manner transparent to the user.

### 4.8 Performance Considerations

The ability to deliver services in real-time is essential. Also important is the ability to satisfy other QoS constraints (application-specific notions of accuracy).

### 4.9 Usecase situation analysis

Some of the required capabilities have already been provided by Globus as evidenced by the fact that fusion services are deployed and successfully used by the community. We are currently researching enforcement issues, issues of agreement-based interaction, as well as scheduling and adaptive techniques that would support them. We also require changes in the security model and advances in overcoming deployment issues such as firewalls.

### 4.10 References

1. Foster., I. and Gannon, D. The Open Grid Services Architecture Platform. www-unix.gridforum.org/mail_archive/ogsa-wg/doc00016.doc.
2. Keahey, K., Fredian, T., Peng, Q., Schissel, D.P., Thompson, M., Foster, I., Greenwald, M. and McCune, D. Computational Grids in Action: the National Fusion Collaboratory. Future Generation Computing Systems (to appear), 18 (8). 1005-1015.
3. Keahey, K. and Motawi, K. Taming of the Grid: Virtual Application Services, Argonne National Laboratory, Mathematics and Computer Science Division Technical Memorandum ANL/MCS-TM-262, 2003.

# 5 Severe Storm Prediction

## *5.1 Customer*

A consortium of meteorologists and environmental modelers are attempting to build a Grid to predict the exact location of severe storms such as tornadoes based on a combination of real-time wide area weather instrumentation and large-scale simulation coupled with data modeling. The virtual organization is widely distributed and often mobile.

The scenario is roughly as follows. Instrument data streams from Doppler radar, satellite imaging, ground-based sensors such as pressure, temperature and humidity detectors, are constantly monitored by data mining agents looking for dangerous patterns. When one is detected, VO members are notified and a large number of simulations are launched automatically. Data mining tools are configured to scan the output of the simulations and compare the results against the evolving data stream from the instruments. Data archives are searched for similar patterns. Some of the instruments are automatically reconfigured to refine the data streams.

As the storm evolves additional simulations are launched to refine the resolution of the predictions. Humans, are not monitoring the entire process and aiding in the process by steering some of the simulations. (The simulations generate output files which can be visualized as animations.) Other individuals on the ground are entering more data from mobile devices. The authorities and media are notified of the predictions.

This scenario is not yet possible because the Grid infrastructure is not yet in place. At the present time, many of the various components exist, but they are not all integrated. The current activity for this group is collaboration on testing the simulation and data mining and integrating the simulations with the data streams.

## *5.2 Resources involved and the services that are being delivered*

The primary resources involved include

1. the sensor network courtesy of several agencies.
2. the data archives of past storm activity and instrument readings
3. the compute resources including the Teragrid resources

The services to be delivered:

- An integrated grid allowing VO members access to the simulation and data mining tools, the data archives and the sensor network tools.
- Eventually an automated, autonomic Grid of services that carry out the scenario described above.

Required Services:

1. VO management. Who has access to what parts of the instrument, data, compute resources is very important.
2. Security (authentication and authorization) for VO members. Also mobile access
3. Datagrid services: metadata catalogs, directory and index services, grid-wide access to data archives, virtual data management
4. Grid-wide monitoring, messaging, event systems and logging services

5. Most instruments will have webservice access interfaces
6. Workflow engines to orchestrate the coupled simulation/datamining/visualization tasks.
7. Compute and data resource brokering services. Scheduling and co-scheduling services will be needed.

### 5.3 Security considerations: environment and threats

- Access to much of the sensor network output is public, however the ability to re-deploy it on-the-fly is very restricted.
- The simulation and datamining run on very high-end compute resources and require special authorization to demand computing of this scale on-demand.
- The VO needs to be able to share resources and access as much as possible.
- The ability to authenticate VO members on mobile devices is essential.

### 5.4 Performance considerations

The ability to make better-than-real-time predictions is very important. Resource allocation must be very efficient. It is expected that many of the compute scenarios will require dynamic reallocation of resources as needed.

### 5.5 Lifetime and evolution

This Grid must be always available. Autonomic capability is an essential component.

### 5.6 Situation analysis

# 6 Virtual Organization Grid Portal

## 6.1 Summary

Given that the grid enables people to be members of many VOs and each VO gives one access to various computational, instrument-based, data and other types of resources, it is very natural for these VO's to produce a Grid portal which provides an end-user view of the collected resources available to the members of the VO. By producing a portal with "one-stop" shopping for users who participate in a VO, the VO makes its resource much more useful and accessible for their users.

- These grid portals have several elements in common:
- Provide a public-face for the VO with various outreach and informational materials
- Provide a set of collaborative tools (discussion, file storage, calendar, announcedments, etc)
- Provide access to any large data stores which are available to the members of the VO
- Provide the ability to make use of any computational resources available to the members of the VO

These portals are usually a combination of web-based and other tools. Typically core functionality is provided via grid-enabled web servers while more sophisticated tools are deployed to user's desktops.

Given that there are a number of common elements which can be reused across multiple grid portals, and to simplify the user experience as they move from one portal to another, it is important to develop best practices and techniques for the development and deployment of Virtual Organization Grid Portals.

## 6.2 Customers

The customers of this capability are effectively

## 6.3 Scenarios

There are an increasing number of grids where the focus is collaboration centered on some scarce physical resource. Often these resources are so large or so expensive that there can only be a very small number of installations across the world. Some of the examples of this type of collaborative activity include Astronomy (NVO, EVO, JVO, or Sloan Digital Sky Survey), High Energy and Nuclear Physics (ATLAS, CMS, LIGO, or D0), fusion research (FusionGrid), earthquake engineering (NEESGrid), and others.

These broad collaborative efforts generally have the following attributes:

- Geographically dispersed access to computation, data and instruments
- The need for environments for participants to meet and work together across large geographical distances

Most of these collaborative activities are by their nature world-wide and cross-organizational. Within the collaboration there are many groups of varying sizes which are dynamically formed to work on a wide range of problems from experiment design, experiment scheduling, equipment operations, management, publication of results, and many others. All of these groups must operate with members scattered around the world in any time zone.

For these collaborations it is very important to maintain the security of the data, ideas, and the interactions of each group. While there is overall collaboration in the use of the equipment, there is often competition between subgroups within the collaborations in their pursuit of research results. In addition to proper security and access control are absolutely necessary when dealing with the control and operation of any type of experimental equipment or the monitoring of the real-time data as it comes from the experimental equipment.

### 6.4 Involved resources

Explain all resources managed and provided by the Grid system. E.g. what hardware, data, software might be involved.

Are these resources geographically distributed? How many resources are involved in the use case?

### 6.5 Functional requirements for OGSA platform

Review section 3.2 of OGSA platform document which proposes functionalities of OGSA platform and choose necessary requirements from them.

Explain which of these functions your use case needs and how it uses them in detail.

If desired function is not included in section 3.2, you should specify what function is required.

### 6.6 OGSA platform services utilization

Review section 4.2 of OGSA platform document which proposes services of OGSA platform and choose necessary services from them.

Explain which of these services your use case uses and how it utilize them in detail. Please include figures if possible.

If desired service is not included in section 4.2, you should specify what service is required.

### 6.7 Security considerations

Draw up environment and possible threats of the use case.

### 6.8 Performance considerations

Explain performance considerations of the use case.

### 6.9 Usecase situation analysis

A discussion of what services relevant to the use case are already there, to what extent they are satisfactory/unsatisfactory, and an articulation of what else needs to be done.

### 6.10 References

List up references to further reading

# 7 Online Media and Entertainment

## *7.1 Summary*

To deliver an entertainment experience, several actors form a VO for this purpose. In a first step we want to focus on the following roles of actors:

- A consumer who consumes the entertainment content
- A service provider that hosts the entertainment content
- A publisher that offers the entertainment content
- A developer that consumes the entertainment Content.

Each roles may be consists of multiple companies, while the entertainment content may consists of many different forms (e.g. move on demand or online games) with different hosting capacity demands and lifecycle. Therefore one of the primary focuses of this use case is to facilitate the ability to dynamically manage resources based on workload demands and current system configuration. During the lifetime of an entertainment content the actors involved in the delivery of the content may change. During the lifetime of a company the entertainment contents it has to deal with may also change. There the other primary focus of this use case is to provide standard interfaces to allow dynamic and open collaboration.

