Specific Targeted Research or Innovation Project  
2.3.2.8 Grid-based Systems for Complex Problems Solving

**D51: Implemented Adaptation of a Grid Resource Manager for Data Mining Tasks**

Due date of deliverable: M16 (31 December 2005)  
Actual deliverable submission date: 5 February 2006

**Start date of project:** 1 September 2004  
**Duration:** 24 months

Technion – Israel Institute of Technology (TECH)

<table>
<thead>
<tr>
<th>Dissemination Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PU</td>
<td>Public</td>
</tr>
<tr>
<td>PP</td>
<td>Restricted to other programme participants (including the Commission Services)</td>
</tr>
<tr>
<td>RE</td>
<td>Restricted to a group specified by the Consortium (including the Commission Services) X</td>
</tr>
<tr>
<td>CO</td>
<td>Confidential, only for members of the Consortium (including the Commission Services)</td>
</tr>
</tbody>
</table>
Deliverable D51: Implemented Adaptation of a Grid Resource Manager for Data Mining Tasks
DATA MINING TOOLS AND SERVICES FOR GRID COMPUTING ENVIRONMENTS

Deliverable D51: Implemented Adaptation of a Grid Resource Manager for Data Mining Tasks

Responsible author(s): Valentin Kravtsov
Co-author(s): David Carmeli, Thomas Niessen, Vlado Stankovski
Revision history

Deliverable administration and summary

<table>
<thead>
<tr>
<th>Project acronym: DataMiningGrid</th>
<th>ID: IST-2004-004475</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document identifier: DataMiningGrid-del-D51GridResourceManager-s-v06</td>
<td></td>
</tr>
<tr>
<td>Leading partner: TECH</td>
<td></td>
</tr>
<tr>
<td>Report version: 06</td>
<td></td>
</tr>
<tr>
<td>Report preparation date: 27.01.2006</td>
<td></td>
</tr>
<tr>
<td>Classification: Public</td>
<td></td>
</tr>
<tr>
<td>Nature: Prototype, Other</td>
<td></td>
</tr>
<tr>
<td>Author(s) and contributors: Valentin Kravtsov in collaboration with all partners</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
<th>Changes made</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan</td>
<td></td>
</tr>
<tr>
<td>Draft</td>
<td></td>
</tr>
<tr>
<td>Working</td>
<td></td>
</tr>
<tr>
<td>Final</td>
<td></td>
</tr>
<tr>
<td>X Submitted</td>
<td>A report template will be defined that all reports will follow</td>
</tr>
<tr>
<td>Approved</td>
<td></td>
</tr>
</tbody>
</table>

The DataMiningGrid © Consortium has addressed all comments received, making changes as necessary. Changes to this document are detailed in the change log table below.

<table>
<thead>
<tr>
<th>Date</th>
<th>Edited by</th>
<th>Status</th>
<th>Changes made</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.10.2004</td>
<td>Jernej Trnkoczy</td>
<td>Plan</td>
<td>A report template will be defined that all reports will follow</td>
</tr>
<tr>
<td>01.11.2005</td>
<td>Valentin Kravtsov</td>
<td>Draft</td>
<td>Filling the main paragraphs of the paper</td>
</tr>
<tr>
<td>12.12.2005</td>
<td>Valentin Kravtsov</td>
<td>Draft</td>
<td>Additions and corrections</td>
</tr>
<tr>
<td>27.01.2006</td>
<td>Werner Dubitzky</td>
<td>Submitted</td>
<td>Finalization for submission</td>
</tr>
</tbody>
</table>

Copyright

This report is © DataMiningGrid Consortium 2004. Its duplication is restricted to the personal use within the Consortium, funding agency and project reviewers.

Citation


Acknowledgements

The work presented in this document has been conducted in the context of the EU Framework Programme VI project IST 2004 004475 DataMiningGrid. DataMiningGrid is a 24-month project that started on 1 September, 2004 and is funded by the European Commission as well as by the industrial partners. Their support is appreciated.

