

DataGrid

JOB SUBMISSION SERVICE ARCHITECTURE AND APIS

WP01: Grid Work Scheduling

Document identifier:	DataGrid-xx-TYP-nnnn-0_0
Date:	26/07/2001
Work package:	WP01: Grid Work Scheduling
Partner(s):	INFN
Lead Partner:	INFN
Document status	DRAFT

Deliverable identifier:

Abstract:

The Job Submission Service (JSS) architecture is presented, and the JSS client API's are specified.



Delivery Slip

	Name	Partner	Date	Signature
From	Massimo Sgaravatto Marco Verlato	INFN Padova		
Verified by				

Document Log

Issue	Date	Comment	Author
0_0	26/07/2001	First draft	Massimo Sgaravatto Marco Verlato

Document Change Record

Issue	Item	Reason for Change

Files

Software Products	User files
Microsoft Word 2000	jss-arch.doc
Adobe Acrobat 5.0	jss-arch.pdf



CONTENT

1. INTRODUCTION.....	4
1.1. OBJECTIVES OF THIS DOCUMENT.....	4
1.2. TERMINOLOGY.....	4
2. OVERVIEW OF THE JOB SUBMISSION SERVICE.....	5
2.1. JOB SUBMISSION	5
2.2. JOB REMOVAL.....	5
2.3. JOB MONITORING.....	5
3. INTERACTIONS WITH THE OTHER COMPONENTS OF THE WMS.....	7
3.1. INTERACTIONS WITH RB.....	7
3.2. INTERACTIONS WITH COMPUTING ELEMENTS.....	7
3.3. INTERACTIONS WITH THE LOGGING AND BOOKKEPING SERVICE	8
4. IMPLEMENTATION OF THE JOB SUBMISSION SERVICE	9
5. JOB SUBMISSION SERVICE CLIENT APIS.....	10
6. REFERENCES.....	14



1. INTRODUCTION

1.1. Objectives of this document

The purpose of this document is to describe the architecture of the Job Submission Service (JSS). It also specifies the API's used by the client (i.e. the Resource Broker).

1.2. Terminology

Definitions

Glossary

CE	Computing Element
JDL	Job Description Language
JSS	Job Submission Service
L&B	Logging and Bookkeeping
RB	Resource Broker
UI	User Interface
WMS	Workload Management System



2. OVERVIEW OF THE JOB SUBMISSION SERVICE

The Job Submission Service (JSS) is the component of the workload management system (WMS) responsible for the actual job management operations, issued on request of the Resource Broker.

In particular the Job Submission Service is responsible to manage the job submission and job removal requests, interacting with the Computing Elements (CE's). It is also responsible to keep tracks of the submitted jobs (until their completion) and to notify the Resource Broker and/or the Logging and Bookkeeping (L&B) Service when significant events occur.

In the planned workload management system architecture, the JSS works in pair with the Resource Broker, and therefore a JSS will be deployed for each installed RB.

2.1. JOB SUBMISSION

When the JSS receives a job submission request from the Resource Broker, it receives the "original" JDL expression sent to the Resource Broker by the User Interface (UI), "augmented" with the Computing Element choice.

As first step the JSS builds a "wrapper" of the user job, responsible to create the appropriate execution environment on the remote worker node. This includes the staging of the input sandbox (i.e. the input files needed to run the job) in the worker node, and the retrieving of the output sandbox (i.e. the output files that the user wants back in her desktop) at job completion.

The Job Submission Service assigns a *jss-jobId* identifier to this wrapper job, and then inserts it in a local queue, where the relevant information for all submitted jobs is stored persistently. The JSS then performs the actual job submission to the chosen remote Computing Element, on behalf of the Grid user who required the job submission (therefore using user's delegated credential).

If this submission can't be performed, for example because the Resource Broker has chose a resource that is not available anymore, the job is removed from the JSS job queue, and the RB is notified about the problem.