## *7.2 Customer and their need*

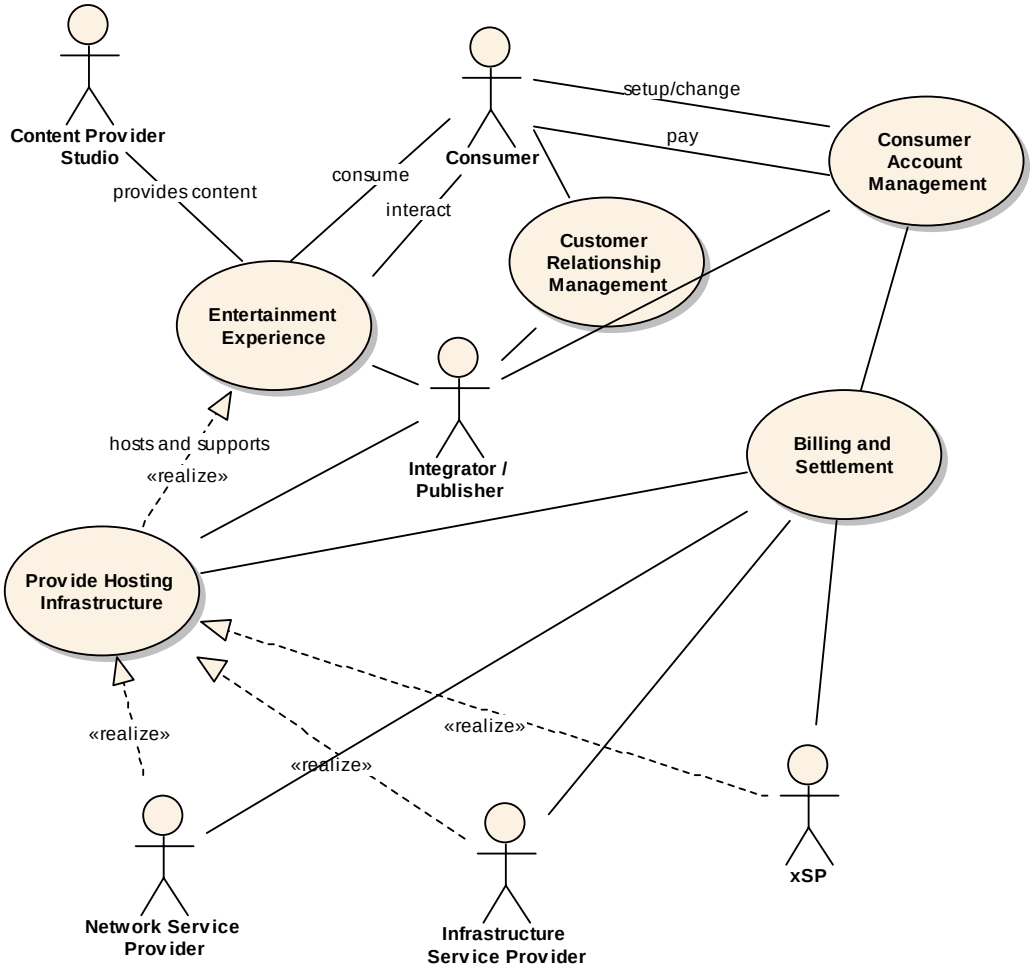
There are two main categories of entertainment experiences with each having unique requirements on the infrastructure that delivers it: consumption and interaction. Consumption of content (e.g. video on demand) does not require a lot of user interaction. Other contents, such as online games, require a lot of user interaction and it is very important to guarantee response times for these contents.

Online entertainment has seen a great adoption over the last couple of months. However, it is still in it's infancy in the areas of content, **business models and infrastructure**. With more online content available, differentiation from competitors will become more important. New commercial opportunities will emerge, for examples usage-based pricing or subscription models for premier consumer experience. Commercial transaction will be tied to entertainment or even inherent to the end user experience.
Because this is a new area, content developers **lack competency in programming for a distributed network**. There is no standard architecture or even best practice for how the back-end datacenters used to deliver the contents are designed. The most common practice today is to design one **stovepipe solution** for each game title, and **manage each solution separately**. Consequently, infrastructure and components deployed for each game are **not reusable**. Furthermore, these stovepipe solutions are designed with a particular level of workload assumed (e.g. 10,000 concurrent users), and scaling beyond this initially assumed workload requires major redesign. As a result, today's datacenters are either over provisioned, or overstressed to the point that service outage occurs. Finally, to make everything even worse, when a game is first designed, there is no way to tell how long the lifetime of the game is going to be. That is, the datacenters for these games may only be needed for only a few days (for a beta-test environment) or needed a few years (e.g. Everquest).

## *7.3 Scenario*

In this scenario, there are 4 actors: consumers, service providers, publishers, and developers. An consumer, for example a game player, will access a portal and authenticate as a known identity. With this authorization he is then able interact with his account or consume an offered

entertainment experience, e.g. play an online game. There may be several providers working in concert with each other. For example a network service provider that offer bandwidth, a hosting capacity provider, who provides server and storage resources, and an application service providers that offer common services like online game engines, standard customer relationship management and helpdesk applications or billing applications. The content provider or studio provides the media content, artwork and game play that the consumer will experience. The integrator or publisher ties the offering together and exposes it to the consumer. The figure below shows some simple interaction between these actors. The interactions between actors may change, and the entertainment content may change as well, therefore it is a key requirement to be able to autonomically manage resource allocation as well as enabling dynamic discovery and interaction of provided infrastructure and services.



The following table lists the main behaviors of each of the actors.

| Consumer | Publisher | Studio | xSP |
|---|---|---|---|
| Sign up for account (with xSP) | | | Create Account |
| | Create a user subscription offer | | |
| | | Purchase and subscribe to hosting environments | Create a business offer for publishers (environment on demand) |
| | | | Provide subscription to game environment (includes reserve/scheduling and purchase) |
| | Delete a user subscription offer | | Delete offering |
| Subscribe to contents/game(s) (with Publisher) | Create Authorization | | Retrieve Authorization information |
| Authorization/authentication | | | Authorization/authentication |
| Find Content | | | Publish available contents |
| Create a M&E session | | | |
| Retrieve / use content | | | Create the On Demand hosting environment (provisioning, failover, workload management) |
| | | | Monitor Resources |
| | | | Add a physical resource |
| | | | Add new functionality /service |
| | | | Upgrade functions / services |
| | | | Delete an environment on demand offer |
| | | | Delete a physical resource from pool of servers |
| | | | Delete resources / services |
| | | | Load balancing |
| | | | Error capture, Problem Determination, Failover, and Recovery |
| | Define meter requirements | | Meter usage |
| Apply a client patch/PTF | | | |
| | | | E-Commerce Intergration |
| | Generate billing record based billing and rating packages | | Generate billing record based on billing and rating packages |
| | Bill player for usage (monthly, per hour, etc) | | Bill publisher for usage/footprint |

### *7.4 Resources and Services*

The datacenter of online entertainment consists of at least the following components in a potentially distributed environment.

- Distributed Server
- Networked storage
- Secure network (including multiple levels of firewalls)
- Player Consoles

The online entertainment business includes at least the following functions:

- Security services (authentication / authorization, identity mapping, etc.)
- Financial services (billing, rating, accounting, etc.)
- Contracting / settlement services
- Customer relations services (logging and data mining of user behaviors)

- Management service (capacity management, workload management)
- Media / Entertainment specific services (e.g. multimodal input)

To solve the problem identified in section 2 the infrastructure hosting the online entertainment environment has to:

- Allow dynamic composition of standard pluggable components (e.g. billing service, customer relations service)
- Be secure and trusted.
- On Demand capacity (autonomic scalability according to workload)
- On Demand aggregation / selection of new services.
- On Demand integration with other companies that has needed competencies.
- There are currently major trust barriers in the online gaming industry, where publishers are very reluctant to share resources / components. To overcome this trust barrier, the components must be based on industry standard interfaces, and must be plug replaceable (i.e. the flexibility to choose components from a wide selection of providers).
- enable new commercial business models
- apply to needs of online game applications

More specific functional requirements, illustrated by specific examples, are listed in the sections below.

## *7.5 Functional Requirements*

### 7.5.1 Discovery

OGSA services must be **discoverable at both runtime and setup time**. For example, a game developer needs to discovery a set of rendering engines and choose to use a particular one based on the end user's screen resolution and connection bandwidth.

OGSA discovery must support **masking**; more specifically, render some services undiscoverable based on, amongst other things, a user's authorization and service level. There are different trust levels between companies. A company may want to expose all components of its software stack to a company that has a joint development agreement in place, but hide these components from other companies.

### 7.5.2 Instantiate new services

New service instances may need to be instantiated. For example, when an additional 2000 players joins an online game, a new game server needs to be provisioned to host these additional players. To provision the new server, the necessary services needs to be instantiated, and there are two aspects to this instantiation: deployment and scheduling/dispatching. Deployment involves transporting the necessary file / data to the server. An example of scheduling / dispatching may involves 1) reserving server resources for a period of time (e.g. reserve 2 hours to run AI logic) 2) determine the order of execution and whether the reservation can be met, and 3) dispatch the appropriate process when the scheduled time arrives.

### 7.5.3 Service Level Management

On of the biggest service level to be managed for online entertainment world is response time. For example, guarantee 50 ms response time for first person game, and 100ms for RPG.

### 7.5.4 Metering and Accounting

Resource usage needs to be logged with respect to each consumer and each provider. This information will be used to charge the consumers based on their usage, as well as used for cost analysis by the providers to determine the pricing.

### 7.5.5 Monitoring

The resource or service owners need to surface certain states so that the user of those resources or services may manage the usage using those state information.

### 7.5.6 Policy

There may be policies at every level of the infrastructure from the low level policies that governs how the resources are monitored and managed to high level policies that governs how business process such as billing are managed. High-level policies are sometimes decomposable into lower-level policies.

### 7.5.7 Grouping / Aggregation of Services -- based on policy and functional requirements

Taking on-line games as an example, the game developers lack competency in many areas such as network programming, rating and billing, eCommerce integration, etc. Therefore, composing services using existing services is a core requirement. There are two main types of composition techniques needed by the online gaming developers: selection and aggregation. **Selection** involves choosing to use a particular service amongst many services with the same operational interface (e.g. select the *fastest* MP3 encoder.) Aggregation involves **orchestrating** a functional flow (workflow) between services. For example, the output of accounting service is fed into the rating service to produce billing records. One other basic function required for aggregation services is to **transform** the syntax and/or semantics of data or interfaces.

### 7.5.8 Security

In such a flexible environment, resources will over time be used for multiple content titles. Therefore trust has to be built on the side of the content providers that such a dynamic environment will not interfere with the goal of consistent user experience. Proper isolation between content offerings also has to be ensured. This level of isolation has to be ensured by the security of the infrastructure.
In addition, several securities related services are required:

- Single sign-on needs to be supported. A player may traverse several organizations in the M&E environment. For example, a player of Everquest may buy a Everquest character on e-bay and pay for it via his pay-pal account. To support single sign-on a game developer may want to use a 3rd party authentication and authorization service, identify mapping service, etc.
- Digital rights management and key management.
- Intrusion detection and protection

### 7.5.9 Certification

A trusted party certifies that a particular service has certain semantic behavior. For examples, a company will only use e-commerce services certified by yahoo shopping.