The partners in the project are University of Ulster (UU), Fraunhofer Institute for Autonomous Intelligent Systems (FHG), DaimlerChrysler (DC), Israel Institute of Technology (TECH) and University of Ljubljana (LJU). The content of this
document is the result of extensive discussions within the DataMiningGrid© Consortium as a whole.

**More information**

Public DataMiningGrid reports and other information pertaining to the project are available through DataMiningGrid public Web site under [www.DataMiningGrid.org](http://www.DataMiningGrid.org).
Executive Summary

This document specifies the common design and describes the implementation of the Resource Broker Service, which was produced in scope of the DataMiningGrid project. It describes the static and dynamic structure of the service design, specifies the exposed APIs and provides the documentation needed for further enhancements of the service.

The strategic decision, which was taken by the Consortium, was to build our architecture on ‘cutting-edge’ grid technologies which are considered as the ‘future of the grid’. However, this also introduces a few technical problems, as such software is not always stable and complete and subject to changes.

As for this deliverable, the Consortium could not identify a mature and useful off-the-shelf resource broker implementation, which will be compliant with the Consortium-selected technology (WS Resource Framework). However, producing a fully functional resource broker is a considerable task, which is beyond the DataMiningGrid project. Therefore, the approach taken in the project is to develop a WSRF-compliant resource broker based on existent Resource Broker, provided by GridBus Project [GridBus05].

This deliverable is based on and references the following deliverables:

- D11: Common Requirements Analysis, Specification and Evaluation of DataMiningGrid Interfaces and Services
- D12: Common Design of DataMiningGrid Interfaces and Services
## Table of Contents

Executive Summary ........................................................................................................ 6  

Table of Contents ......................................................................................................... 7  

1 Introduction .............................................................................................................. 9  
   1.1 General Requirements ....................................................................................... 10  
   1.2 Specific requirements ...................................................................................... 10  

2 Design of the Resource Broker Implementation ..................................................... 12  
   2.1 Layered Diagram of Resource Broker .......................................................... 12  
   2.2 Wrapping resource broker as WSRF-compliant service ............................... 14  
   2.3 Information integrator introduction ............................................................... 14  
   2.4 Resource broker sequence diagrams ............................................................ 15  
      2.4.1 Sequence diagram for WS-Resource creation ....................................... 15  
      2.4.2 Sequence diagram for WS-Resource invocation ................................. 16  

3 Conclusions and Future Work .................................................................................. 17  
   3.1 Conclusions .................................................................................................... 17  
   3.2 Future work .................................................................................................... 18  

4 References ............................................................................................................... 19  

5 Acronyms and Abbreviations ................................................................................... 20  

6 Appendix A - SOA and State-of-the-Art Middleware ............................................. 21  
   6.1 Grid middleware – SOA approach ................................................................. 21  
   6.2 Grid middleware – Globus Toolkit 4 .............................................................. 22  
      6.2.1 XML, SOAP, WSDL ............................................................................. 22  
      6.2.2 WS-Security family ............................................................................... 23  
      6.2.3 WS-Addressing, WSRF, and WS-Notification ...................................... 23  
   6.3 Grid middleware – GridBus Resource Broker ............................................. 25
6.4 GridLab Resource Management System (GRMS) ........................................ 26
6.5 Community Scheduler Framework .......................................................... 27
1 Introduction

A grid resource broker is a grid middleware component that brokers the running of grid jobs (making use of information services to obtain grid status information about available resources) and schedules jobs. Previous deliverables, especially D12 (system architecture), emphasize the need for a resource broker service as one of the major components of DataMiningGrid architecture. In this document, we will show that given our requirements (D11), the system will be able to achieve its goals after introduction of our Resource Broker Service implementation. We will explain in details the dynamic and static structure of our Resource Broker Service implementation and survey an existing available solution, justifying our middleware selection.