2.2. JOB REMOVAL

For what concerning job removals, when the Resource Broker requires the removal of a job, the JSS receives the *dg-jobId* identifier of this job. The JSS finds the correspondent *jss-jobId* identifier, contacts the Computing Element where this job has been previously submitted requiring its cancellation, and when the job has been removed from the Computing Element, it is then removed also from the queue of the JSS. As last step, the Resource Broker is notified.

2.3. JOB MONITORING

The Job Submission Service is also responsible to monitor and keep tracks of the submitted jobs, until their completion, notifying the Logging and Bookkeeping Service and/or the Resource Broker when significant events occur.

It is also in charge to resubmit failed jobs, to recover from remote or local failures, to handle error conditions.

The Job Submission Service must also address the problem of credential renewal: since valid user's credentials are needed for all the job lifetime, and since it is not safe to give long-term credentials to



**JOB SUBMISSION SERVICE
ARCHITECTURE AND APIS**

Doc. Identifier:
DataGrid-xx-TYP-nnnn-0_0

Date: 26/07/2001

the job (because a large security breach window would be created), short term credentials must be considered. The JSS must therefore periodically analyse the credentials for all users with currently queued jobs, and if a user's credentials are about to expire, they must be renewed. Moreover these refreshed credentials must be forwarded to the correspondent Computing Element (unless a mechanism for credential renewal is implemented in the Computing Element as well).



3. INTERACTIONS WITH THE OTHER COMPONENTS OF THE WMS

3.1. INTERACTIONS WITH RB

The communication between the Resource Broker and the Job Submission Service will be based on the client/server model.

The JSS will have a main daemon listening continuously on a well-known port for requests from the client (i.e. the Resource Broker). Once a request (job submission or job removal) is received, a thread servicing this request is executed.

The client/server model will be considered also for the asynchronous notifications from the JSS to the RB (issued as “answers” to the requests issued by the RB, and to notify the Resource Broker when a job has completed its execution).

3.2. INTERACTIONS WITH COMPUTING ELEMENTS

The Job Submission Service must interact with the Computing Elements to request the submission/removal of jobs, to query the status of previously submitted jobs, to be informed on the completion of jobs, etc...

All these interactions must be properly authenticated.

In order to implement a system usable for real applications and in real production environments, the Job Submission Service must be reliable, fault tolerant, and resilient to the different types of failures that could occur in a distributed Grid environment.

First of all the system must be resilient to local failures (for example a crash of the machine running the Job Submission Service): this can be managed by maintaining a queue where all relevant information for all submitted jobs is stored persistently, so in case of failure, the system can recover reading this information.

Besides being able to recover from local failures, the JSS must be able to “manage” other different error conditions, such as failures in the “entities” of the Computing Elements acting as “gateways” from the JSS to the remote resources, failures in the network between the JSS and the remote Computing Elements, etc... This means, for example, that the JSS can’t simply rely only on callbacks sent from the Computing Elements, but instead it must periodically probe the remote Computing Elements where jobs have been submitted.

An other important issue that must be considered is the scalability of the system: the JSS must be able to manage many (hundreds/thousands) of jobs, with possible different characteristics and requirements, submitted by many different Grid users.

Accessing the remote Computing Elements for these operations is not a trivial task, since the various Computing Elements, distributed in possible different administrative domains, can be very different and can rely on different mechanisms, policies, implementations: they can be different in hardware, they can run different operating systems, they can be managed by different local resource management systems, they can use different authentication and authorization mechanisms, etc...

These issues will be addressed relying on standard protocols: “forcing” the Computing Elements to use standard protocols for authentication, resource discovery and management, it will be easier to



implement “generic” mechanisms and solutions to support the interactions between the JSS and the different Computing Elements.

3.3. INTERACTIONS WITH THE LOGGING AND BOOKKEEPING SERVICE

Among the various components of the workload management system, also the JSS is responsible to “push” some events to the Logging and Bookkeeping Service.