### 7.5.10 Lifecycle / Change management

Upgrade services or retire services with minimal impact to deployed and/or running services. This could be accomplished by a workflow which provisions the required services, and dynamically modify the current running environment by modifying its selection rules and / or workflows.

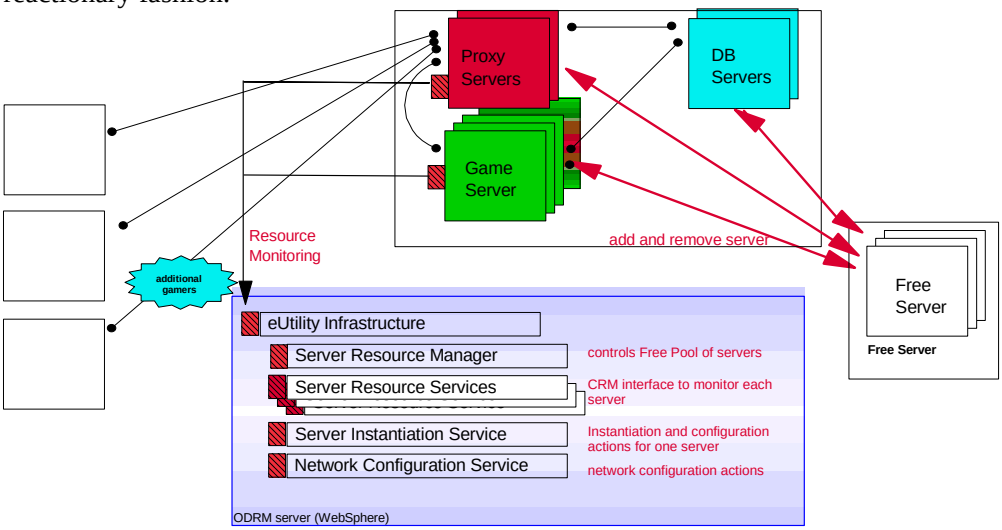
### 7.5.11 Failure Management

OGSI soft state management could be one way to implement a heartbeating function. Resource instrumentation can provide additional information about how well resources are functioning. Logging service is needed to keep track of resource's history of performance and is necessary for

first error capture. Capture failure and trigger recovery actions. For example, when a game server's performance is degraded because of a software problem, apply patch.

### 7.5.12 Provisioning Management

Take online gaming as an example of the M&E industry. On-line games' workloads are very close to uniform sinusoidal waves, but typical server farms are still only about 20% utilized. It is ideal for the providers of the data centers to not over provision for the peak workload, but instead, use just enough capacity to meet the required service level agreements in both a predictive and a reactionary fashion.



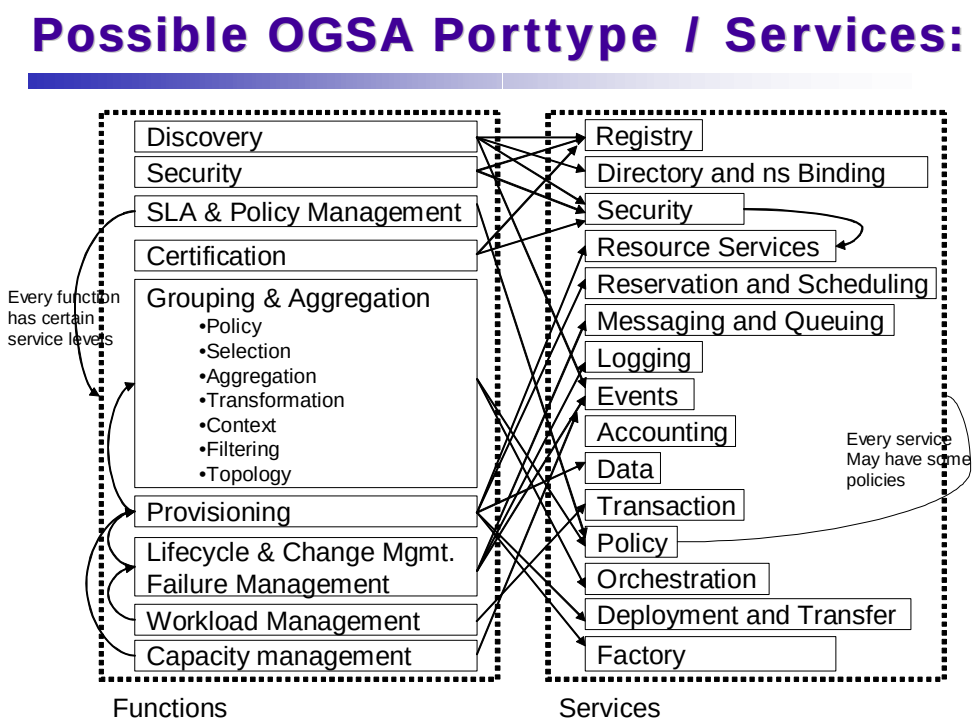

### 7.5.13 Workload Management

Taking online games as an example, the amount of workload is a direct result of how many concurrent players are being hosted on a game server. If the game server (A) is responsible for a 20 square mile area in the game world, and a battle occurred in that area, many players will rush to that area, causing workload on that server to increase. As players enter that area and leave other areas, other servers' workload will decrease. So, when the workload of server A gets above certain threshold, a load balancing routine needs to be triggered to rebalance the resources (i.e. servers). That is, redistribute workloads across servers with idle capacity.

### 7.5.14 Application Specific (e.g. multimodal input)

Additional domain specific services may be needed; for example, a voice recognition engine.

## *7.6 OGSA Service Mapping*




### 7.6.1 Security Considerations

Each consumer, service provider, developer and publisher must have its own security identity and context (e.g. relationships with other entities). All security functions traditional in the enterprise environment must be addressed including privacy and non-repudiation.

### 7.6.2 Performance Considerations

The backend server infrastructure has to be able to scale driven by increasing concurrent number of consumers and amount of content. Another aspect of scalability is the number of content pieces or game titles that will be served by a single datacenter. New titles will also require more compute, network and storage resources per player.

### 7.6.3 Situation Analysis

Several cutting-edge technologies and products already in the market attempt to solve one or more issues described above. Such attempts take a proprietary approach and have limited scope. The OGSA, however, is an open, extensible, and comprehensive architecture, which can be used to address these problems.

We are now in proof of concept phase, after which, we would like to prototype OGSA base Commercial GRID system.

## *7.7 Reference*

# 8 Service-Based Distributed Query Processing using OGSA and OGSA-DAI

## *8.1 Summary*

A service-based distributed query processor supports the evaluation of queries expressed in a declarative language over one or more existing services. These services are likely to include database services, such as those provided by the OGSA-DAI project (www.ogsa-dai.org), but may also include other computational services. As such, a service-based distributed query processor supports service orchestration, and can be seen as complementary to other infrastructures for service orchestration, such as workflow languages. In a Grid setting, distributed query processing can benefit from the facility to discover and make use of computational resources on demand, based on the anticipated resource requirements of a request. A distributed query processor on the grid can itself be cast as a service, referred to here as a Grid Distributed Query Service (GDQS).

In principle, a GDQS can be used in any Grid application that must integrate and analyse structured data collections. Regardless of the application domain, there are several primary phases in a typical use case involving GDQS. Some of those phases are transparent to the user, whereas some require interaction with the user. All, however, imply particular requirements from the grid software infrastructure. Each phase will be examined in more detail in Section 1.3; below is a summary:

- **Factory discovery and service instance creation phase.** The user has to discover a GDQS factory by querying a Grid Data Service Registry (GDSR). It is the users responsibility to have the knowledge of an appropriate registry and a reasonable search criteria. Once the factory is discovered an instance can be created.
- **Resource discovery phase**. The GDQS needs to obtain metadata about the computational capabilities of available grid nodes in order to be able to optimize and efficiently schedule a query plan. This phase is transparent to the user.
- **GDQS setup phase.** The User is required to prepare the GDQS instance for accessing multiple data sources and analysis services. This involves providing the factory handles and an appropriate configuration document for OGSA-DAI services that wrap the data sources being integrated, as well as providing the WSDL URLs of the services that are to be used for analysis. The GDQS uses this information to import the database schemas of the data sources and WSDL content of the services so that it can process (compile and optimize) the submitted query.
- **Query (request) submission phase**. The user is required to formulate a query in Object Query Language (OQL) and submit it to GDQS.
- **Query Execution and result delivery phase.** Once the query is submitted, the GDQS compiles, optimizes, schedules and executes the query utilizing the available computational resources on the grid by taking into account the information collected in the resource discovery and GDQS setup phases. The results are, then, delivered to the user subject to the interaction patterns allowed by the OGSA-DAI Grid Data Service (GDS) port type interaction semantics [ref to OGSA-DAI].

## *8.2 Customers*

The potential users of SB-DQP can be both from commercial or scientific background. The fundamental characteristics of the usage pattern is the requirement to integrate data from distributed and heterogeneous resources with analysis capabilities provided as services. For example, distributed query processing is considered a relevant technology in bioinformatics, in which there are many distributed structured data stores, and in which an individual analysis often

needs to access several of these stores and several analysis tools. In bioinformatics, there are several hundred important structured data stores (of very variable size) and many analysis tools applicable to data that can be extracted from these stores. Currently many bioinformaticians apply a sequence of disconnected (or largely manually connected) activities to achieve data and analysis integration. A declarative interface that uses a standard query language to combine such disconnected activities in an optimized way is of particular interest to the bioinformatics community.
A detailed scenario that illustrates the potential value of the GDQS for bioinformaticians is given in Section 1.3. The scenario provided illustrates the integration of data from two distributed data resources, the Gene Ontology (GO) database, the Genome Information Management System (GIMS) in combination with an analysis tool, namely BLAST.