Figure 1 indicates the relationship of the DataMiningGrid Resource Broker with other system components.
1.1 General Requirements

To understand what exactly we expect from the DataMiningGrid Resource Broker, we first must understand the environment within it is expected to operate. According to Ian Foster, a grid system is a system that:

1) 'coordinates resources that are not subject to centralized control ...'
A grid integrates and coordinates resources and users that live within different control domains—for example, a user’s desktop vs. central computing; different administrative units of the same company; or different companies; and addresses the issues of security, policy, payment, membership, and so forth that arise in these settings. Otherwise, we are dealing with a local management system.

2) '... using standard, open, general-purpose protocols and interfaces ...'
A grid is built from multi-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access. As it discussed further below, it is important that these protocols and interfaces be standard and open. Otherwise, we are dealing with an application-specific system.

3) '... to deliver non-trivial qualities of service.'
A grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating, for example, to response time, throughput, availability, and security, and/or co-allocation of multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts. [Foster02].

A resource broker within a grid is the service that provides the bridge between resource demands and available resources. From the grid definition we can conclude that a resource broker must also be able to coordinate resources which are not subject to centralized control and follow standard, open, general-purpose protocols and interfaces, and finally, to deliver non-trivial qualities of service.

1.2 Specific requirements

The requirements document (D11), includes several functional and non-functional requirements, lead to the incorporation of a resource broker into the DataMiningGrid architecture (which is described in detail in D12). We will list the most relevant requirements and explain why we opted for the resource broker solution in general and decided to adopt and adapt GridBus as our for our purposes in particular:

**Functional requirement 1: Middleware able to execute generated workflows including conditional control flows**

As mentioned in the previous deliverable documents, the main user access tool to DataMiningGrid functionality is Triana [Triana05a]. Triana is used to define, generate and to locally execute workflows. However, after a workflow is defined, it is still need to be executed on the grid. This is where the DataMiningGrid
Resource Broker comes in. The main purpose of the GridBus Resource Broker is to match jobs (user requests) with available grid resources, and to execute these jobs on the grid. Thus, a resource broker combined with Triana provides the user with a very powerful tool for executing workflows (including conditional control flows) utilizing grid resources.

**Functional Requirement 2: Provide monitoring services.**

One of the basic features of job execution in general and job execution on the grid in particular is the monitoring the state of jobs. In order for users to query the state of the jobs they initiated, a resource broker needs to implement this capability. The GridBus Resource Broker includes a full set of monitoring utilities, which are responsible for monitoring jobs from the moment they are submitted until their completion.

**Nonfunctional requirement 1 (supplementary): The DataMiningGrid services and tools should be compatible with existing grid infrastructure.**

Our implementation of WSRF-compatible GridBus Resource Broker follows the most recent de-facto standards. In addition, a future extension of the usage of such standards is planned (for example – next version of GridBus Resource Broker should support JSDL [JSDL05]).

**Nonfunctional requirement 2 (supplementary): Extensible and ‘future-proof’**

One of the reasons that the partners adopted the GridBus Resource Broker is that GridBus is actually an ongoing project. The GridBus middleware is well maintained and constantly updated. As explained below, we chose to expose a very limited set of functionality, thus minimizing the risk of future incompatibility and maximizing the simplicity of the system.

**Non-functional requirement 3 (supplementary): Grid infrastructure allows interoperability between heterogeneous programming environments and different operating systems**

Platform independence is a natural consequence of the fact that the GridBus Resource Broker is written in Java. In addition, WSRF-compatibility provides interoperability between heterogeneous programming environments.
2 Design of the Resource Broker
Implementation

The selected grid technology (WS-Resource Framework implemented with Globus
Toolkit 4) currently provides resource brokering services over the grid. Therefore, TECH in collaboration with all partners has designed and implemented
the DataMiningGrid Resource Broker. Our implementation is based on the
GridBus Resource Broker, which was described above.

