

# DataGrid

## DEFINITION OF ARCHITECTURE, TECHNICAL PLAN AND EVALUATION CRITERIA FOR SCHEDULING, RESOURCE MANAGEMENT, SECURITY AND JOB DESCRIPTION

---

Document identifier:	<b>DataGrid-01-D1.2-0112-0-3</b>
Date:	<b>14/09/2001</b>
Work package:	<b>WP1: Workload Management</b>
Partner:	<b>INFN</b>
Document status	<b>DRAFT</b>
Deliverable identifier:	<b>DataGrid-D1.2</b>

---

Abstract: This document provides a description of the architecture of the Workload Management System, including the User Interface, the Resource Broker, the Job Submission and the Logging and Bookkeeping services. It also explains the evaluation criteria that the Resource Broker applies to select resources, the Job Description Language, and the security infrastructure for the Workload Management System.



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

**Delivery Slip**

	<b>Name</b>	<b>Partner</b>	<b>Date</b>	<b>Signature</b>
<b>From</b>	Francesco Giacomini	INFN	14/09/01	
<b>Verified by</b>	Francesco Prelz	INFN	14/09/01	
<b>Approved by</b>				

**Document Log**

<b>Issue</b>	<b>Date</b>	<b>Comment</b>	<b>Author</b>

**Document Change Record**

<b>Issue</b>	<b>Item</b>	<b>Reason for Change</b>

**Files**

<b>Software Products</b>	<b>User files</b>
Word	DataGrid-01-D1.2-0112-0-3.doc



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

---

**CONTENT**

<b>1</b>	<b>INTRODUCTION .....</b>	<b>4</b>
1.1	OBJECTIVES OF THIS DOCUMENT.....	5
1.2	APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS.....	6
1.3	DOCUMENT EVOLUTION PROCEDURE.....	6
1.4	TERMINOLOGY.....	7
<b>2</b>	<b>EXECUTIVE SUMMARY .....</b>	<b>8</b>
<b>3</b>	<b>THE LOGGING AND BOOKKEEPING SERVICE.....</b>	<b>9</b>
3.1	JOB STATUS .....	9
3.2	JOB IDENTIFICATION.....	10
3.3	EVENTS.....	11
3.3.1	Event types.....	11
3.3.2	Event sources.....	12
3.4	EXTERNAL INTERFACES.....	13
3.4.1	The Producer API.....	13
3.4.2	The Consumer API.....	13
3.5	INTERNAL DESIGN .....	14
<b>4</b>	<b>THE JOB SUBMISSION SERVICE .....</b>	<b>17</b>
4.1	EXTERNAL INTERFACES.....	17
4.2	DEPENDENCIES .....	17
4.3	INTERNAL DESIGN .....	17
4.4	RECOVERY FROM FAILURES .....	18
<b>5</b>	<b>THE RESOURCE BROKER .....</b>	<b>19</b>
5.1	EXTERNAL INTERFACES.....	19
5.1.1	Job Control API.....	19
5.1.2	Notification API.....	20
5.2	DEPENDENCIES .....	20
5.3	EVALUATION CRITERIA FOR SCHEDULING.....	20
5.4	INTERNAL DESIGN .....	21
<b>6</b>	<b>THE USER INTERFACE.....</b>	<b>25</b>
6.1	THE JOB DESCRIPTION LANGUAGE.....	25
6.2	FUNCTIONAL MODEL.....	27
6.2.1	Job Control.....	27
6.2.2	Job Monitoring.....	28
6.3	DEPENDENCIES .....	28
<b>7</b>	<b>THE SECURITY FRAMEWORK.....</b>	<b>29</b>
7.1	SECURITY IN THE WORKLOAD MANAGEMENT SYSTEM.....	29



# DEFINITION OF ARCHITECTURE, TECHNICAL PLAN AND EVALUATION CRITERIA FOR SCHEDULING, RESOURCE MANAGEMENT, SECURITY AND JOB DESCRIPTION

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

## 1 INTRODUCTION

The Workload Management System (WMS) is the component of the DataGrid middleware that has the responsibility of managing the Grid resources in such a way that applications are conveniently, efficiently and effectively executed. Several interacting WMS components were identified in the architecture definition phase of the DataGrid project, and are described in the present document: the User Interface, the Resource Broker, the Job Submission Service, and the Logging and Bookkeeping Service.

The User Interface allows a user to interact with the Grid in order to perform operations such as submit jobs, control their execution, and retrieve their output. A job is represented by a Job Description, expressed in a Job Description Language (JDL), which typically includes, but is not limited to, the executable to run, the input and output specifications, the resources required to run the job.

The Resource Broker, given a Job Description, tries to find the best match between the job requirements and the resources available on the Grid, whose characteristics are retrieved from information services provided by the Data Management Work Package (WP2) and by the Monitoring Work Package (WP3). The output of the search is a Computing Element where the job, while running, has access to all resources specified in the Job Description, such as data or storage space.

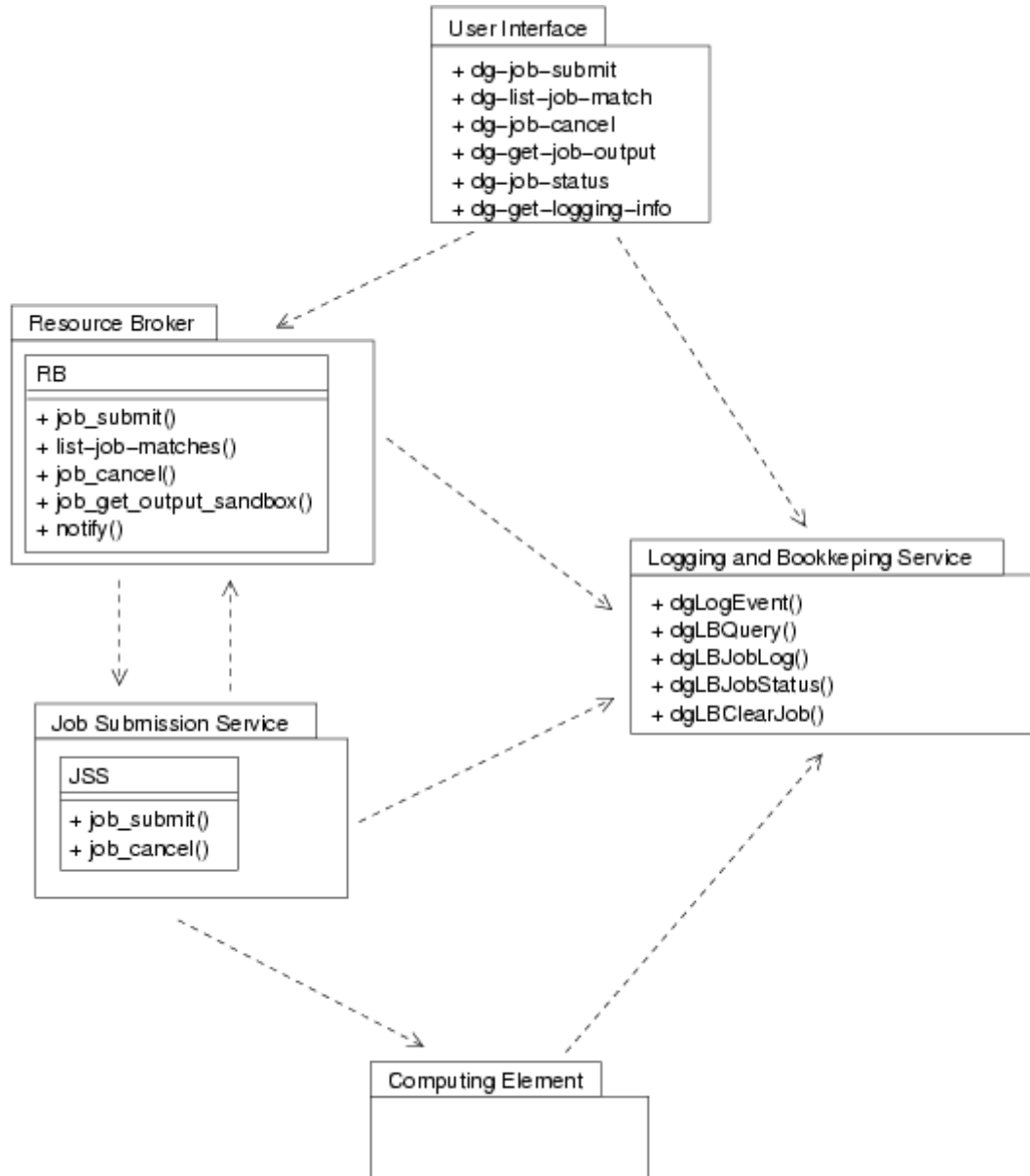
The Job Submission Service performs the actual submission of a job to a Computing Element. Its purpose is also to monitor the job execution and react if an event affects the job.