## *8.3 Scenarios*

The following OQL query is meant to provide a starting point for constructing a scenario that illustrates how a bioinformatician can interact with a GDQS causing it to pass through the phases introduced in Section 8.1. First the query is explained and then scenarios are provided that exemplify the

```
select p.proteinId, blast(p.sequence)
from p in protein, t in proteinTerm
where t.termId='GO:0008372' and
p.proteinId=t.proteinId
```

This query returns, for each protein annotated with the GO term 'GO:0008372' (i.e., unknown cellular component), those proteins that are similar to it. Assume that (as in [21]) the protein and proteinTerm extents are retrieved from two databases, respectively: the Genome Information Management System (GIMS) [img.cs.man.ac.uk/gims] and the Gene Ontology (GO) [www.geneontology.org], each running under (separate) MySQL relational database management systems. The query also calls the BLAST sequence similarity program [www.ncbi.nlm.nih.gov/BLAST/], which, given a protein sequence, returns a set of structures containing protein IDs and similarity scores. Note that the query is essentially a select-project-join query but retrieves data from two relational databases, and invokes an external application on the join results. A service-based approach to processing this query over a distributed environment allows the optimiser to choose from multiple providers (in the safe knowledge that most heterogeneities are encapsulated behind uniform interfaces), and to spawn multiple copies of an operator to exploit parallelism. In the example query, for instance, the optimiser can choose between different GO and GIMS databases, different BLAST services, and different nodes for evaluating the query sub-plans.

### 8.3.1 Service Discovery and Instance Creation

Figure 1 illustrtaes the interaction during the first phase. The first interaction in the figure refers to the fact that a GDQS factory registers itself to a GDSRegistry as part of its initialization. The client queries a Registry using `GridService::FindServiceData` operation to find an appropriate GDQS factory (GDSF) (interaction 2). The client then creates an instance of the GDQS using the OGSA factory port-type.

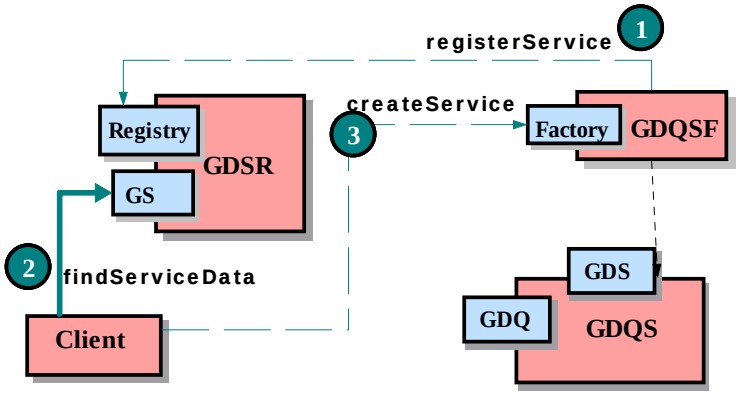



**Figure 1 Service Discovery and GDQS instance Creation**

### 8.3.2 Setting up the GDQS instance

It is necessary for the GDQS to collect database schema information of the data sources being integrated. Figure 2 illustrates interaction during the setup phase through which the GDQS acquires this information. The client discovers a GDS Factory for a particular data source (interaction 2) and passes the handle of this factory (GSH:GDSF) along with a configuration script obtained by querying the factory (interaction 3), to the GDQS instance via an importSchema call (interaction 4). It is also necessary to provide a configuration document to determine the type of the GDS being created. The client should be able to interrogate the GDS factory to find out the set of configurations supported, and choose the most convenient one. The GDQS instance, then, creates a GDS instance (GDS1) using the factory handle and the configuration document provided by the client (interaction 5), and obtains the database schema of the data source wrapped by that GDS (interaction 6).

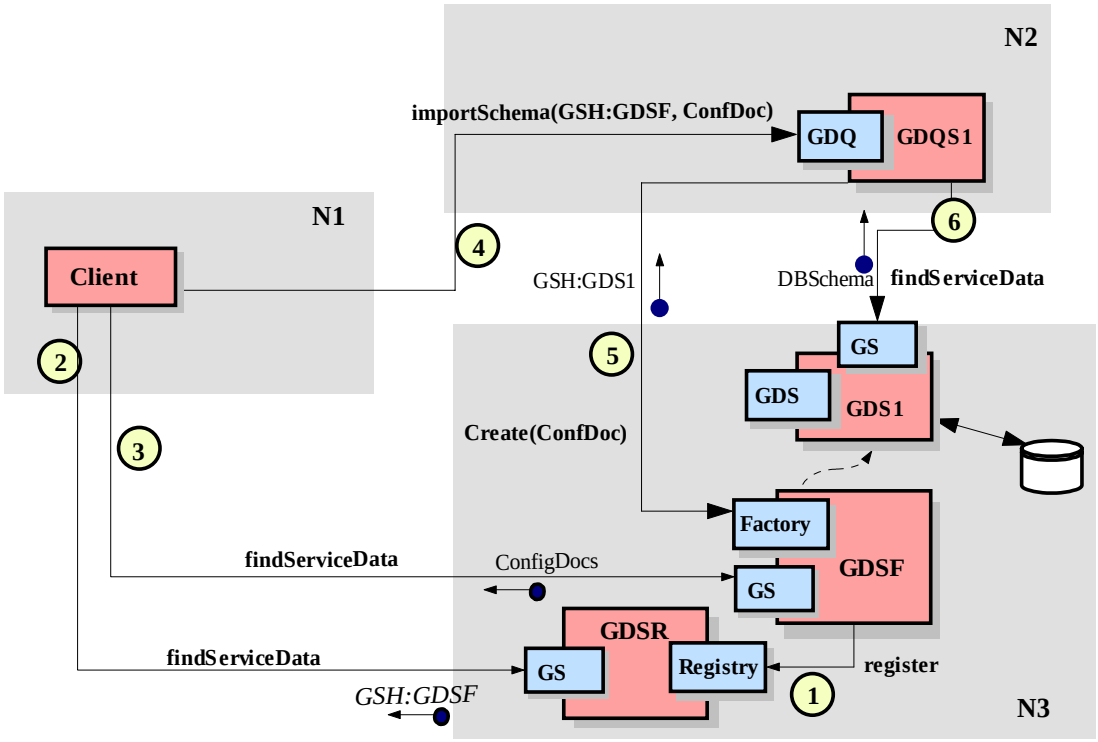



**Figure 2 Importing Schema Information of Data Sources**

### 8.3.3 Collecting Computational Resource Metadata

It is also important for the GDQS to collect sufficient data about the available computational resources on the Grid to enable the optimiser to schedule the distribution of the plan partitions as efficiently as possible.

Although the current OGSA reference implementation does not fully support this need, it does provide a high-level Index Service, to enable collecting, caching and aggregating of computational resource metadata. Figure 3 illustrates the service-based architecture that enables a GDQS to collect resource metadata from multiple nodes on the Grid. In this set-up, an index service collects dynamic information on the system it is deployed in using back-end information providers. The GDQS identifies a central index service as its server for caching and aggregating metadata, and causes (2) it to subscribe to other distributed index services. The remote index services send (3) notification messages at specified periods whose payload is resource metadata in a format determined by the back-end information provider. The GDQS can use (4) a findServiceData call to obtain the aggregated information as SDEs from its server.

Note that one would expect the index service hierarchy to have been set up as part of a virtual organisation's infrastructure, since the identification of Grid nodes that constitute the organisation's resource pool is beyond the operational scope of the GDQS.

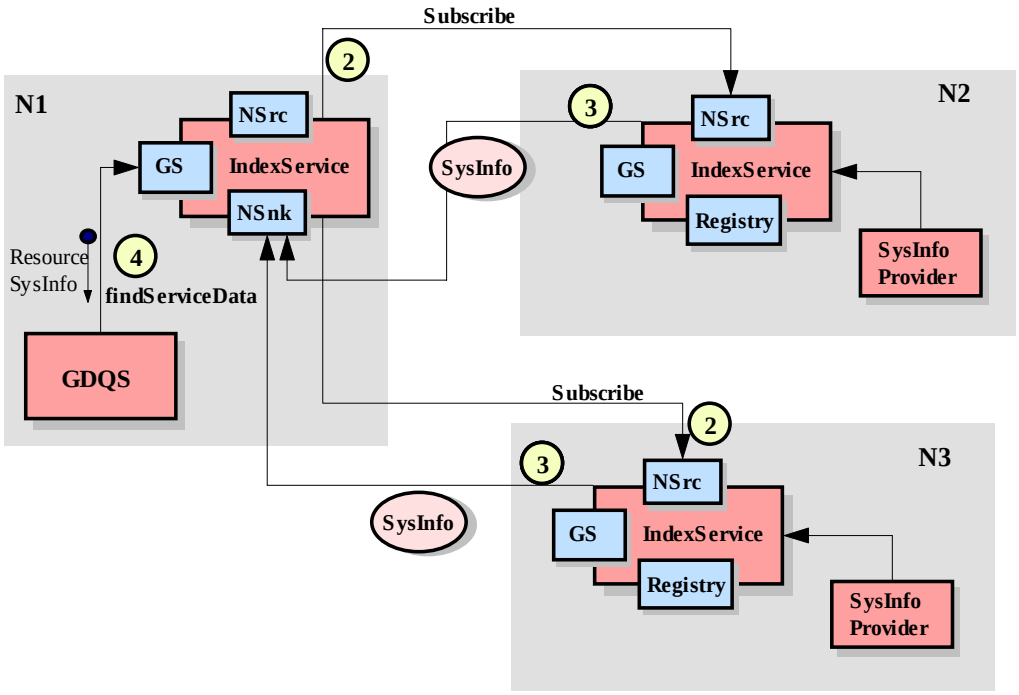



**Figure 3 Acquiring Computational Resource Metadata**

### 8.3.4 Query (Request) Submission

Most of the interactions (apart from the initial query submission) in this phase are inter-service interactions transparent to the user. Figure 4 illustrates those interactions. After importing the schemas of the participating data source, the client can submit queries (1) via the GDS port type using a perform call. Note that the format and semantics of query submission is compliant with that of OGSA-DAI framework. The submitted query is compiled and optimised into a distributed query execution plan. The GDQS, then creates a set of Grid Query Evaluator Services (GQES) for executing each query-sub plan ( or partition) generated by the query optimizer on a different node on the grid. The scheduling of the GQES instances is also done in an optimized way based on the metadata collected. Once the GQES instances are created on their designated execution nodes (and these could be, potentially, anywhere in the Grid), the GDQS hands over to each (2) the plan partition assigned to it. This is what allows the DQP framework to benefit from (implicitly) parallel evaluation even as the uniform service-based interfaces hide most of the low-level complexity necessary to achieve this. Finally, (some of the) GQES instances interact (3) with other GDS instances to obtain data, after which the results start to propagate (4) across GQES instances and, eventually, back to the client via the GDT port type.