2.1 Layered Diagram of Resource Broker

The GridBus Resource Broker follows a service-oriented architecture and is
designed on object-oriented principles with a focus on the idea of promoting
simplicity, modularity, reusability, extensibility and flexibility. The architecture of
the broker is shown in Figure 2.
Figure 2. GridBus Resource Broker Architecture – with DataMiningGrid extensions

The broker can be thought of as a system composed of three main sub-systems:

- The application interface sub-system

The broker can be thought of as a system composed of three main sub-systems:
• The core sub-system
• The execution sub-system.

The input to the broker is an application-description, which consists of tasks and the associated parameters with their values, and a resource description which could be in the form of a file specifying the hosts available or information service which the broker queries. At the application interface sub-system there is the application and resource-description. The application description interpreter and the resource discovery module convert these inputs into entities, called jobs and servers with which the broker works internally in the core sub-system. A job is an abstraction for a unit of work assigned to a node. It consists of a task and variables. A variable holds the designated parameter value for a job, which is obtained from the process of interpreting the application description. A server represents a node on the grid, which could provide a compute, storage, information or application service. The task requirements and the resource information drive the discovery of resources such as computational nodes, application and data resources.

2.2 Wrapping resource broker as WSRF-compliant service

GridBus Resource Broker is a very powerful tool, which was carefully picked from the variety of existent implementations of resource brokers. It has many benefits. However, it cannot be immediately integrated into the DataMiningGrid architecture as it is. The current version of GridBus Resource Broker (2.4) comes as a stand-alone application. In order to fit it into our architecture we must wrap it up as a WSRF-compliant service, exposing its main features through predefined Web interfaces. TECH has completed this work recently. Currently, we choose to expose only a very limited number of features through Web interfaces. Only the main functions such as job scheduling and monitoring are available to the user. In the future, the development team of the GridBus Resource Broker is planning to release a WSRF-compliant version, which will probably provide many advanced and modified features through Web interfaces. However, currently there is no formal definition of this process, and in order to keep the software compatible to future versions of the GridBus Resource Broker, we prefer to minimize the risk of exposing advanced features.

2.3 Information integrator introduction

The GridBus Resource Broker is designed to work with a small number grid information services. However, MDS4 (Monitoring and Discovery System – Globus Toolkit 4 implementation of information service) is currently not supported. We also implemented a few utilities, which provide the missing functionality. The purpose of our extension is to contact the MDS4 repository and query it for the available grid computation resources currently available in the grid. This facilitates that the Resource Broker obtains all necessary information. This is the first step towards introduction of an Information Integrator Service into DataMiningGrid system.
2.4 Resource broker sequence diagrams

2.4.1 Sequence diagram for WS-Resource creation

Figure 3. Sequence diagram for WS-Resource creation

1. Our client only needs to know the URI of the factory service (ResourceBrokerFactoryService). With it, it can invoke the createResource operation. This will return an endpoint reference containing the URI of the instance service, along with the key of the recently created resource.

2. So, the factory service has to create a new resource. This necessarily has to be done through the resource home, which is in charge of managing all the resources. However, we have to locate our resource home first. Fortunately, this is quite easy since we can delegate this task on a Globus-supplied class called ResourceContext. In the previous chapter, we used this class to retrieve the service’s singleton resource. It can also be used to obtain a reference to the service’s resource home.

3. Now that we have the resource home, we can ask it to create the new resource. The creation method returns an object of type ResourceKey. This is the resource identifier which we need to create the endpoint reference we will be returning to the client.