The Logging and Bookkeeping Service collects information about the scheduling system (referred to as *logging information* in the scope of this document) and about active jobs (or *bookkeeping information*) from the other components of the Workload Management System.

The dependencies among the software components of the Workload Management System are shown in Figure 1-1. For each component also its main public interface is summarised.

As shown in the figure, the Logging and Bookkeeping Service does not depend on any of the other components. Therefore it will be presented first, in Section 3. The Job Submission Service, the Resource Broker and the User Interface are then presented in Sections 4, 5 and 6 respectively.

All communication between components running on different machines is properly authenticated and encrypted where needed. All operations require an appropriate authorization to be granted. The security framework is presented in Section 7.



**Figure 1-1.** The software components that are part of the Workload Management System and their dependencies.

## 1.1 OBJECTIVES OF THIS DOCUMENT

The primary objective of this document is to give a view of the Workload Management System, to describe the offered functionality and the interactions between the identified components.



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

*Doc. Identifier:*  
DataGrid-01-D1.2-0112-0-3

*Date:* 14/09/2001

---

## 1.2 APPLICABLE DOCUMENTS AND REFERENCE DOCUMENTS

### Applicable documents

- [A1] The DataGrid Architecture – Version 2  
<http://grid-atf.web.cern.ch/grid-atf/doc/architecture-2001-07-02.pdf>

### Reference documents

- [R1] The Logging and Bookkeeping Architecture  
[http://www.cnaf.infn.it/giacco/wp1/lb\\_draft.pdf](http://www.cnaf.infn.it/giacco/wp1/lb_draft.pdf)
- [R2] Job Submission Service Architecture and APIs  
<http://www.pd.infn.it/~sgaravat/Grid/jss-arch.pdf>
- [R3] Resource Broker Architecture and APIs  
<http://www.infn.it/workload-grid/docs/20010613-RBArch-2.pdf>
- [R4] Job Submission User Interface Architecture  
[http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0103-0\\_0-Document.pdf](http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0103-0_0-Document.pdf)
- [R5] Home Page for the Condor Project  
<http://www.cs.wisc.edu/>
- [R6] Job Description Language HowTo  
[http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0\\_0-Document.pdf](http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_0-Document.pdf)
- [R7] Job Submission User Interface man Pages  
[http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0101-0\\_1-Document.pdf](http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0101-0_1-Document.pdf)
- [R8] Security in WP1  
<http://www.infn.it/workload-grid/docs/20010601-security-draft-ruda.pdf>
- [R9] Ten Actions When Superscheduling  
Scheduling Working Group, Global Grid Forum  
<http://www.cs.nwu.edu/~jms/sched-wg/WD/schedwd.8.5.pdf>
- [R10] Universal format for logger messages - IEEE draft  
<http://www-didc.lbl.gov/NetLogger/draft-abela-ulm-05.txt>
- [R11] WP1 Inputs to the DataGrid Grid Information Service Schema Specification  
<http://www.infn.it/workload-grid/docs/wp1-gis-gos.pdf>

## 1.3 DOCUMENT EVOLUTION PROCEDURE

The requirements definition process in the DataGrid project is deeply influenced by the novelty of the proposed computing model. The Applications end users and developers will need to acquire experience with the Grid environment in order to steer and refine their requirements. This, along with



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

*Doc. Identifier:*  
DataGrid-01-D1.2-0112-0-3

*Date:* 14/09/2001

---

fast evolution in the available technology, may require that the architecture described in this document be revised in the course of the project.

## **1.4 TERMINOLOGY**

### **Glossary**

API	Application Programming Interface
CA	Certification Authority
CE	Computing Element
DBMS	Data Base Management System
JDL	Job Description Language
JSS	Job Submission Service
LB	Logging and Bookkeeping
PKI	Public Key Infrastructure
RC	Replica Catalog
RB	Resource Broker
UI	User Interface
ULM	Universal Logger Message
VO	Virtual Organization
WMS	Workload Management System
WP	Work Package



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

---

## 2 EXECUTIVE SUMMARY

This document describes the breakdown and architectural details of the scheduling, resource management and job description services that will be provided by the Workload Management Work Package (WP1) of the DataGrid project. The inter-relations (and APIs) among the various identified WP1 components and with services provided by other DataGrid Work Packages (as identified by the project Architecture Task Force [A1]) are described. The details of the internal design and implementation of the WP1 components can be found in [R1-R8].

The components of the Workload Management System described in this document are:

- The Logging and Bookkeeping Service. It provides a reliable storage and retrieval service for logging and bookkeeping information generated by other WMS components and sent to the service in the form of event messages. In particular from the events stored in the logging and bookkeeping databases it is possible to reconstruct the status of a job that was previously submitted to a Resource Broker for execution on the Grid.
- The Job Submission Service. It is responsible for the actual job management operations (submission, cancellation and monitoring) once the Resource Broker has chosen a suitable Computing Element for running the job.
- The Resource Broker. It is the core component of the Workload Management System. Its main task is to find a Computing Element that best matches the requirements and preferences of a submitted job, considering also the current distribution of load on the Grid. Once a suitable Computing Element is found, the job is passed to the Job Submission Service for the actual submission. Additionally the Resource Broker allows cancelling a job and retrieving the output once a job has completed.
- The User Interface. It allows a user to access the functionality offered by the Grid, in particular the possibility to submit jobs and retrieve their output.

The description of a job is expressed in the Job Description Language, which is based on the *Classified Advertisements* developed by the Condor project. The choice is justified by some useful characteristics, including:

- Semi-structured data model: no specific schema is required.
- Symmetry: all entities in the Grid, in particular applications and computing resources, can be expressible in the same language.
- Simplicity, for both syntax and semantics.

Feedback from the applications, in the duration of the DataGrid project, will determine whether any extensions to this JDL model are needed.

The Workload Management System is also the only DataGrid component that is able to uniquely identify a job in the system. The unique job identifier is a URI assigned by the User Interface, which includes:

- The address of the Logging and Bookkeeping Service that keeps the information concerning the job.
- The address of the Resource Broker that was used to schedule the job.
- A string including a timestamp, a process ID and a random number to guarantee uniqueness.

### 3 THE LOGGING AND BOOKKEEPING SERVICE

The Logging and Bookkeeping (LB) Service [R1] is the Grid service responsible to store and manage logging and bookkeeping information generated by the various components of the WMS. The same LB Service can be used by multiple instances of the same component and store information about multiple jobs of multiple users.

For the purpose of this document the following definitions hold:

- Bookkeeping information refers to currently active jobs, i.e. jobs that are within the Workload Management System. It consists of the job definition, expressed in JDL, its status (see Section 3.1), resource consumption and possibly user-defined data. This information is typically kept by the system for the whole job lifetime and for a relatively short time afterwards.
- Logging information concerns the Workload Management System itself. These data are kept for a longer term and are used mainly for debugging, auditing and statistical purposes.

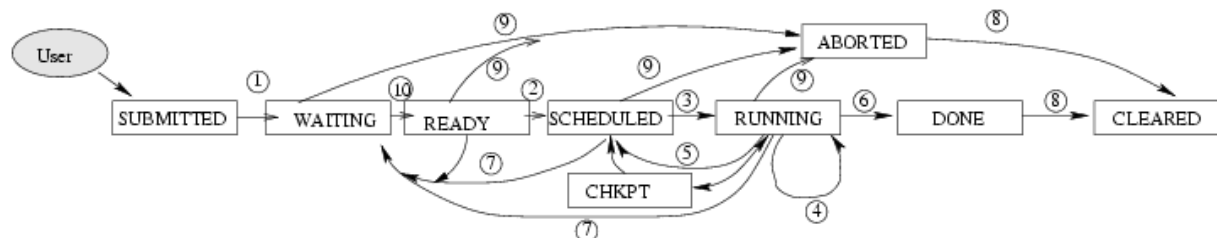
Some data may be stored both as bookkeeping and logging information, but the use of those data would be different in the two cases.