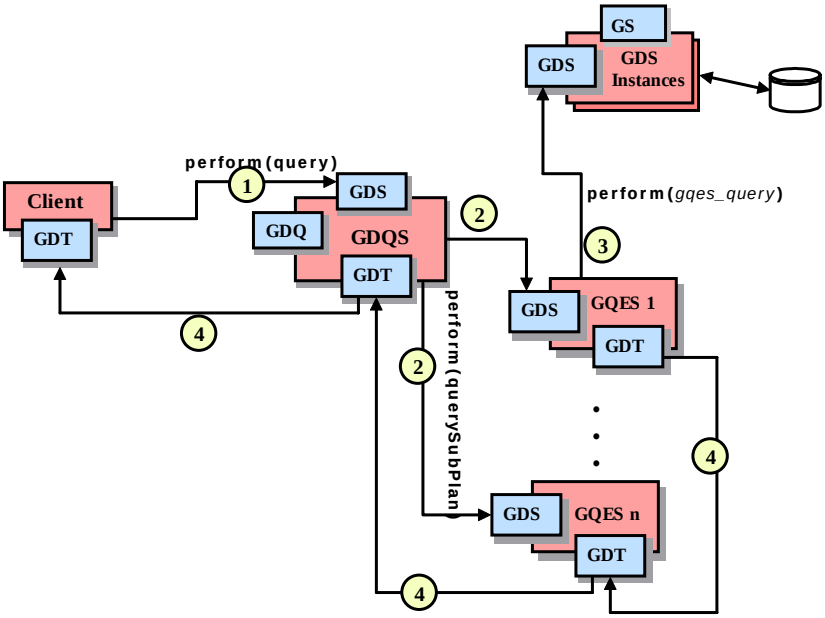



**Figure 4 Query Execution - Overview**

### 8.3.5 Query Execution and Result Delivery

For the example query given at the beginning of Section 8.3, the query submission gives rise to the Grid Service (GS) interaction diagram in Figure 5. The GQESs that scan stores, viz., N1 and N2, are instantiated in different hosts. Conditions at N2 (e.g., available memory) are such as to justify the GDQS having assigned the hash join to N2. For the BLAST operation call, the GDQS saw benefits in parallelising it over two GQESs N3 and N4. The GDQS receives the request (1) and compiles it into the distributed query plan in Figure 7(d), each partition of which is assigned to one or more execution nodes. Each execution node corresponds to a GQES instance which is created by the GDQS (2). The GDQS then dispatches (3), as an XML document, each plan partition to its designated GQES instance. Upon receiving its plan partitions, each GQES instance initiates its evaluation. Query execution is a data flow computation using the iterator model, in which each operator implements an fopen(), next(), close() interface. Data flows from the GQES instances that execute partitions containing operators whose semantics requires access to stores.

Within each GQES instance, the initialisation procedure starts when an open() call reaches the topmost operator. This call propagates down the operator tree from parent to children at every level until it reaches the leaf operators. Then, interaction with other GDSs occurs. The handle for each such GDS will have been planted by the GDQS in the XML document passed to each GQES instance that needs it. For example, in node N2 (in Figure 5), when the stream of open() calls reaches the sequential scan operator, it causes the N2 GQES to interact with the GDS instance on N2, whereby data becomes ready to flow upwards from the protein extent in the GDS through which the GIMS database is accessed.

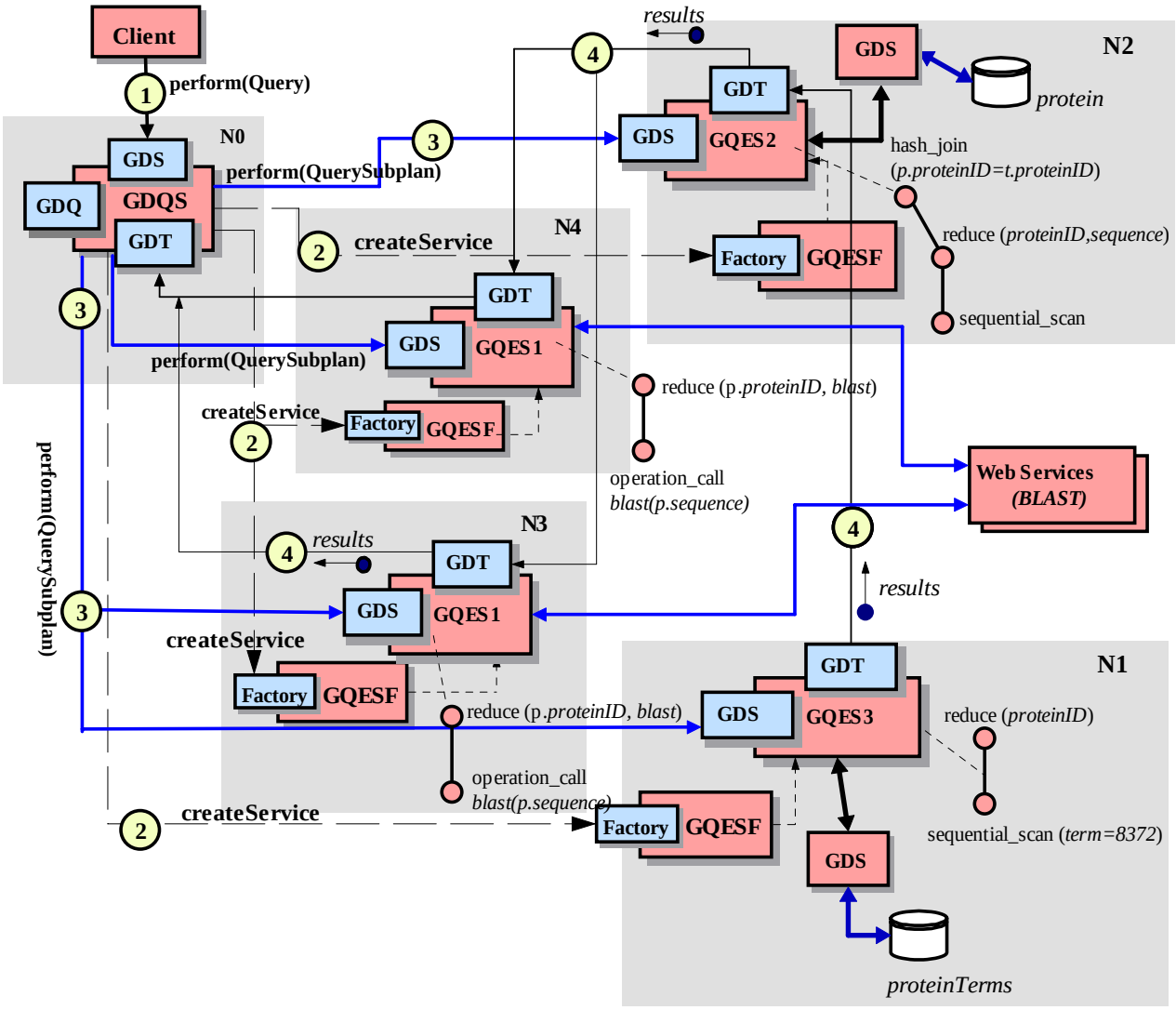



**Figure 5 Query Execution and Result Delivery - Detailed**

Note that many forms of disruptive heterogeneity in the data stores are encapsulated by the standard GDS interface. As such, SB-DQP exploits the power that the Grid metaphor embodies, viz., query evaluation is carried out over heterogeneous data and computational resources but the heterogeneity is encapsulated behind the universal GS interface, giving rise to consistent and uniform inter-service interaction semantics.

## 8.4 Involved resources

A GDQS can be expected to make use of computational resources for: (i) running query evaluator services, several of which may collaborate in the evaluation of a single query; (ii) moving data from primary sources to analysis tools or to evaluators that join or manipulate the data in a query; and (iii) holding intermediate results for performance or reliability. All such computational services need to be identified and allocated dynamically to support the specific needs of complex requests.

In terms of the services used by a GDQS, these are likely to include: (i) service registries, as service descriptions must be imported into a GDQS before queries are evaluated over them; (ii) structured data access services, as consistent access to structured stores is important for reducing set-up costs; and (iii) flexible transport services, for example supporting streaming of data and delivery to multiple sites in parallel.

### *8.5 Functional requirements for OGSA platform*

- *Discovery and Brokering.* It is very important for SB-DQP to be able to discover available computational resources, Grid Data Services (GDS) and Analysis Services (AS). The discovery of the GDSs is needed for importing the database schemas of the data sources over which a query is to be formulated. Discovery of analyses services is needed to identify the type of operations and data types supported/required by those operations, so that they can be embedded in a query. The crucial requirement here is a uniform model that will enable both the SB-DQP clients (users) and the DQP service itself to discover and interpret the metadata about such services but also to relate them to the information about computational resources (hosting environments, machine capabilities such as CPU speed, available memory etc.).