4. The resource home will take care of actually creating a new ResourceBrokerResource instance.

5. Next, the resource home will add the new ResourceBrokerResource instance to its internal list of resources. This list will allow us to access any resource if we know that resource’s identifier. [GDP05].
6. The client invokes the matchJobs operation in the instance service (ResourceBrokerService).
7. However, the matchJobs operation is stateless. It needs to retrieve a resource to actually work. The resource identifier is in the endpoint reference that is included in the invocation. Fortunately, the ResourceContext helper class once again shields us from all the potential nastiness. It will be in charge of reading the EPR and finding the resource it refers to.
8. However, it’s interesting to note that, internally, ResourceContext uses the ResourceHome to find the resource.
9. Once we have the resource, the instance service can access all its state information, such as the ‘FarmingEngineKey’ resource property. Our resource (ResourceBrokerResource) will allow us to modify the RP’s using get/set methods (in this case, with a setJobs() method).
10. Last thing to do – is actually to start job scheduling and return the results location to the user. [GDP05]
3 Conclusions and Future Work

3.1 Conclusions

We have identified the mostly suitable open source implementation of a grid resource broker and modified it to suit our architecture. As the result of this process, we have now a fully functioning WSRF-compliant resource broker (referred to as the DataMiningGrid Resource Broker or simply Resource Broker) which encapsulates the GridBus Resource Broker. The DataMiningGrid Resource Broker provides the following features:

- Discovery of resources on the grid
- Transparent access to computational resources running middleware such as:
  - Globus 2.4
  - Globus 3.2 (pre-WS)
  - Globus 4.0
  - Alchemi (1.0)
  - Unicore 4.1
  - XGrid v.1.0

- And queuing systems such as
  - Condor 6.6.9
  - OpenPBS 2.3.
  - Sun N1 Grid Engine 6 (SGE)
  - This includes support for all basic services like: job scheduling and execution for batch jobs, job monitoring and status reporting, and gathering output of completed jobs and directing it to user-defined locations.

- Economy based scheduling with built-in algorithms for cost, time and cost-time optimizations.
- Data-aware scheduling which considers network bandwidths, and proximity of data to computational resources
- XML-based application description format, XML-based resource description format
- Support for data sources managed by systems such as Storage Resource Broker (SRB), and the Globus Replica Catalog.
- Support for queuing systems such as PBS on clusters
- Persistence to enable failure management and recovery of an executing grid application
- Extensibility: the broker is engineered to support extensions in the form of custom schedulers, middleware plug-ins, application-description interpreters, and persistence providers.
- Platform independence, which is a natural consequence of a java implementation. [GridBus05].
3.2 Future work

The next thing to do is an extension of the WSRF interface. A certain risk is involved in this process, as we have to rely on the availability of future versions of the GridBus Resource Broker and that they will still be compatible with our WSRF wrappers. Currently, only the basic features of the Resource Broker are exposed. We also need to analyze and extend the Web resource functionality of the Resource Broker. This process will require closer collaboration with the GridBus project.

In the next phase of the Resource Broker integration into the system architecture, additional issues are likely arise and will require further adaptation of WSRF compatibility of the Resource Broker. Additionally, integration of the Resource Broker and Triana might require some sort of new functionality, which will be implemented in the future.

Extensive testing of the Resource Broker integrated into DataMiningGrid architecture should be one of the last but important milestones of the Resource Broker development. Currently, the Resource Broker acts as a standalone WSRF compliant service application. An integration of it into the system architecture will probably lead to code changes and enhancement which must be thoroughly tested before the project submission.
4 References

References cited in this document and throughout the project are listed in the DataMiningGrid Project Manual.
5 Acronyms and Abbreviations

Acronyms and abbreviations used in this document are listed and described more comprehensively in the DataMiningGrid Project Manual.
6 Appendix A - SOA and State-of-the-Art Middleware