#### 3.1 JOB STATUS

During its lifetime a job may go through a number of states:

- SUBMITTED** The user has submitted the job to the User Interface.
- WAITING** The Resource Broker has received the job.
- READY** A Computing Element matching job requirements has been selected.
- SCHEDULED** The Computing Element has received the job.
- RUNNING** The job is running on a Computing Element.
- CHKPT** The job has been suspended and checkpointed on a Computing Element.
- DONE** The execution of the job has completed.
- ABORTED** The job has been terminated.
- CLEARED** The user has retrieved all output files successfully. Bookkeeping information is purged some time after the job enters this state.

The above states and the allowed transitions between them are shown in Figure 3-1.



**Figure 3-1. Job lifecycle.**

The allowed transitions are triggered by the following events (the numbers correspond to those in Figure 3-1):

1. A job submission request is passed by the UI to the RB.



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

2. A job submission request is passed by the JSS to the chosen CE.
3. The job is started on a CE.
4. The job has been affected by an event caused by the CE local resource management system, e.g. it has been suspended or its priority has been lowered.
5. The job is stopped and queued again in the CE.
6. The job has completed.
7. The job is passed again to the RB.
8. The bookkeeping information about the job is cleared.
9. The job is terminated.
10. A suitable CE has been found where the job can run.

A typical scenario for the life cycle of a job is as follows: a user submits the job through the User Interface. The UI looks for an appropriate Resource Broker and, when found, logs the submission event to the Logging and Bookkeeping Service. The job is now in the SUBMITTED state. While the job is considered by the RB it is in state WAITING. When a suitable Computing Element is found, the job is passed to the Job Submission Service and enters the READY state. JSS then submits the job to the Computing Element and the job enters the SCHEDULED state. From the SCHEDULED state the job typically first goes to RUNNING and then to DONE (if the job execution can complete) or to ABORTED (if the job execution is terminated). The final CLEARED state is achieved either when output data are retrieved or when a pre-specified timeout expires (the user has not reacted to the DONE or ABORTED states).

A job status will not be explicitly stored in the logging and/or bookkeeping databases but will be determined from the appropriate stored events in a coherent fashion according to the state diagram in Figure 1-1. Inconsistent event patterns due to event distribution failures will need to be dealt with.

Not storing the state directly, the system is more robust with respect to time differences between individual components (no strict time synchronization is expected); it may easily happen that information about the most recent event arrives earlier than information about an older one.

### **3.2 JOB IDENTIFICATION**

Depending on the context, a job can be referred to using different identifiers, including:

`dg_jobId` is the identifier used between Grid components. `dg_jobId` is also the main key for all the entries in the logging and bookkeeping databases.

`jss_jobId` is the identifier within the Job Submission Service.

`local_jobId` is the identifier assigned by the local resource management system which lies behind a Computing Element.

`dg_jobId` is generated by the User Interface upon job submission, since the UI is the first component that logs an event in the LB database.

A job must be uniquely identifiable on the whole Grid. `dg_jobId` serves as the Grid-wide identifier. It is the first and usually only job identifier directly used by users. To achieve uniqueness `dg_jobId` includes:

1. The IP number of the UI machine.
2. A timestamp.



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

*Doc. Identifier:*  
DataGrid-01-D1.2-0112-0-3

*Date:* 14/09/2001

3. The identifier of the UI process.
4. A random number.

The actual format of `dg_jobId` is a URL:

```
https://<LB server address>/<unique string>?<RB server address>
```

where `<LB server address>` is the specification of the LB server name and port, `<unique string>` is the string specified above and `<RB server address>` is the specification of the RB server name and port used to submit the job. For the use external to the Workload Management System `dg_jobId` should be taken as an opaque string, i.e. its internal structure should not be in any way interpreted.

### 3.3 EVENTS

Logging and bookkeeping is based on a “push” model, whereby the various WMS components and the Computing Element actively send suitable messages to the Logging and Bookkeeping Service whenever certain events occur. Usually these events cause job state transitions; therefore they need to be notified to the LB Service in order to keep the job state up-to-date.

A job state transition is recorded in the LB database as one or two events generated by WMS components. A single event is sufficient when the transition is within a component. When the state transition means also moving the responsibility for the job management between two components two events are preferred, one generated by the component that passes the job on and one generated by the component that accepts the job. Such an approach also helps in detecting communication problems.

Besides job state transitions, additional important events, such as a status change of a WMS component, must be considered as well.

The user and/or the system administrator can control the verbosity level of log messages, although a certain minimum level is guaranteed and enforced (i.e. the LB server cannot be switched off).

An event carries at least information on its type, the time it was generated and the source. If the event concerns a job it also contains its `dg_jobId`. Event types and sources are presented in the next two sections.

#### 3.3.1 Event types

Event types are organized in several categories.

- Events concerning a job transfer between components:

`JobTransferEvent` A component generates this event when it tries to transfer a job to some other component. This event contains the identification of the receiver and possibly the job description expressed in the language accepted by the receiver. The result of the transfer, i.e. success or failure, as seen by the sender is also included.

`JobAcceptedEvent` A component generates this event when it receives a job from another WMS component. This event contains also the locally assigned job identifier.

`JobRefusedEvent` A component generates this event when the transfer of a job to some other component fails. The source of this event, which also includes the reason for the failure, can be either the receiver or the sender, e.g. when the receiver is not available.



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

- Events concerning a job state change during processing within a component:
  - JobAbortEvent The job processing is stopped due to system conditions or user request. The reason is included in the event.
  - JobRunningEvent The job is started on a CE.
  - JobChkptEvent The job is checkpointed on a CE. The reason is included in the event.
  - JobScheduledEvent The job has been rescheduled on a CE.
  - JobDoneEvent The job has completed. The process exit status is included in the event.
  - JobClearedEvent The user has successfully retrieved the job results, e.g. the output files specified in the output sandbox (see Section 6.1); the job will be removed from the bookkeeping database in the near future.
- Events associated with the Resource Broker only:
  - JobMatchEvent An appropriate match between a job and a Computing Element has been found. The event contains the identifier of the selected CE.
  - JobPendingEvent A match between a job and a suitable Computing Element was not found, so the job is kept pending by the RB. The event contains the reason why no match was found.
- System specific events:
  - ComponentStatusEvent A WMS component has changed its state, e.g. it has been restarted.
  - ClusterStatusEvent A CE has changed its state, e.g. it has been rebooted.

These events are not related to any specific job, so they are only recorded in the logging database.
- Dynamic events:
  - JobStatusEvent It contains information about resources consumed by the job. This event is generated periodically by the CE and eliminates the need for direct communication from the LB Service to the CE. Two types of information should be considered: cumulative information (e.g. CPU time) and non-cumulative information (e.g. memory consumption). For cumulative properties only the most recent value is kept in the database. For non-cumulative properties, on the other end, all the values are stored in order to allow for example the creation of appropriate profiles.

### 3.3.2 Event sources

All the components of the Workload Management System and the Computing Element log events to the logging and/or bookkeeping databases, using the producer API described in Section 3.4.1.

The User Interface generates only events of type JobTransferEvent.

The Resource Broker can generate events of type JobTransferEvent, JobAcceptedEvent, JobRefusedEvent, JobAbortEvent, ComponentStatusEvent, JobMatchEvent, JobPendingEvent.



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

*Doc. Identifier:*  
DataGrid-01-D1.2-0112-0-3

*Date:* 14/09/2001

The Job Submission Service can generate events of type JobTransferEvent, JobAcceptedEvent, JobRefusedEvent, JobAbortEvent, ComponentStatusEvent.

The Computing Element can generate events of type JobTransferEvent, JobAcceptedEvent, JobRefusedEvent, JobRunningEvent, JobChkptEvent, JobScheduledEvent, JobDoneEvent, JobAbortEvent, JobStatusEvent, ComponentStatusEvent.

### **3.4 EXTERNAL INTERFACES**

The external interface offered by the Logging and Bookkeeping Service consists of two APIs:

1. Producer (Logging) API, used by event sources to log information into the logging and/or bookkeeping databases.
2. Consumer (Server) API, used to retrieve information stored in the logging and/or bookkeeping databases.