- *Metering and accounting.* SB-DQP can potentially use many GDSs and other grid and web services. Each of these may have its own impact on the overall billing cost of the distributed query service. SB-DQP must be able to integrate into metering, accounting and billing mechanisms employed by other participating data sources and/or services and if possible choose from among the most convenient ones based on user preferences. This is only possible if such seamless integration is supported at the infrastructure level.

- *Data sharing and management.* Data sharing and management is fundamental to SB-DQP. It does this at two levels. At the lower level it relies on Grid Data Services for accessing data sources, and at a higher level it processes the data it obtains (joins, reduces, analyses etc) in a way that conforms to the principles of a data-flow architecture. It does not however, currently, address the problem of schema integration and consistency. SB-DQP would benefit from such data management facilities as semantic data model integration, transparent data caching and consistency management.

- *Monitoring.* SB-DQP requires monitoring in several contexts. First, it should monitor the progress of the services it orchestrates. Progress information has to be collected from the Evalutor services (GQESs), GDSs and analysis services. Second, since a query can potentially involve long running interactions (because of large amounts of data or network conditions) the SB-DQP should respond by re-allocating resources and re-scheduling evaluator services. This, in turn, requires monitoring of computational resources to collect dynamic information to aid in reaching a decision as to how to adapt to the changing conditions.

- *Multiple security infrastructures.* In most of the cases the distributed query will require access to multiple data resources access to which may be restricted by different security policies and infrastructures. It is essential for the SB-DQP to rely on infrastructure support for obtaining access permission to multiple resources on behalf of the client in a transparent way.

- *Optimization of resource usage.* SB-DQP uses a query optimizer (the Polar* system) which is responsible for generating an efficient execution plan for a declarative OQL query over distributed services (both data and computational, since OQL supports invocation of external functions). As such, SB-DQP offers system-supported optimization of declarative requests with implicit parallelism. In that respect

- *Transport Management.* As the SB-DQP executes the queries as a data flow computation, efficient data transport is of paramount importance. Shipping only XML data over SOAP/HTTP is not particularly convenient for data intensive applications. It is very desirable to have multiple transport protocols, including very efficient ones, to be available for inter-service interactions.
- *Fault tolerance and disaster recovery.* Fault tolerance is particularly important for long running queries that can potentially return large amounts of data.

### 8.6 OGSA platform services utilization

- *Name resolution and discovery.* The discovery of Grid Services via an easy to use interface that enable rich queries to be submitted against metadata maintained in the registries, is important for the usability of the SB-DQP. The setup of SB-DQP requires the discovery of Grid Data Service Factories for importing the schemas of the participating data sources.
- *Service domains.* SB-DQP can be seen as a good example of service domains. It coordinates and orchestrates multiple Grid Query Evaluator Services and other Web services in a particular context during its lifetime.
- *Messaging and events.* There may be several contexts where SB-DQP needs to be notified of events. If the schemas of the participating data sources change the DQP would want to know about those changes so that the queries can be validated against the new database schemas. Another context is progress monitoring. When the query execution is in progress, the SB-DQP needs to receive notification messages that indicate the state of the execution at each query evaluation node. It is also required to receive regular updates on the state and availability of the computational resources, so that the query evaluation can be re-scheduled if needed.
- *Transaction.* Currently distributed transactions are not supported in SB-DQP, but it would certainly benefit from transaction interfaces provided by the infrastructure in the future.
- *Service orchestration.* SB-DQP implements a service orchestration framework in two sense: both in terms of the way its internal architecture handles the construction and execution of distributed query plans and in terms of being able to query over data and analysis resources made available as services. The latter form of service orchestration can be seen as complementary to other infrastructures, such as workflow languages.

### 8.7 Security considerations

The nature of the security challenges facing a GDQS are likely to vary from setting to setting, but may be quite demanding. For example, a single query may run over services within different domains of control, and could benefit from allocating evaluators to run on nodes that are under different domains of control. There may also be privacy issues on the data being manipulated by a query – for example, a requester may be reluctant ever to allow data from a private source to leave their organisation, but may want to join that data with data from a public source. Thus single-enterprise, multi-enterprise and all-comers scenarios are all possible.

### 8.8 Performance considerations

There are many aspects to the performance of a distributed query. As queries are declarative, their execution must be planned. Query planning needs access to comprehensive information on the costs of using the services of relevance to a query, and also requires information on the computational resources available for evaluating a query.
Different operations in a query plan may prefer different forms of transport. For example, many distributed query processors support pipelined parallelism, but some operations are blocking, and

thus may be more suited to bulk data delivery. Which operators should be used to evaluate a portion of a query will depend on the capabilities and load of the computational resources available. Parallelism can often be exploited to improve the performance of query evaluation, but scheduling is clearly challenging in an open environment such as the Grid.

### *8.9 Use case situation analysis*

As stated in Section 8.1 each phase in the use case has implication on the required services/functionalities from he underlying infrastructure. The following list is an attempt to identify those requirements for each phase and to what extent they are met by the current OGSA reference implementation.

**Service discovery and instance creation.** The primary requirement here is the ability to discover the GDQS Factory and GDS Factories for the data sources by submitting a query to the service registries. This requires the service registries to support both the ability to specify and publish potentially rich information on the services being registered, and the ability to query this rich information using a well-known (easy to use) query language.

The existing OGSA reference implementation does not sufficiently support the ability to query against the service descriptions. The idea of Service Groups proposed in the latest draft of the GS specification provides more complete support in this regard.

**Setting up the GDQS instance**. One important requirement here is that the Grid Data Services must provide the schema of the database they wrap in a well-defined way. In other words the GDQS must be able to query the GDS instances to obtain the schema of a particular data source. Service Data Elements are one obvious candidate to provide such information in a well-defined way. Currently, querying this information via SDEs is not supported. GDQS obtains the database schemas by a custom extension to OGSA-DAI framework. The requirement referred to here, however, is more directly relevant to OGSA-DAI project rather than OGSA.

**Collecting computational resource metadata.** The relevant OGSA service here is the Index Service which is not part of the core OGSI but is provided as a higher level service. Although the Index Service seems to offer a flexible approach to collecting grid resource metadata, there are some issues that remains unresolved. The SB-DQP requires several classes of metadata to be interrelated and provided in a coherent way. The classes of metadata required are:

- The capability of a grid node (a machine that offers its computational resources to the grid user community) in terms of the CPU power, available memory, available disk space etc.
- Dynamic (real-time) information on the communication load on network connection between a set of grid nodes.
- The characteristics of a grid node in terms of the services it hosts. For example the information as to whether a particular grid node hosts a Grid Data Service Factory or a Grid Query Evaluation Factory.

Currently there is no a coherent way of collecting and relating such classes of metadata.

**Query (request) submission.** The implication of a query request in regard with the use of infrastructure services is that the GDQS has to dynamically create instances of GQESs on an arbitrary number of grid nodes to execute the sub-queries. Currently it is only possible the create a grid service instance on a node if its factory is already deployed on that particular node. This constraints the query optimizer to consider only a limited set of grid nodes (only those where a GQES factor exists). It is desirable to have the ability to dynamically ship the factory code to a hosting environment and deploy it so that any grid node can be considered for scheduling GQES instances.

**Query execution and result delivery.** The primary requirement here is being able to bind to efficient transport mechanisms. Currently only XML over SOAP/HTTP is seamlessly supported. The Reliable File Transfer Service that provides access to Globus Grid FTP APIs does not seem to be seamlessly integrating with the service interfaces. What is needed is direct support for efficient data transfer at the inter-service interaction level.

### *8.10 References*

1. M.N. Alpdemir, A. Mukherjee, N.W. Paton, P. Watson, A.A.A. Fernandes, J. Smith, T. Gounaris, Grid Distributed Query Service (GDQS) Design, OGSA-DAI Design Document , 2, December, 2002.
2. J. Smith, A. Gounaris, P. Watson, N.W. Paton, A.A.A. Fernandes, and R. Sakellariou. Distributed query processing on the grid. Proc. 3rd Int. Workshop on Grid Computing, J.Sterbenz, O.Takada, C.Tschudin, B.Plattner (eds.), Springer-Verlag, 279-290, 2002.

# 9 Grid Workflow

## *9.1 Summary*

Workflow is drawing attention as a convenient way of making new services by connecting existing services (like shell-scripts of UNIX systems). A new Grid service can be created and used by registering a workflow definition to a workflow engine. The definition is interpreted by the workflow engine, and calls several other Grid services as is specified in the definition.

## *9.2 Customers*

Workflow will be used by both users and providers of Grid services. The cases when workflow will be used are as follows:

1. Connection of simple services: Users (or service providers) make a new Grid service by connecting several simple services (whose execution time is relatively short). For example, by connecting a stock information service and a currency exchange rate information service, a foreign stock information service can be made.
2. Job workflow: Users (or service providers) combine several jobs, specifying their execution order, input, output, etc. Here, jobs include both scientific and commercial jobs. For a scientific job example, simulation service and visualisation service is connected using workflow. (Of course there are many other examples like compound simulation, data grid, etc.) Scientific job workflow may require huge amount of data transfer between services. As for commercial jobs, an example would be summing up sales result at each branch shop in parallel, and then collecting them at the head office.
3. Description of business process: Service providers describe business processes by connecting several services. For example, a travel agency connects a flight ticket reservation service, a hotel reservation service, and a vehicle reservation service to make a new travel reservation service. This kind of workflow is well investigated in the area of Web Services. Business process may take a long time (ex. one month) to finish, and may need exception handling mechanism (ex. cancellation of reservation).
4. System administration: Service providers describe a service for system administration using workflow. For example, a system administration workflow obtains an application program from an application repository using a file transfer service and deploys it to a Grid service container.