6.1 Grid middleware – SOA approach

Over the last four decades, software architectures have attempted to deal with increasing levels of software complexity. However, the level of complexity continues to increase, and traditional architectures seem to be reaching the limit of their ability to deal with the problem. At the same time, traditional needs of IT organizations persist; the need to respond quickly to new requirements of the business, the need to continually reduce the cost of IT to the business, and the ability to absorb and integrate new business partners and new customer sets, to name a few. The industry has gone through multiple computing architectures designed to allow fully distributed processing, programming languages designed to run on any platform, greatly reducing implementation schedules, and a myriad of connectivity products designed to allow better and faster integration of applications. However, the complete solution continues to elude us. Now the concept of service oriented architecture (SOA) is being promoted in the industry as the next evolutionary step in software architecture to help IT organizations meet their ever more complex set of challenges. SOA is just that, an architecture. It is more than any particular set of technologies, such as Web services; it transcends them, and, in a perfect world, is totally independent of them. Within a business environment, a pure architectural definition of a SOA might be something like ‘an application architecture within which all functions are defined as independent services with well-defined invokable interfaces which can be called in defined sequences to form business processes’. Note what is being said here:

- All functions are defined as services. This includes purely business functions, business transactions composed of lower-level functions, and system service functions. This brings up the question of granularity, which will be addressed later.
- All services are independent. They operate as ‘black boxes’; external components neither know nor care how they perform their function, merely that they return the expected result.
- In the most general sense, the interfaces are invokable; that is, at an architectural level, it is irrelevant whether they are local (within the system) or remote (external to the immediate system), what interconnect scheme or protocol is used to effect the invocation, or what infrastructure components are required to make the connection. The service may be within the same application or in a different address space within an asymmetric multiprocessor, on a completely different system within the corporate Intranet, or within an application in a partner’s system used in a B2B configuration.

Grid computing is much more than just the application of large numbers of MIPS (millions of instructions per second) to effect a computing solution to a complex problem. Grid computing also involves the virtualization of all the system resources including hardware, applications, and data, so that they can be utilized wherever and however they are needed within the grid. Because of importance of virtualization of data sources and the decomposition of
applications into component-based services, it should be easily understood that a true SOA should better enable getting maximum resource utilization in a grid environment [IBM2003].

6.2 Grid middleware – Globus Toolkit 4

The Globus Toolkit is an open source software toolkit used for building grids. The Globus Alliance and many others all over the world are developing it. The toolkit includes software for:

- security,
- information infrastructure,
- resource management,
- data management,
- communication,
- fault detection,
- portability

It is packaged as a set of components that can be used either independently or together to develop applications. Every organization has unique modes of operation, and collaboration between multiple organizations is hindered by incompatibility of resources such as data archives, computers, and networks. The Globus Toolkit was conceived to remove obstacles that prevent seamless collaboration. Its core services, interfaces and protocols allow users to access remote resources as if they were located within their own machine room while simultaneously preserving local control over who can use resources and when. GT4 makes heavy use of Web services mechanisms to define its interfaces and structure its components. Web services provide flexible, extensible, and widely adopted XML-based mechanisms for describing, discovering, and invoking network services; in addition, its document-oriented protocols are well suited to the loosely coupled interactions that many argue are preferable for robust distributed systems. Web services specifications that underlie GT4 comprise the core specifications that define the Web services architecture (XML, SOAP, WSDL); WS-Security and other specifications relating to security; and the WS-Addressing, WSRF, and WS-Notification specifications used to define, name, and interact with stateful resources.

Web services specifications that underlie GT4:

This group of services comprise the core specifications that define the Web services architecture (XML, SOAP, WSDL); WS-Security and other specifications relating to security; and the WS-Addressing, WSRF, and WS-Notification specifications used to define, name, and interact with stateful resources.

6.2.1 XML, SOAP, WSDL

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

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

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

### 6.2.2 WS-Security family

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

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

### 6.2.3 WS-Addressing, WSRF, and WS-Notification

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

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

The WS Resource Framework (WSRF) specifications define a generic and open framework for modeling and accessing stateful resources using Web services.

The WS-Resource Framework specifications:

WS-ResourceLifetime support management of the state through properties associated with the Web service. It defines mechanisms for WS-Resource
destruction, including message exchanges that allow a requestor to destroy a resource, either immediately or by using a time-based scheduled resource termination mechanism.