In particular the JSS must notify the L&B service the following events:

- *JobAcceptedEvent* (when a job has been accepted by the JSS)
- *JobRefusedEvent* (when the JSS is not able to accept a job received by the RB)
- *JobTransferEvent* (when a job is transferred from the JSS to the CE, that is when the job state transition *waiting* → *ready* occurs)
- *JobAbortEvent* (when the JSS stops the job processing; this can occurs on request of the Resource Broker, or this can be triggered by the JSS itself, if a “bad” resource has been chosen by the Resource Broker, and therefore the computation on this Computing Element can’t be performed)
- *ComponentStatusEvent* (when the JSS announces its status changes)

The JSS uses the logging (producer) API to pass this information to the L&B service.



4. IMPLEMENTATION OF THE JOB SUBMISSION SERVICE

The implementation of the Job Submission Service will be performed in an incremental fashion, following a fast prototyping approach.

The first steps will be integrating existing tools and services that could provide some of the required functionalities, performing the required customisations, and implementing the missing functionalities.

For example, in the first prototype, as standard protocols for security and resource management, the protocols defined by the Globus toolkit [Globus] will be exploited in the interactions between the JSS and the Computing Elements. In particular the authentication and authorization mechanisms of Globus GSI (Grid Security Infrastructure) [GSI], and the Grid Resource Allocation and Management (GRAM) protocol [GRAM] will be exploited.

For the Job Submission Service itself, as first steps the Condor-G system [CondorG], implementing a reliable submission system to Globus resources, will be investigated and exploited.



5. JOB SUBMISSION SERVICE CLIENT APIS

JSS(class constructor)

SYNOPSIS

```
#include "jssclient.h"  
JSS::JSS(int port);
```

DESCRIPTION

Instantiates a JSS server for the specified port.

- *Port* – The port number of the JSS. If the input value is an empty string, the environment variable *JSSPort* is considered, if defined, otherwise a default port number (8881) is used.

RETURN VALUES

None

Errors

None



error_message

SYNOPSIS

```
#include "jssclient.h"  
const string& JSS::error_message();
```

DESCRIPTION

On encountering an error on a client call, the *status* member value is updated with an error code (enumeration type).

error_message() is invoked by the caller to get a text string describing this error.

RETURN VALUES

Returns the string containing the error message.

Returns an empty string if there is no error message.

Errors

None



job_submit

SYNOPSIS

```
#include "jssclient.h"
void JSS::job_submit(const string& JDLexpression, const string& InputSandboxDir, const string&
OutputSandboxDir);
```

DESCRIPTION

Submits a request to the JSS for submitting a job.

- *JDLExpression* – the JDL expression received by the RB from the UI, augmented by the RB with the attribute *GlobusResourceContactString*, in case with the attribute *QueueName*, and with the attribute *LocalDiskPathname*
- *InputSandboxDir* – the absolute pathname in the RB/JSS file system where the files specified as InputSandbox are stored, assuming that this directory contains all and only these files.
- *OutputSandboxDir* – the absolute pathname in the RB/JSS file system where the files specified as OutputSandbox must be copied

RETURN VALUES

None

Errors

JSS_CONNECTION_ERROR in the *status* member value of the JSS instance.



job_cancel

SYNOPSIS

```
#include "jssclient.h"  
void JSS::job_cancel(const list<string>& JobIdList);
```

DESCRIPTION

Submit a request to the JSS for canceling one or more jobs.

- *JobIdList* – the list of dg-jobids to be cancelled.

RETURN VALUES

None

Errors

JSS_CONNECTION_ERROR in the status member value of the JSS instance.



6. REFERENCES

[Globus] Foster, I. and Kesselman, C., "Globus: A Toolkit-Based Grid Architecture", Foster, I. and Kesselman, C. eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, pp. 259-278.

[GSI] Foster, I., Kesselman, C., Tsudik, G., and Tuecke, S., "A Security Architecture for Computational Grids", *ACM Conference on Computers and Security*, 1998, pp. 83-91.

[GRAM] Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., Tuecke, S., "A Resource Management Architecture for Metacomputing Systems", *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[CondorG] Frey, J., Tannenbaum, T., Foster, I., Livny, M., Tuecke, S., "*Condor-G: A Computation Management Agent for Multi-Institutional Grids*", Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10), 2001