Combination of above examples is also possible. For example, one can think of a workflow which obtains weather information from various place of a country (above example: 1), and executes weather simulation job using the information and visualizes the result (above example: 2).

In addition, everything is abstracted as Grid service in OGSA. Therefore, everything which is abstracted as Grid service can be dealt with workflow.

A workflow definition itself should be seen as a Grid Service. Thus, workflow should comply with the various rules which the Grid Service Specification requires. For example, a workflow definition should have FindServiceData operation in GridService PortType, and may need to support Notification; and a workflow instance should be created by a Factory.

## *9.3 Scenarios*

As described above, workflow is used in various cases. Here, I will describe “application deployment scenario” in which typical relationship between other services/functions is shown.

### 9.3.1 Application deployment scenario

In this scenario, we assume that a system administrator or a user of a Grid system wants to deploy (install) an application to a Grid container.
The process is executed by a **service orchestration engine**. In the service orchestration, firstly, an application program is obtained from an application repository which may be implemented as **shared storage**. The storage may be found using a **discovery service**. If the storage has functionality of **data cache / replication**, the program code can be efficiently obtained.
When connecting to the storage, **authentication / authorization** should be performed in order to restrict the access to the program. For authentication and authorization, a **policy management service** may be needed to get security policy for deciding if providing the program is allowed or not.
After obtaining the program, it is deployed using a deployment service which may be a part of an **administration service**. Here, **authentication / authorization** should be performed again. It may be needed to reserve the resource (the Grid container) beforehand using a **reservation service**.
All these processes might need to be logged using a **logging service**, and the log information might be passed to an **accounting service** for accounting. Again, for logging and accounting, a **policy management service** may be needed to obtain policies for them.

## *9.4 Involved resources*

Computational resources are required in order to interpret and execute workflow descriptions. For managing long-lived workflow, non-volatile memories like files or databases are needed.

## *9.5 Functional requirements for OGSA platform*

In the scenario described above, following functionalities are required.

1. **Workflow**

   With this functionality, several services are connected to realize application deployment. This functionality is represented as “Flow” in [1].

2. **Discovery**

   In the above scenario, service discovery functionality is needed to discover storage service which contains the application program to deploy. This functionality is represented as “Discovery and brokering” in [1].

3. **Shared storage**

   In the above scenario, shared storage is used as an application repository. This functionality is represented as “Data sharing” in [1].

4. **Authentication and authorization**

   Obtaining application programs and deploying them into a Grid system may require authentication / authorization. This functionality is described in “Multiple security infrastructures” and “perimeter security solutions” in [1].

5. **Application deployment**

   This functionality is required to deploy an application to a Grid container. This functionality is included in “Administration” functionality in [1].

6. **Advanced reservation**

This functionality may be required to execute the application on reserved resources. This functionality is described in "provisioning" functionality in [1].

7. **Logging and accounting**

   Processes like obtaining / deploying application programs might be logged, and the information might be used for accounting. This functionality is represented as "metering and accounting" in [1].

8. **Policy**

   Authentication, authorization, metering, and accounting may require policies.

### *9.6 OGSA platform services utilization*

1. **Service orchestration service**

   This service corresponds to "workflow" functionality, and is used as "workflow engine".

2. **Name resolution and discovery service**

   This service corresponds to "discovery" functionality.

3. **Security service**

   This service corresponds to "authentication and authorization" functionalities.

   In some cases, security is not implemented as services but functions attached to each service. However, some of the security functions such as decision of authorization may be implemented as services.

4. **Data management service**

   This service corresponds to "shared storage" functionality.

5. **Administration service**

   This service includes "application deployment" functionality.

6. **Provisioning and resource management service**

   This service includes "advanced reservation" functionality.

7. **Metering and accounting**

   This service corresponds to "logging and accounting" functionality.

8. **Policy service**

   This service corresponds to "policy" functionality.

### *9.7 Security considerations*

There may be a need to deny access to workflow definitions from non-registered users. To implement this, authentication and authorization should be performed when creating a workflow instance using a Factory, and when accessing a workflow instance.
In addition, services called from workflow may require authentication and authorization. To support this, delegation mechanism like GSI may be needed.

### *9.8 Performance considerations*

If execution time of a service called from a workflow is long enough, performance of a workflow engine does not matter much. However, if it is short, performance of a workflow engine may be important.

In addition, if there is need to transfer large amount of data between services called from a workflow definition, it is not efficient for a workflow engine to receive and send the data. Therefore, it may be needed to allow description of direct data transfer between services [7].

### 9.9 Usecase situation analysis

Many important works have been done in the field of Web Services. For example, there are WSFL[2] by IBM, XLANG[3] by Microsoft, BPEL4WS[4] derived from both of them, WSCI[5] by SUN, WSCL[6] by HP. In the Grid computing field, GSFL[7] was proposed by ANL. In addition, WfMC (The Workflow Management Coalition) is working in this field for a long time. These significant works can be a basis of a workflow specification of OGSA.

### 9.10 References

1. Foster, I and Gannon, D. The Open Grid Services Architecture Platform, 2003. http://www-unix.gridforum.org/mail_archive/ogsa-wg/doc00016.doc
2. Web Service Flow Language (WSFL 1.0), May 2001, http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf
3. XLANG Web Services for Business Process Design, 2001, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm
4. Business Process Execution Language for Web Services, Version 1.0, July 2002, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
5. Web Service Choreography Interface (WSCI) 1.0 Specification, 2002, http://wwws.sun.com/software/xml/developers/wsci/
6. Web Services Conversation Language (WSCL) 1.0, March 2002, http://www.w3.org/TR/wscl10/
7. GSFL: A Workflow Framework for Grid Services, July 2002, http://www-unix.globus.org/cog/projects/workflow/

# 10 Grid Resource Reseller

## 10.1 Summary

It is not always desirable for owners of Grid resources to interface with end users directly. Inserting a supply chain between the resource owners and end users will allow the resource owner to concentrate on their core competence (e.g. in maintaining large supercomputers) and avoid providing costly interaction and support to a large number of consumers, allowing them instead to deal with a few large customers (potentially only one) who resell the resources.

End users can purchase resources bundled into attractive packages by the reseller (aggregation); these resources might in fact come from several resource owners.

The resellers can make money from reselling aggregated computational resources without having to own any resources themselves, thereby minimizing their own risk. In general, the reseller maintains resource provision by sustaining their relationships with upstream providers. However, to protect the agreed service level with the end users, the reseller may occasionally find it necessary to switch provider, either temporarily or permanently. Instead of worrying about maintaining resources, the reseller can focus on providing good customer care as well as marketing resource bundles to their target market(s).

This use case is adapted from the "Computational Reseller" use case, which was written by Jon MacLaren and William Lee, and appears in the GESA Use Cases Document [1].

## 10.2 Actors

There are three key actors in the Grid Resource Reseller scenario. The first of these is the **"Resource Owner"** of which there may be several in this scenario (which is considered from the point of view of the reseller). The Resource Owner is imagined to own resources which are expensive and rare, e.g. a supercomputer, although this does not have to be the case. These owners want to sell resources on in bulk, dealing with only a few large customers, who are resellers. They are interested in ensuring that they sell all their resource, although they are less concerned about the actual usage of the resource – that is the concern of the resellers, who are their customers. There will, however, be service level agreements between the resource owner and the resellers.

Next, there is the central actor, the **"Resource Reseller"**. The reseller acts as both customer (of resource owners, or upstream providers), and provider (to end users or downstream providers). The reseller need not be interested only in resource utilization, as their primary concern will be making a profit, i.e. if they can get all their customers to buy pre-paid resource usage packages (like "free minutes" on mobile phones), they do not care if these are ever used. In fact, a certain amount of overselling may be possible, i.e. if everyone used all their pre-paid resources at once, the reseller would be in trouble, but in fact this is extremely unlikely. A reseller will have service level agreements both with the providers and consumers of the resources. The reseller will have many more consumers than providers (e.g. an order of magnitude more), providing a natural fan-out as the supply chain moves from the resource owner to the end users.

Finally, there are the **"End Users"**, who are customers of a Resource Reseller. They are the real consumers of the resources. They do not know who owns the resources they use, as they get all their resources, plus associated service and support, from the reseller. They will be free to select a reseller who is suitable for them, e.g. based on the packages the reseller offers, and their cost.

Naturally, the resource owner, reseller(s) and end users will be part of different organizations, and may be geographically distributed.

In the scenario presented below, we only consider there to be a single reseller between several resource owners, and many end users. However, in considering the requirements for this scenario, it is important to envisage the possibility of a chain of resellers (as is the case for internet providers today).

## *10.3 Scenarios*

As this use case is extremely general, there are many possible scenarios. Further, these examples are all similar, differing only in the details. Therefore, only one example is provided.

### 10.3.1 Computational Chemistry Reseller

Consider the example of a reseller who has strong links with the chemical industry and the expertise to support a wide range of chemistry applications running on supercomputers. To establish their business, they offer supercomputer owners the chance to sell resource in bulk to them, on the understanding that they will resell the resource. The reseller agrees to respect the policies of the resource owners when reselling. One resource owner provides cycles which are only for use by academic users; another offers a two-tier price structure, where cycles that are sold on to non-academic users are charged for at a higher tariff. Two resource owners specifies that the provided cycles must not be sold on to another reseller. Due to this complication, the reseller decides only to deal with end users.

As well as sourcing supercomputer centers, the reseller wants to provide access to all the popular chemistry packages. In some cases, the reseller can lease the licenses from the resource owners, some of whom have installed a subset of the target software. However, the reseller also sources some of these packages directly from the manufacturer, and must arrange for the staging (or installation) of the software on the target machines.