WS-ResourceProperties defines how the type definition of a WS-Resource can be associated with the interface description of a Web service, and message exchanges for retrieving, changing, and deleting WS-Resource properties.

WS-RenewableReferences defines a conventional decoration of a WS-Addressing endpoint reference with policy information needed to retrieve an updated version of an endpoint reference when it becomes invalid.

WS-ServiceGroup describe how these mechanisms are extensible to groups of Web services

WS-BaseFaults defines a base fault XML type for use when returning faults in a Web services message exchange.

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

GT4 architecture summery

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

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

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

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


Standards - Wherever possible, Globus implements standards or other broadly adopted specifications - so as to facilitate the construction of operable and reusable components and the use of standard tools.
Related tools - GT4 components do not, in general, address end-user needs directly: most are more akin to a TCP/IP library or Web server implementation than a Web browser. Instead, GT4 enables a range of end-user components and tools that provide the higher-level capabilities needed by specific user communities. These components and tools constitute, together with GT4 itself, the ‘Globus universe’. [GT405].

6.3 Grid middleware – GridBus Resource Broker

The Gridbus Resource Broker is designed to support both computational and data grid applications. For example, it has been used to support composition and deployment of neuroscience (compute intensive) applications and High Energy Physics (Belle) Data grid applications on Global Grids. The architecture of the broker has emphasis on simplicity, extensibility and platform independence. It is implemented in Java and provides transparent access to grid nodes running various middleware. The main design principles of the broker include:

*Assume nothing about the environment*

No assumptions are made anywhere in the Broker code as to what to expect from the grid resource except for one - that the resource provides at least one way of submitting a job and if running a flavour of Unix will provide at least a POSIX shell. In addition, no assumption is made about resource availability throughout an execution. The implications of this principle have a huge impact throughout the broker such as

- The broker has no close integration with any of the middleware it supports. It uses the minimum set of services that are required to run a job on a resource supported by the middleware. The advantages of this are:
  - In a grid with multiple resources configured differently, the broker tries to make use of every resource possible by not imposing a particular configuration requirement. For example, in the case of Globus 2.4, all is required is that the GRAM service be set up properly on the resource.
  - The broker can run jobs on resources with different middleware at the same time.
  - The broker need not be refactored if there is a new version of the middleware.

- The broker is able to handle gracefully jobs and resources failing throughout an execution. The job wrapper and job monitor code is written to handle every failure status possible. The scheduler does not fail if a resource drops out suddenly.

- The failure of the broker itself is taken care of by the recovery module if persistence has been configured.
Client-centric design

The scheduler has just one target: that is to satisfy the users’ requirements especially if the deadline and budget are supplied. Even in the absence of these, the scheduler strives to get the jobs done in the quickest way possible. Thus, resources are evaluated by the scheduler depending on how fast or slow they are executing the jobs submitted by the broker. In keeping with Principle 1, the broker also does not depend on any metrics supplied by the resource – it does its own monitoring.

Extensibility is the key

In grid environments, transient behaviour is not only a feature of the resources but also of the middleware itself. Rapid developments in this still-evolving field have meant that middleware goes through many versions and unfortunately, interface changes are a norm rather than the exception. Also, changing requirements of grid users require that the broker itself be flexible enough for adding new features or extending old ones. Thus, every possible care has been taken to keep the design modular and clean. The advantages due to this principle:

- Extending broker to support new middleware is a zip – Requires implementation of only three interfaces. (For more details refer to Programming section)
- Getting broker to recognize the new information sources is also easy
- The differences in middleware are invisible to the upper layers such as the scheduler and vice versa. Thus any changes made in one part of the code remain limited to that section and are immediately applicable. For example, after adding a new middleware, the scheduler is immediately able to use any resource using that middleware.
- XPML is extensible. Adding any new constructs is easy, using the same reflection framework (see Programming Section). You could also do away with XPML altogether and implement your own favourite interface to describe applications. [GridBus05].