#### **3.4.1 The Producer API**

The producer API and its implementation should conform to the following requirements:

1. Non-blocking calls: clients calling the producer API should never be blocked indefinitely by the logging operation, even if the service is down.
2. Atomicity: no partial message can be stored.
3. Reliability: the Logging and Bookkeeping Service should be highly reliable and available.
4. Local persistence: logging and bookkeeping information should be kept locally persistent on the machine where it is produced until it is correctly stored in a possibly remote database.

For the sake of simplicity of use the producer API consists of a single function, `dgLogEvent()`, that any component can use to send an event to the LB Service. The function call arguments include the `dg_jobId`, the source component, the event type, and the set of attributes specific to that event.

#### **3.4.2 The Consumer API**

The consumer API allows a client application to retrieve information stored in the logging and bookkeeping databases and to have some control on the bookkeeping process (e.g. clearing the information about a job).

The API defines several data types and functions.

The following data types are defined:

<code>dgJobId</code>	Grid job identifier.
<code>dgLBErrCode</code>	Error code.
<code>dgLBContext</code>	Context of calls to LB functions.
<code>dgLBEventCode</code>	Event type identifier.
<code>dgLBEvent</code>	Event type.
<code>dgLBQueryRec</code>	LB query.
<code>dgLBJobStatCode</code>	Job status identifier.
<code>dgLBJobStat</code>	Job status.



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

The API functions are classified in several categories: job identifier manipulation, context handling, error handling, event handling, connection management, querying.

### **Job identifier manipulation**

`dgJobIdCreate ()` create a `dgJobId`.  
`dgJobIdParse ()` convert a generic string of characters to a `dgJobId`.  
`dgJobIdUnparse ()` convert a `dgJobId` into a generic string of characters.  
`dgJobIdLBserver ()` extract from a `dgJobId` the LB server contact string.  
`dgJobIdResourceBroker ()` extract from a `dgJobId` the RB contact string.

### **Context Handling**

`dgLBInitContext ()` allocate a `dgLBContext`.  
`dgLBFreeContext ()` free a `dgLBContext`.

### **Error Handling**

`dgLBError ()` given a `dgLBContext` return a human-readable error for a previous call to a LB function.

### **Event Handling**

`dgLBFreeEvent ()` free a `dgLBEvent`.  
`dgLBParseEvent ()` convert the string representation of an event to a `dgLBEvent`.  
`dgLBUnparseEvent ()` convert a `dgLBEvent` to its string representation.

### **Connection Handling**

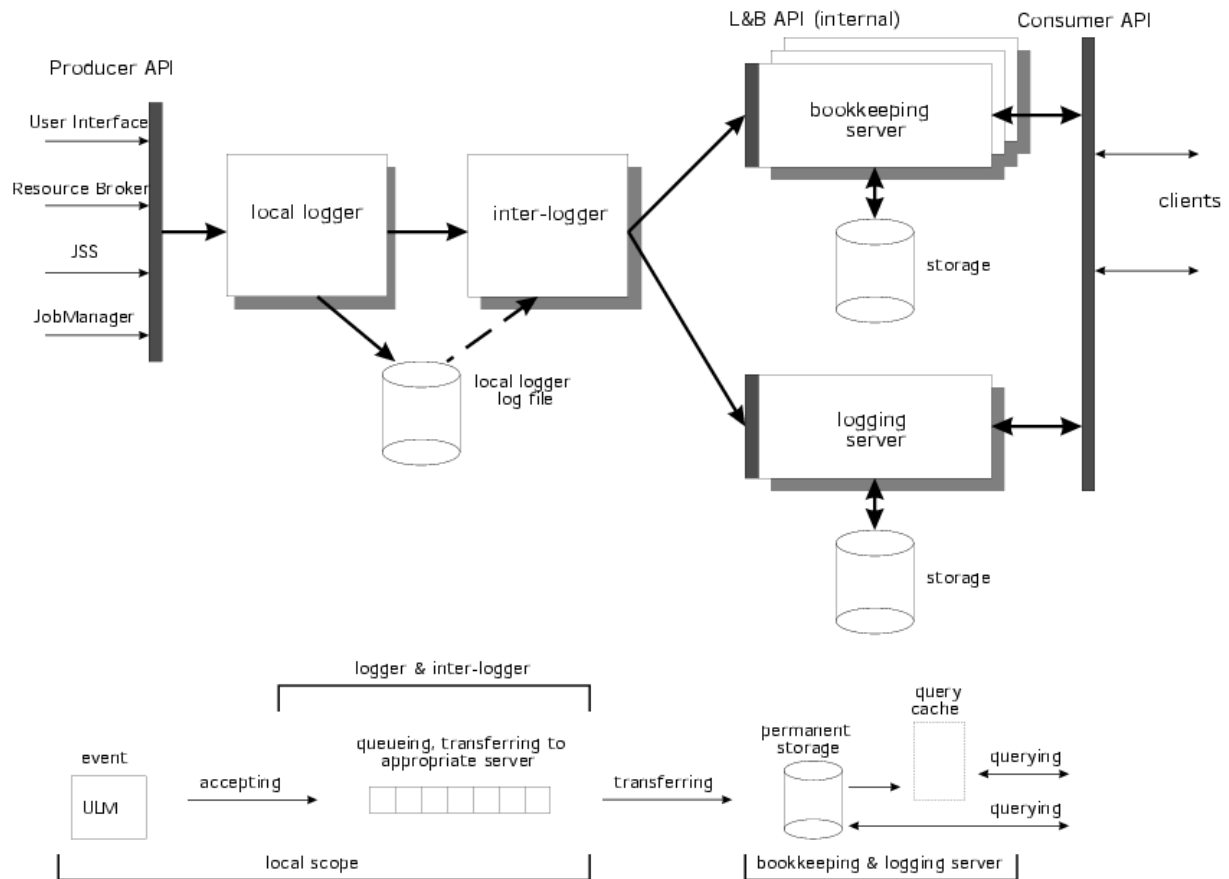
`dgLBOpen ()` connect to a LB, initialising a `dgLBContext`.  
`dgLBClose ()` close the connection to a LB server.

### **Querying**

`dgLBQuery ()` query the LB Service according to a specified `dgLBQueryRec`.  
`dgLBJobLog ()` query the LB Service for all the information about a given job.  
`dgLBUserLog ()` query the LB Service for all the information for all the jobs belonging to a given user.  
`dgLBUserJobs ()` list the `dg_jobIds` for all the jobs belonging to a given user.  
`dgLBJobStatus ()` get the status of a given job, including the bookkeeping information.  
`dgLBClearJob ()` purge all the bookkeeping information for a given job.

## **3.5 INTERNAL DESIGN**

The internal architecture of the Logging and Bookkeeping Service is sketched in Figure 3-2. It is designed to be highly reliable and available, so that it can support both a community Resource Broker, presumably itself running on a highly available server, and a personal Resource Broker running on a user's personal machine, which usually is not very reliable and can be even disconnected from the Grid for most of the time. In the latter case it is fundamental that state information is not kept on the RB but in a safer place.



**Figure 3-2. Overview of the logging architecture.**

The design foresees several cooperating components.

The **local logger** is responsible for accepting messages from their sources via the producer API and for passing them to the **inter-logger**, which is then responsible to forward them to the logging and/or bookkeeping servers. The inter-logger, running as a separate process, makes the logging procedure robust with respect to local and network faults.

The information flow between the local-logger and the inter-logger is implemented on top of inter-process communication mechanisms and is backed up by a log file that allows a correct recovery of the inter-logger if some problems occur.

The **bookkeeping server** and the **logging server** accept messages from the inter-logger and save them on their permanent storage. They also support queries generated by the consumer API. In particular the bookkeeping server supports job state queries.

Event messages are in ULM (Universal Logger Message) format [R10].

Answers to job state queries are computed on demand from the sequence of stored events. For reasons of efficiency a cache mechanism is also considered, whereby a finite state automaton for each job is asynchronously kept up-to-date starting from the relevant log messages. A query is then simply translated in a read operation of the current automaton state.



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

*Doc. Identifier:*  
**DataGrid-01-D1.2-0112-0-3**

*Date:* 14/09/2001

---

In the future the bookkeeping server will also provide an event subscription service: a user can declare its interest for a certain event and when such an event occurs a user-specified action is performed.