Finally, the reseller engages in a publicity campaign to attract users to it's services. They market monthly packages of resources which includes pre-paid ("free") items such as CPU cycles, secure and backed-up disk storage, and software licenses. To make itself as attractive as possible, the reseller deliberately resells the resources at a loss for the first three months of operation as a one-off "not to be repeated" offer (loss-leading).

To facilitate the execution of the user's work, the reseller provides a resource broker. Users submit their work to the broker, which matches the users preference with the policies of the resource owners. Based on this matching, plus information about the state of the resources themselves, the user's job is dispatched.

Where the reseller's service level agreement with the end user is "broken", the user may be entitled to some compensation. This may be described as part of the service level agreement itself.

It is useful to summarize the potential advantages of this scenario from the perspective of each type of actor:

1. The **Resource Owner**. There are a number of reasons why a supercomputing centre might wish to sell its cycles to a reseller.
    a) If all cycles are sold this way, the resource owner never needs to deal directly with large number of customers; this is useful as it is costly to maintain high quality of customer care. This policy enables them to manage their resources in a small number of large transactions.
    b) During a period of low local usage, a centre might want to make a one-off sale of a large amount of otherwise redundant cycles.

c) A centre with seasonal peaks and troughs in local user usage might want to sell an amount of cycles (varying per month) to match expectation, thus maintaining steady usage.

2. The **Resource Reseller**. The reseller bundles the resources available to it from the various upstream providers, including some licenses it can obtain from the software vendors at a reduced rate (as it deals mainly with academics and in large quantity). An example offer is that for a reasonable monthly fee, the chemist gets 200 "free" CPU-hours on a Cray T3E, plus thirty uses of Guassian98 thrown in (exceed that, and he gets charged quite a lot, of course.) They also include some compensation deal when jobs are not delivered due to downtime (a kind of insurance). A Reseller who has insights in the market trend can predict future demand and source resource provision from upstream vendors in advance when the price is attractive.
3. The **End User**. The chemist wants to get resources from the reseller because getting bundled resources reduces transaction costs in dealing with all parties manually. Also, he would expect to have better customer care and risks are shared with the reseller if upstream vendors default. Finally, the academic might be able to get his bundle for less because he gets it from the same reseller he gets his electricity / mobile phone time from. It encourages companies with existing micro-transaction technology (such as telecom, utility, etc.) to participate as resellers.

### *10.4 Involved resources*

The **Resource Owner** is selling resources to one or more **Resource Resellers** (see also the GESA-WG Computational Provider Scenario [1]).

Each **Resource Reseller** in the supply chain is buying resources from one or more **Resource Owners** and upstream **Resource Resellers**. The reseller may bundle these resources before selling them to **End Users** or to downstream **Resource Resellers.**

The **End Users** buy (possibly) bundled resources from the **Resource Resellers**.

Ultimately, it is the resources bought from the providers that are being consumed by the end users. This could potentially be any Grid resource. These resources could be geographically distributed, and could belong to a number of resource owners.

### *10.5 Functional requirements for OGSA platform*

The presented scenario has many requirements, however, here we have chosen to describe those functions specific to the activity of reselling, i.e. we ignore generic requirements for work scheduling and execution which will arise from other use-cases. I've stuck with the headings from Section 3.2 of the OGSA Platform document [2].

The following functions are required for reselling:

**1. Discovery and Brokering**

In the scenario, each reseller operates a broker to dispatch the user's work to the available resources. The most important requirement here is that the broker can perform some sort of matching between the users' preferences, and the resource owners' policies (perhaps something like the Condor ClassAd scheme [3]). Using the evaluated list of possibilities, the broker then uses information like acceptable turnaround time and cost to select specific resources for the work.

A reseller must be able to discover resource owners (or downstream resellers), and end-users must be able to identify resellers. Service Level Agreements must be agreed between

these pairs of entities. However, in our scenario, these are infrequent (even one-off) activities, and will be achievable through existing mechanisms such as networking, advertising, etc.

**2. Metering and Accounting**

The model for accounting and charging in the scenario is quite sophisticated. The Resource Owner will sell large amounts of cycles to one or more resellers. The price for these cycles will be negotiated between the two parties; it is unlikely to be uniform for multiple resellers. Further, whether the cycles are used or not is not really the concern of the resource owner; some partial refund for unused cycles may be arranged between the two parties. In the situation of overuse, the resource owner would want to limit the amount of cycles that the reseller could use. Whether the resource owner would refuse any overrun, or whether overrun would be charged for at a far-higher rate, would be down to policy.

For the reseller, they must do their utmost to sell sufficient packages of resources to cover their expenditure, plus running costs, plus some profit margin. It should be possible for users to sign up for some sort of monthly plan, on-line, without human intervention. The reseller will need to bill the end users on the basis of usage, which is covered by existing plans in OGSA Platform. It is worth noting again that if the reseller obtains most of their money through contracts for pre-paid resource use, that they can oversell their resources (like hotel and airplane overbooking) to maximize income. Like the resource owner, their income need not depend on the actual usage of the resources.

In terms of charging, different granularities of trading must be supported. This also implies the ability to use different payment options, such as purchase order/invoicing or Credit Card, etc.

There are several different charging schemes mentioned above. However, all the models described should be possible within OGSA Platform. Similarly, it should be possible for the accounting systems to operate autonomously for the vast majority of circumstances (include underrun and overrun). While the systems being designed in the GESA Working Group [4] have cases like these in mind, it is hard to see how this functionality can be covered by the charging systems proposed in the OGSA Platform document [2] (see Section 5.9 in particular); these seem to focus mainly on tariff-based charging, based on “accounting schemas”, and do not contain the concept of reselling.

**3. Monitoring**

The Resource Owner must be able to track the usage by the clients of the various resellers to check for resources being overused.

**4. Policy**

End Users and Resource Owners will have potentially complicated policies, as may the resellers. A reseller must not be able to sell on a resource in a way that violates the Resource Owner’s policy, e.g. selling cycles to an industrial user at an academic rate. Similarly, a reseller should not be able to run a user’s work on resources which violate their policy, e.g. running a job from a user with an “environmentally friendly only” policy on a computer owned by a corporation frequently responsible for pollution, etc.

There must be some way in which to aggregate the policies of all upstream providers.

**5. Extended Service Level Agreements**

This is not a heading in OGSA Platform, but it’s something needed in this scenario, and other GESA-WG use cases [1]. We want to incorporate cost information into the SLAs

between parties. In certain circumstances, we would also like it to be possible to define rates of compensation in the SLA, e.g. if the user can't access their pre-paid resources for 24 hours or more in a month, they will be refunded £2, etc. This is the subject of ongoing work within the GESA-WG group [4].

### 10.6 OGSA platform interfaces utilization

The following interfaces (or services?) are necessary to provide functions in the previous section.

1. **Policy** [17]

   The scenario described here has sophisticated requirements for policy definition and handling within OGSA Platform. In particular, we have a need to aggregate several policies within a supply chain.

2. **Metering and accounting**

   This interface will need to be made more flexible if it is to cope with the requirements of the scenario described in this document.

3. **Provisioning and resource management**

   Required for SLA agreement and monitoring. Will need to be able to handle the extended SLAs discussed in the previous section.

4. **Brokering** [18]

   Brokering functionality is required. The policy matching aspects of this are probably to be handled by the Policy interface.

5. **Monitoring service** [19]

   This service is used for monitor function.

### 10.7 Security considerations

The Resource Owner and Reseller chain should be able to provide the user with assurances on privacy, where this is required.

### 10.8 Performance considerations

Where the reseller chain is a few steps long, it should still be possible for the user to get good performance when accessing the resources.

### 10.9 Usecase situation analysis

We do not believe that there are any examples of this use case in the Grid. (Although Application Service Providers exist, these also own the computational resources used to process the work, and so do not qualify as Resellers.) Of course, there are hundreds of examples in other areas, most notably internet provision and mobile phone provision. We are confident that once the enabling technology is present, that reseller businesses will be established.

### 10.10 References

1. Keahey, K., MacLaren, J. and Newhouse, S. (Eds.), "GESA Use Cases", February 2003. http://www.doc.ic.ac.uk/~sjn5/GGF/draft-ggf-gesa-use-cases-01-7.pdf

[17] The explanation of the policy interface in [2] is very vague and is not clear what it is.
[18] This function should be added to the OGSA platform interfaces.
[19] This function should be added to OGSA platform service.

2. Foster, I. and Gannon, D. (Eds.), "The Open Grid Services Architecture Platform", February 2003. http://www-unix.gridforum.org/mail_archive/ogsa-wg/doc00016.doc
3. Raman, R., Livny, M. and Solomon, M. "Matchmaking: Distributed Resource Management for High Throughput Computing", *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, July 28-31, 1998, Chicago, IL.
4. Grid Economic Services Architecture Working Group (GESA-WG). Home page: http://www.ggf.org/3_SRM/gesa.htm

## 11 Security Considerations

Each use case has its own security considerations and they are described in corresponding subsection.

## 12 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

Dennis Gannon
Indiana University
Bloomington, IN 47405
Phone: 812-855-5184
Email: gannon@cs.indiana.edu

Hiro Kishimoto
Grid Computing & Bioinformatics Laboratory
Fujitsu Laboratories Limited
Kawasaki, Japan 211-8588
Phone: +81-44-754-2628
Email: hiro.kishimoto@jp.fujitsu.com

## 13 Contributors

We gratefully acknowledge the contributions made to this document by Nedim Alpdemir, Takuya Araki, Boas Betzler, Kate Keahey, William Lee, Tan Lu, Norman Paton , Jon MacLaren, Andreas Savva, Charles Severance, David Snelling, and Ravi Subramaniam.

## 14 Acknowledgements

This work was supported in part by IBM and by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38 and DE-AC03-76SF0098; by the National Science Foundation; and by the NASA Information Power Grid project.

## References

Each chapter has reference section.