6.4 GridLab Resource Management System (GRMS)

GridLab Resource Management System (GRMS) is an open source meta-scheduling system, developed by PSNC under the GridLab Project, which allows developers to build and deploy resource management systems for large scale distributed computing infrastructures. GRMS provides developers of user-level functionalities with a more abstract view of low level and complex grid technologies. The main goal of GRMS is to manage the whole process of remote job submission and control to various queuing systems (e.g. Condor, PBS, LSF, N1 Grid Engine), clusters systems or resources directly. GRMS main features include:

- Modular & plugable architecture,
- Dynamic resource selection and discovery
- Advanced scheduling algorithms and plugins available for:
Deliverable D51: Grid Resource Manager

- multicriteria optimization of resource allocations
- job re-scheduling (using user-level checkpointing and job migration)
- time-constraint scheduling (advanced resource reservation),
- job dependency and work-flow management
  - Portable (all internal modules implemented in Java)
  - Compatibility and integration with common technologies, including Globus Pre-WS and new GT4/WSRF services, Mercury grid monitoring system, iGrid, and many others
  - Email alerts and SMS notifications for users and administrators,
  - Enhanced support for VO-level authorization scenarios with GAS
  - Detailed job run accountability and tracking
  - Remote job control
  - File staging support
    - Scalability and capability of handling the configuration and processing of thousands of jobs.

New features in GRMS v2.0 (not available yet):
- Improved stability and fault tolerance, new (Server side) workflow module to deal with complex job dependencies and time constraints flexible multicriteria resource mapping and job scheduling.

GRMS was seriously considered as a core component for the DMG project. It was finally rejected because the following reasons:

- GRMS is currently not WSRF enabled, and relies on old technology. Currently stable in conjunction with GT2, and so not integrate with the new WSRF GT4 components.
- New GT4 support is planned gradually. This year GRMS will use GSI security, MDS2, GridFTP, and GT4 GRAM. The next phase (no clear schedule yet) will integrate it with MDS4 and RFT
- GRMS development roadmap is still unclear, according to PSNC partnership with commercial companies. No precise information regarding this issue could be obtained from PSNC at this stage.

6.5 Community Scheduler Framework

The Community Scheduler Framework (CSF) is an open source framework for implementing a grid meta-scheduler. It’s an add-on to the Globus Toolkit v.3 and was developed by Platform Computing. The Community Scheduler Framework is OGSI-compliant and supports WS-Agreement, a draft specification of the Global Grid Forum. WS-Agreement is an XML language for specifying an agreement between a resource or service provider and a consumer, and a protocol for creation of an agreement using agreement templates.

The CSF is composed of several services:

- The Job Service creates, submits, controls, and monitors jobs.
- The Reservation Service reserves resources on the grid, guaranteeing that resources will be available for running jobs when the scheduled jobs require them.
The Queuing Service supports basic scheduling capability. It allows administrators to customize existing policies and define new scheduling policies for their virtual organization. The Queuing Service also provides an application programming interface (API) for scheduler plug-ins. Scheduler plug-ins are used to extend existing scheduling policies or implement custom scheduling policies.

The RM-Adapter Service provides a grid service interface that translates information between the grid service protocol and the protocol required by the various heterogeneous local cluster schedulers or resource managers such as LSF, Open PBS, SGE, and Condor.

CSF Supports:

- Round-robin job scheduling
- Advance reservation booking, query and control
- Reservation-based job scheduling
- Job throttling support for increased reliability
- File staging support

Why not CSF?

- The currently official release version of CSF supports only GT v3.2 and is not WSRF compliant.
- While documentation on the Community Scheduler Framework 4.0 which suppose to support GT4 do exist in the Globus Web site, this documentation is minimal and not exhaustive. There is no official ‘CSF 4.0’ version available for download.
- Further development of CSF by Platform Computing (Developers of LSF) is not seen in the near future, and there is no available support for this product.