# DEFINITION OF ARCHITECTURE, TECHNICAL PLAN AND EVALUATION CRITERIA FOR SCHEDULING, RESOURCE MANAGEMENT, SECURITY AND JOB DESCRIPTION

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

## 4 THE JOB SUBMISSION SERVICE

The Job Submission Service (JSS) [R2] is the component of the Workload Management System responsible for the actual job management operations, issued on request of the Resource Broker.

In particular the JSS is responsible to manage the job submission and job removal requests, interacting with Computing Elements. It is also responsible to monitor submitted jobs until their completion and to notify the Resource Broker and/or the Logging and Bookkeeping Service when significant events occur.

In the proposed architecture the JSS is strictly coupled with a Resource Broker; therefore a JSS will be deployed for each installed RB.

### 4.1 EXTERNAL INTERFACES

The Job Submission Service offers a single interface, used by the Resource Broker, whose API includes two functions:

`job_submit()` submit a job to the specified Computing Element, managing also input and output sandboxes (see Section 6.1).

`job_cancel()` kill a list of jobs, identified by their `dg_jobId`.

Both `job_submit()` and `job_cancel()` are non-blocking operations. Their success or failure is notified asynchronously to the Resource Broker (see Section 5.1.2).

A job submission request from the Resource Broker consists of the original job description (the JDL expression sent to the Resource Broker by the User Interface) augmented by the Computing Element chosen by the broker. Before being submitted, the user job is wrapped in another job, whose purpose is to create the appropriate execution environment on the CE worker node. This includes the download of the *input sandbox*, i.e. the files needed to run the job, and the upload of the *output sandbox*, i.e. the files that the job produces and that the user will retrieve at job completion. The download and upload are from and to a storage area the RB and the JSS have access to.

The job submission and removal operations to the CE are executed on behalf of the user using some delegated credentials (see Section 7).

### 4.2 DEPENDENCIES

As shown in Figure 1-1, the Job Submission Service relies on services provided by other WMS components:

- The Resource Broker offers an asynchronous notification functionality. The events that the JSS can pass to the RB include the results for the `job_submit()` and `job_cancel()` operations of the JSS and events concerning submitted jobs.
- The Logging and Bookkeeping Service allows logging interesting events affecting jobs or the JSS itself.

Of course the JSS also interacts with Computing Elements, where jobs actually run.

### 4.3 INTERNAL DESIGN

The internal design of the Job Submission Service is based on a client-server model: a master thread listens to requests coming from the Resource Broker and dispatches them to slave threads for



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

*Doc. Identifier:*  
DataGrid-01-D1.2-0112-0-3

*Date:* 14/09/2001

---

execution. When a requested operation, either a job submission or a job removal, has completed, the RB is notified asynchronously about the result.

The JSS is also responsible to resubmit failed jobs, if the failure depends on the Grid infrastructure and not on the job itself. For this an appropriate job monitoring is needed. The monitoring cannot be based entirely on a callback mechanism from the CE, e.g. because the CE itself can fail or the network connection between the CE and the JSS can be interrupted. That is why the JSS also polls resources and jobs to check for their status.

To support the monitoring the JSS keeps an internal registry of submitted jobs. Each of them is identified by a `jss_jobId` (see Section 3.2). The registry is persistent in order to be able to recover from JSS failures.

#### **4.4 RECOVERY FROM FAILURES**

The proposed architecture for the Workload Management System is resilient to many error conditions, since mechanisms to recover from many failures that could occur in a Grid system have been considered.

While it is not possible to address failures originated by jobs (e.g. crashes of running jobs due to bugs in the job code), which are simply reported to the submitting user (e.g. the standard output and the standard error files can be included in the output sandbox and sent back to the submitting machine), most of the possible errors generated by the Grid infrastructure have been taken into account.

Local failures (e.g. a crash of the machine running a specific service) are addressed by saving all the relevant information persistently. For example both the Job Submission Service and the Resource Broker maintain queues where all the important information for all submitted jobs is stored persistently, so, in case of failure, the system can recover by reading this information.

Problems could also occur if the JSS is not able to submit a job to a specific Computing Element chosen by the Resource Broker. This can happen because the chosen resource is not “valid” anymore, for example because of changes in the authorization policies, so that the considered user cannot submit jobs anymore to that resource, or because of a problem in the network infrastructure, so that the remote resource cannot be contacted anymore. In these cases the job is sent back to the RB, which can try to find another suitable resource for the job.

When a job is submitted to a specific Computing Element, the Job Submission Service must take care of its execution. Many error conditions can occur, such as the failure of the CE/JSS interface, the failure of the network connection between the JSS and the CE, the crash of the job. To detect and manage these remote failures the JSS cannot rely only on callbacks actively sent by CEs, since these could be lost in case of failure, but it must periodically probe all the remote CEs where jobs have been submitted.



# DEFINITION OF ARCHITECTURE, TECHNICAL PLAN AND EVALUATION CRITERIA FOR SCHEDULING, RESOURCE MANAGEMENT, SECURITY AND JOB DESCRIPTION

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

## 5 THE RESOURCE BROKER

The Resource Broker (RB) [R3] is the core component of the Workload Management System. Its main task is to find a Computing Element that best matches the requirements and preferences of a submitted job, considering also the current distribution of load on the Grid. Once a suitable Computing Element is found, the job is passed to the Job Submission Service for the actual submission.

Additionally the Resource Broker allows cancelling a job and retrieving the output once a job has completed.

Another important task performed by the Resource Broker is the maintenance of a consistent and persistent database concerning the information about handled jobs (i.e. status, JDL) local to the Resource Broker itself.

In order to achieve its goal the Resource Broker interacts with other Grid services, in particular it obtains information about data location from the data replication services and information about the status of Grid resources from the information services [R11]. The latter information is cached in a dedicated server that converts it to a “JDL” view for symmetric matching to the user requests.

Determining the hierarchy and distribution of RB instances is an open issue. On one extreme a single, central RB for the whole of DataGrid would provide optimal resource allocation fairness, but poses scalability problems that are very hard to address with commodity hardware. On the other extreme one could imagine a dedicated RB (some sort of *Grid-browser*) for each end-user, giving freedom in the choice of resource, but also originating races for distributed resources that cannot be fairly arbitrated. The solution we plan to explore is an intermediate one, providing a RB for each *Virtual Organization* participating in DataGrid. In order to be able to experiment different choices, however, the RB is designed to work both at the “personal” and at the VO level.

### 5.1 EXTERNAL INTERFACES

The Resource Broker offers one interface to external Grid components and one interface to be used only internally within the WMS, in particular by the JSS. The corresponding APIs are presented in Section 5.1.1 and Section 5.1.2 respectively.

#### 5.1.1 Job Control API

The job control API consists of the functions that allow a client application, for example a User Interface, to:

- Submit a job.
- Cancel a job.
- Retrieve the output produced by a job.

The API contains the following methods:

`job_submit()` pass a job to the RB, which then finds a suitable CE to run it. The job is then transferred to the JSS for the actual submission.

`job_cancel()` kill a previously submitted job.

`job_cancel_all()` kill all previously submitted jobs belonging to a certain user.

`job_get_output_sandbox()` retrieve the output sandbox of a completed job.



# DEFINITION OF ARCHITECTURE, TECHNICAL PLAN AND EVALUATION CRITERIA FOR SCHEDULING, RESOURCE MANAGEMENT, SECURITY AND JOB DESCRIPTION

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

`list_job_matches()` pass a job to the RB and return a list of suitable CEs to run it.  
The RB performs only the matchmaking without actually submitting the job.

At submission time a job is expressed in the Job Description Language (see Section 6.1). In all the other cases a job is identified by its `dg_jobId` (see Section 3.2).

## 5.1.2 Notification API

The notification API represents the interface of the RB towards the JSS and is used to asynchronously communicate to the RB relevant events concerning a job so that the RB can take appropriate actions.

The API contains only the method `notify()`. The notification reason passed as a parameter to the function can assume one of the following values: `JOB_ACCEPTED`, `JOB_REFUSED`, `JOB_ABORTED`, `JOB_DONE`, `JOB_CANCELLED`, and `JOB_OUTPUT_TRANSFERRED`.

## 5.2 DEPENDENCIES

As shown in Figure 1-1 a Resource Broker depends on services provided by other WMS components:

- The Job Submission Service provides the actual submission of a job to a Computing Element and its monitoring.
- The Logging and Bookkeeping Service provides persistent storage of job status information. It also allows logging and storing persistently information related to the RB itself.

Moreover the RB depends on several other Grid services to accomplish its scheduling goals:

- The Replica Management services provide support for resolution of logical names, replica location and replica creation. The Resource Broker interacts with the Replica Management services in order to find out the most suitable Computing Element taking into account the Storage Elements where both input data sets are physically stored and output data sets should be staged on completion of job execution.
- The Information System provides information about characteristics and status of Grid resources.

## 5.3 EVALUATION CRITERIA FOR SCHEDULING

Selecting the “best” resource that a submitted job can use, typically a Computing Element, is a complex task. A summary of the typical steps that are used by a human user to select an appropriate resource can be found in [R9]. The first efforts in automating the selection procedure in the Workload Management System are along the same lines.

Many factors can affect the scheduling decision and their importance may even differ depending on the job. These factors include:

- Location of input and output data.
- Authorisation to use a resource.
- Allocation of resources to the user or to the groups the user belongs to.
- Requirements and preferences of the user.
- Status of the available resources.

The choice for the right balance of information on which the scheduling decisions can be made is an open problem. One of the major goals in the DataGrid project is to make the most appropriate or

clever strategies emerge from the experiences of the Applications end users with the first project releases. The impact on the RB architecture is that the scheduling procedure has to be modular and easily replaceable with support for different scheduling strategies at the same time.

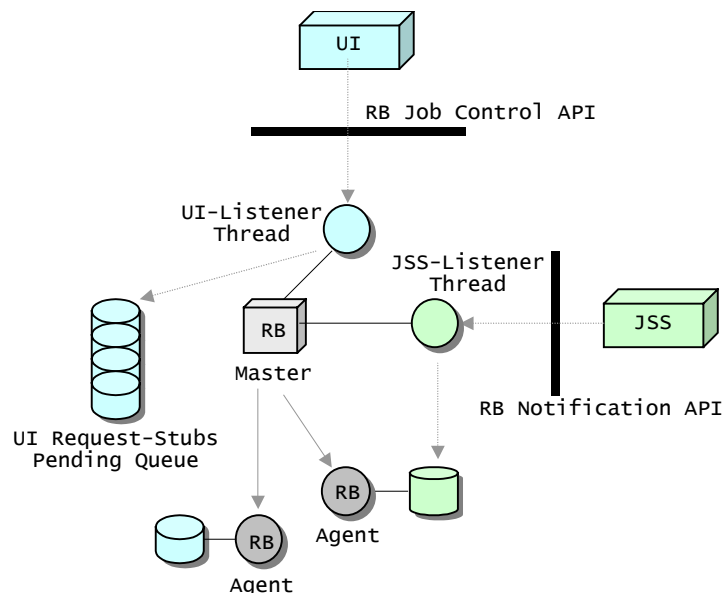
The information that is sent by the RB to the LB Service needs to allow the measurement and comparison of scheduling efficiency estimators such as the ones in this (non exhaustive) listing:

- The permanence time of individual jobs in the RB queue.
- The amount of resources that are available in the system for a given job.
- The amount of job submissions that abort due to incorrect resource information (including authorisation and access policy information).
- The frequency of job aborts due to insufficient computing resources (and the amount of computing resources that are consumed in this case).
- The frequency of job resubmissions by the RB and the JSS.
- The amount of jobs that are aborted by the end users.

#### 5.4 INTERNAL DESIGN

The internal design of the Resource Broker is based on a multithreaded client-server model: client applications, such as the User Interface and the Job Submission Service, are provided with the above mentioned APIs to communicate with the RB server. The model is distributed in the sense that clients and servers may be running on different hosts.

The following figure depicts the Resource Broker process model, giving a simplified view of the main executed threads.



**Figure 5-1. The Resource Broker process model.**

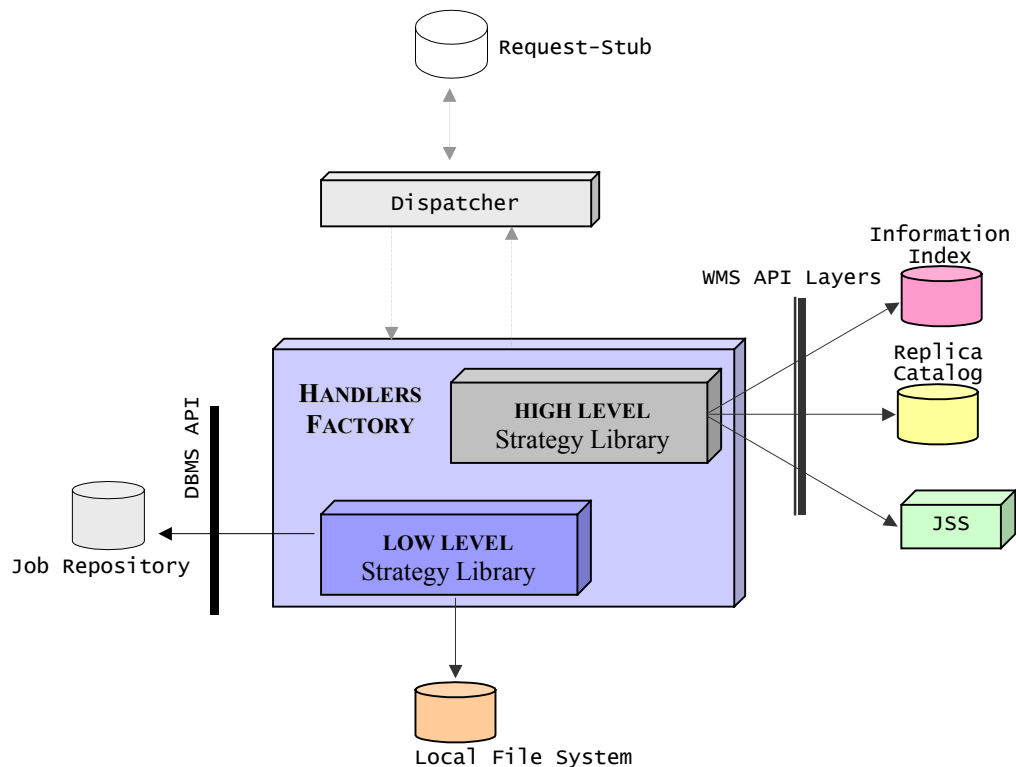
Within the Resource Broker a master process (*RB-Master*) spawns two main server threads: *UI-Listener* and *JSS-Listener*.

The UI-Listener listens continuously on a well-known port for connection requests from clients (UI). Once a connection from a client is established, it creates an UI Request-Stub to service that connection. UI Request-Stubs are immediately sent to a queue local to the RB-Master and the UI-Listener goes back to listen for more connections. If there are free resources (i.e. the thread-pool internal to the RB-Master is not empty) the RB-Master spawns a thread (*RB-Agent*) for servicing the first queued request.

The JSS-Listener listens on a different port to asynchronous notification events coming from the Job Submission Services, dispatches and executes them immediately attaching slave threads.

According to the previous figure, it should be clear that the RB-Agent is the entity responsible to perform the actual execution of those tasks needed to accomplish a given request issued by either the User Interface or the Job Submission Services. Moreover, the RB-Agent attached to a given UI Request-Stub is responsible for returning results to the client, if so required.

The following figure gives a simplified view of the internal design of an RB-Agent:



**Figure 5-2. The Resource Broker Agent internal design.**

The Dispatcher receives/sends data from/to the client Request Stub and dispatches them to the Handlers Factory.

The Handlers Factory runs within an RB-Agent. When a request is received, the appropriate software modules (also called *strategy components*) are dynamically loaded, configured and run. The intent is to define a whole family of algorithms to perform basic operations, encapsulate them in the strategy components, and make them easily interchangeable.



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

The high-level strategy components include those that implement the interaction with external services, such as the Job Submission Service, the Logging and Bookkeeping Service, the Replica Catalog, the Information System. Interaction with these services is needed in order to perform operations such as the resolution of logical file names into physical file names, the collection of information about available resources and their status, matching job requirements against available resources.

In particular information such as policies to access resources, characteristics and status of resources and availability of the required application environment is published in the Information System. The query of the Information System by the RB-Agent is performed in different steps: first an Information Index (II), which is just a cache sitting by the RB, is queried, to get a set of possible candidates; then the search is refined, querying directly these candidate resources, to get more valuable and up-to-date information. In order to benefit from the symmetry properties of the Job Description Language (JDL, see Section 6.1) the resource characteristics are also translated into JDL format.

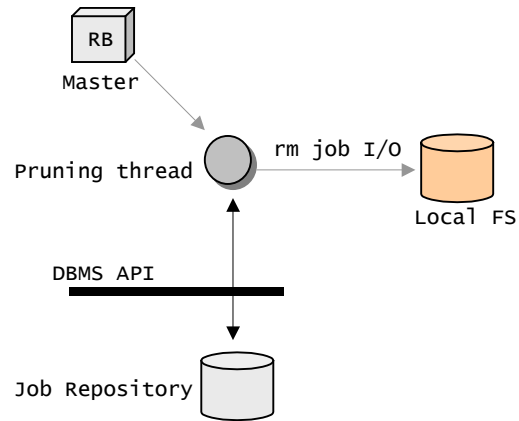
The low-level strategy components address internal operations, such as basic I/O operations for moving input and output sandboxes, collection of information about active jobs and management of a Job Repository, where job information is kept persistent and up-to-date.

As mentioned before, the Job Repository contains information about all jobs handled by the Resource Broker. The RB queries the Job Repository to properly serve client requests originating by other WMS components.

Here is the list of valid job status values in the Resource Broker:

- RBJS\_WAITING      the Resource Broker received the job submission request from the UI, but either there are no available local resources to execute that request (for example it is not possible to spawn a new thread to handle the request) implying that the request is still waiting in the Resource Broker internal queue or the matching process for such a job has not been completed yet.
- RBJS\_READY        the job is ready to be submitted to the Job Submission Service for further processing (i.e. actual submission).
- RBJS\_ACCEPTED    the Job Submission Service has correctly accepted the job.
- RBJS\_REFUSED     the Job Submission Service has refused the job. From that point, after a given span of time during which no further status transitions occur for that job, the Resource Broker shall change the state to RBJS\_ABORTED forcing the removal of the job.
- RBJS\_DONE         the job has successfully completed its execution and its output sandbox has been transferred back to the Resource Broker.
- RBJS\_ABORTED     the job has been terminated.

It should be reminded that once a job terminates its execution, the output sandbox is copied back to the storage space of the RB machine, waiting to be retrieved following the reception of a request from the client. Additionally, at submission time the input sandbox of the job is transferred to the Resource Broker. As shown in Figure 5-3, the RB-Master periodically executes a pruning thread to clear such storage space in order to protect against misbehaving users or systems.



**Figure 5-3. Resource Broker pruning thread.**

---

## 6 THE USER INTERFACE

The User Interface (UI) [R4] is the component of the Workload Management System that allows a user to access the functionality offered by the Grid, in particular the possibility to submit jobs and retrieve their output.

The UI is based on two elements:

1. A language to describe characteristics, requirements, preferences of a job.
2. A set of commands to manage jobs on the Grid.

Some general principles have been followed for the design of the User Interface:

- Consistency: commands and/or menus should have the same format, parameters should be passed to all commands in the same way and command punctuation should be similar.
- Avoid modes: a mode could be when a user needs to interrupt and rollback any current activity in order to be able to proceed with something else or when the same action has different results in different situations.
- Transparency: the user interaction with the system should be as simple as possible.
- Error recovery mechanisms: the interface should be designed to minimise user mistakes and provide facilities for recovering from possible mistakes.
- User guidance: the interface should have built-in “help” facilities.

Running in a Grid environment, the User Interface should also be:

- Portable: it should run on a potentially large number of platforms.
- Configurable: it should allow easy configuration and reconfiguration.
- Secure: it should support secure (i.e. properly authenticated and authorised) transactions with other Grid Services.
- Light weighted: it should not require a significant amount of local resources, such as disk space or CPU cycles, both for running and installing.

### 6.1 THE JOB DESCRIPTION LANGUAGE

The main properties that have guided the choice of a language to properly describe the characteristics of a job are:

- Semi-structured data model: no specific schema is required.
- Symmetry: all entities in the Grid, in particular applications and computing resources, should be expressible in the same language. This makes it easier for example to perform a match between a job description and resource descriptions when looking for a suitable Computing Element where to run a job.
- Simplicity: the description language should be simple both syntactically and semantically.

The *classified advertisements* (*ClassAds*) defined by the Condor project [R5] have been adopted as Job Description Language (JDL) [R6] because they satisfy all the mentioned properties.

A ClassAd is a set of named expressions called attributes. An expression contains literals and attribute references composed with operators. In particular ClassAds can be nested, i.e. a ClassAd can be itself an expression, allowing the representation of aggregates of resources or jobs.



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

When describing a job using the JDL some of the attribute names are predefined and have a special meaning for the components of the Workload Management System. Moreover some of them are mandatory, while others are optional. The set of predefined attributes include, but it is not limited to:

Executable	name of the executable to run on the worker node.
Arguments	string containing all the command line arguments to pass to the executable.
StdInput	standard input file.
StdOutput	standard output file.
StdError	standard error file.
InputSandbox	“short” list of “small” files to transfer in the job working directory. Wildcards are allowed and are expanded by the User Interface.
OutputSandbox	“short” list of “small” files to transfer from the job working directory. Wildcards are allowed and are expanded on the Computing Element at the end of job execution.
InputData	list of input data files, usually expressed in a logical way.
ReplicaCatalog	Replica Catalog to use to convert logical file names to physical file names.
DataAccessProtocol	list of protocols the job is able to use to access files.
OutputSE	the Storage Element where the job will store output files.
Requirements	expression representing a set of requirements to be matched against available Grid resources.
Rank	expression representing preferences to be matched against suitable Grid resources.

For example the following expression in JDL:

```
[
  Executable          = "simula";
  Arguments           = "1 2 3";
  StdInput            = "simula.config";
  StdOutput           = "simula.out";
  StdError            = "simula.err";
  InputSandbox        = { "/home/joe/simula.config",
                          "/usr/local/bin/simula" };
  OutputSandbox       = { "simula.out", "simula.err", "core" };
  InputData           = "LF:test367-2";
  ReplicaCatalog      = "ldap://pcrc.cern.ch:2010/rc=ReplicaCatalog,
dc=pcrc, dc=cern, dc=ch";
  DataAccessProtocol  = { "file", "gridftp" };
  OutputSE            = "lxde01.pd.infn.it";
  Requirements        = other.Architecture == "INTEL" &&
                          other.OpSys == "LINUX";
  Rank                = other.AverageSI00;
```



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

Doc. Identifier:  
DataGrid-01-D1.2-0112-0-3

Date: 14/09/2001

]

says that:

- The executable to run on the Computing Element node is `simula`, which also takes as command line arguments 1 2 3.
- The standard input is the file `simula.config`.
- The files `/usr/local/bin/simula` and `/home/joe/simula.config` reside on the submitting machine and need to be transferred to the remote Computing Element as part of the input sandbox.
- Standard output and standard error are `simula.out` and `simula.err`. Since they are produced on the Computing Element they need to be transferred on the submitting machine at the end of the job. That's why they are included in the output sandbox.
- The output sandbox includes also a `core` file in case this is produced.
- The input data is contained in a file whose logical name is `test367-2`.
- The logical file name can be resolved in a physical file name using the replica catalog available at the URL "ldap://pcrc.cern.ch:2010/rc=ReplicaCatalog, dc=pcrc, dc=cern, dc=ch".
- The job is able to access files either via the `file` or `gridftp` protocols.
- The job will put the output it produces to the Storage Element whose contact point is `lxde01.pd.infn.it`.
- The job requires to be run only on machines with an Intel architecture running the Linux operating system (`other` represents the other entity when the match-making is performed).
- The job prefers to run on the Computing Element with the highest SpecInt2000 value. The AverageSI00 value is published by each Computing Element through the Information System.

## 6.2 FUNCTIONAL MODEL

The functionality offered by the User Interface includes:

- Job control.
- Job monitoring.

The user interface is expressed as a set of commands [R7].

### 6.2.1 Job Control

The User Interface allows a user to submit a job, to cancel a job, to list resources matching a given job description, to retrieve the output sandbox of a job.

To address the above functionality the following commands are provided:

<code>dg-job-submit</code>	submit a job to be run on Grid resources.
<code>dg-job-cancel</code>	kill a previously submitted job.
<code>dg-list-job-match</code>	list the resources that match a job requirements. It provides the end user with the capability to manually override the decisions that the WMS may make.



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

*Doc. Identifier:*  
DataGrid-01-D1.2-0112-0-3

*Date:* 14/09/2001

---

`dg-get-job-output` retrieve the standard output, the standard error and the output sandbox generated by a completed job.

### **6.2.2 Job Monitoring**

The User Interface allows a user to check the status of a job, including the bookkeeping information, from its submission to the time it is removed from the bookkeeping database. In other words the user can follow the job while it moves along the state machine presented in Section 3.1. The command to do so is `dg-job-status`.

Moreover the User Interface provides access to some logging information related to the job and stored by the Logging and Bookkeeping Service. The command to do so is `dg-get-logging-info`.

### **6.3 DEPENDENCIES**

The User Interface depends on two other Workload Management System components:

- The Resource Broker provides support for the job control functionality of the UI.
- The Logging and Bookkeeping Service provides support for the job monitoring functionality of the UI.



**DEFINITION OF ARCHITECTURE,  
TECHNICAL PLAN AND EVALUATION  
CRITERIA FOR SCHEDULING,  
RESOURCE MANAGEMENT,  
SECURITY AND JOB DESCRIPTION**

*Doc. Identifier:*  
DataGrid-01-D1.2-0112-0-3

*Date:* 14/09/2001

---

## **7 THE SECURITY FRAMEWORK**

For the grid to be an effective framework for largely distributed computation, users, user processes and grid services must work in a secure environment, where all interactions between two entities are properly authenticated and, where needed, encrypted, and all accesses to resources are properly authorised.

The secure environment within the DataGrid project is based on a Public Key Infrastructure (PKI): each user and each service owns a credential, consisting of a pair of keys, one private and one public, whose main purpose is to prove their identity. The user or service identity and their public key are included in a X.509 certificate signed by a trusted Certification Authority (CA), whose purpose is to guarantee the association between that public key and its owner.

A PKI only provides simply an authentication and cryptographic mechanisms, while a security framework should also include support for:

- An authorisation mechanism to access all types of resources.
- Resource usage accounting.
- Auditing.
- User credential delegation.
- User credential renewal.

Several solutions can be envisaged for each of the above issues, which are all subject of intense research.

### **7.1 SECURITY IN THE WORKLOAD MANAGEMENT SYSTEM**

The proper security infrastructure, along the lines described in the previous section, needs to be established for the Workload Management System [R8].

Within the WMS the User Interface, the Resource Broker, the Job Submission Service and the Computing Element need a delegated user credential. Delegation of a user credential allows another entity to act on behalf of that user. A typical example is a job the user has submitted and that should access other resources, e.g. data, belonging to that user. Another reason to have support for credential delegation is for limiting the risk that the original credential is compromised. On the other hand the possibility that someone else can act on behalf of the user opens a security problem, whose solution is to delegate only limited credentials, limited in time and/or capability.

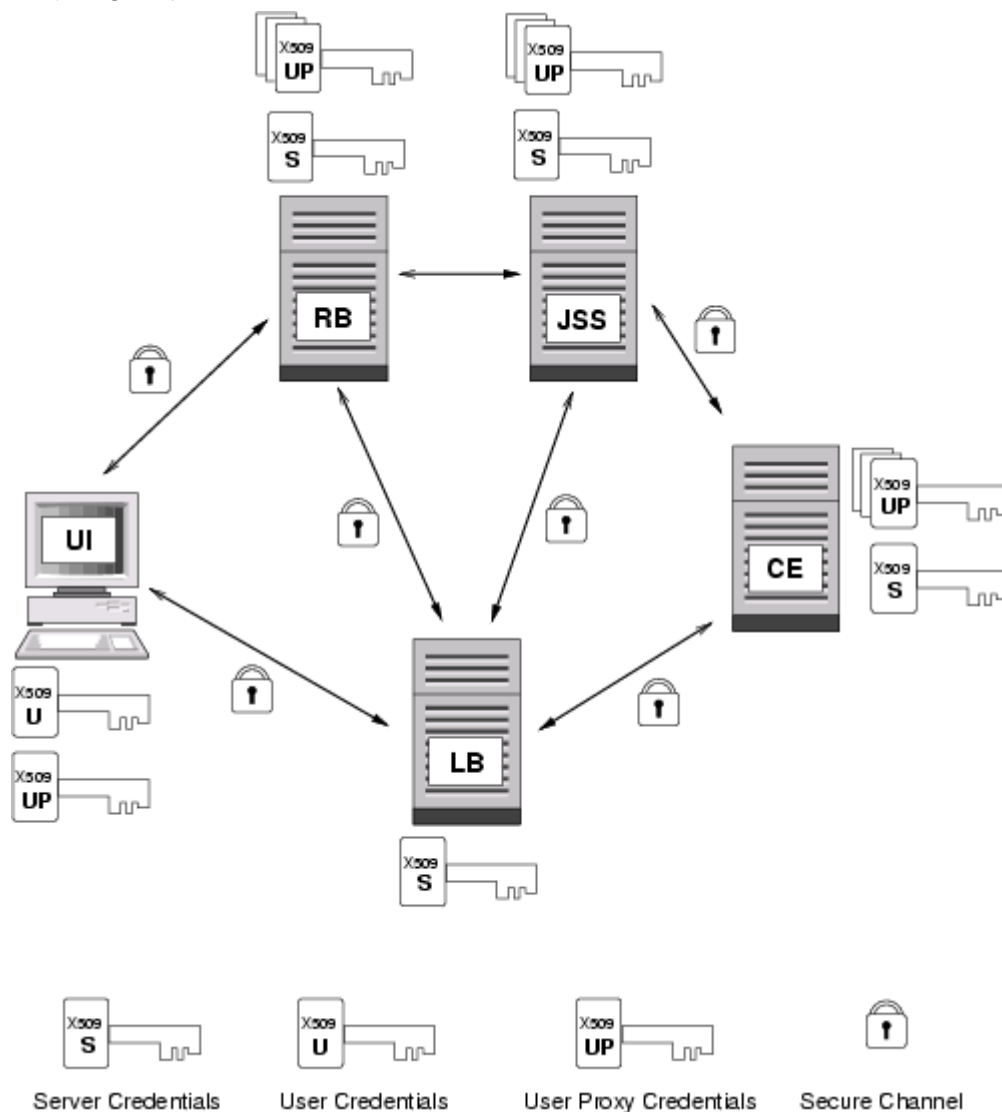
However, in certain scenarios, e.g. jobs running for a long time, a limited credential is not enough, so a mechanism to renew an about-to-expire delegated credential is needed. The exact functionality of such mechanism and the way it should work are not clear yet; nevertheless the basic idea is to have an additional service that is delegated by the user to provide delegated credentials to the entities that will request them. Such repository must always be under the control of the user.

The UI has a delegated user credential to limit the risk of compromising the original credential in the hands of the user. All the others need a delegated credential to act on behalf of the user in several occasions, including:

- Accessing information in the Information Service.
- Writing and reading information to and from the Logging and Bookkeeping Database.
- Submitting jobs to Computing Elements.

All interactions between WMS components, especially those that are network-separated, will be mutually authenticated. Depending on the specific interaction, an entity authenticates itself to the other peer using either its own credential or a delegated user credential or both. For example when the User Interface passes a job to the Resource Broker, the UI authenticates using a delegated user credential whereas the RB uses its own service credential. Instead when the RB logs an event to the Logging and Bookkeeping Service, the RB authenticates using both its own service credential and a delegated user credential, whereas the LB Service uses only its own server credential.

Figure 7-1 summarises the security framework that will be used among the components that belong to the Workload Management System. All network communication is properly authenticated using server credentials, (delegated) user credentials or both.



**Figure 7-1.** The components of the Workload Management System communicate over secure network channels. Authentication is established using server credentials, delegated user credentials or both.