Grid Job Management Framework

Author: P-O Östberg, p-o@cs.umu.se
Contents

1 Abstract 1

2 About this document 1

3 About GIRD 1

4 Background & Motivation 2

5 Design criteria 3
  5.1 Small, single-purpose components 3
  5.2 Standards conformance 3
  5.3 Middleware abstraction 3
  5.4 Ease of use 4
  5.5 Software versatility 4
  5.6 Layers of functionality 5

6 Architecture 7
  6.1 Task Group Management Service (TGMS) 7
  6.2 Task Management Service (TMS) 7
  6.3 Brokering & Submission Service (BSS) 7
  6.4 Job Control Service (JCS) 7
  6.5 Resource Selection Service (RSS) 8
  6.6 Log Accessor Service (LAS) 8
  6.7 Dynamic Configuration Service (DCS) 8
  6.8 Data Management Service (DMS) 9
  6.9 Client API 9

7 Use cases 10
  7.1 Job control 10
  7.2 Basic job submission 11
  7.3 Reliable job submission 11
  7.4 Reliable submission of multiple jobs 12
  7.5 Tracking execution progress 12
  7.6 Interactive jobs 12

8 Terminology 13

9 State models 14
  9.1 Job states 14
  9.2 Task states 15
  9.3 Task group states 16

10 Development history 17

11 Future work 18
1 Abstract

The Grid Job Management Framework (GJMF) is a multi-tier service framework for grid job submission, control and management. The goal of the GJMF is to offer a generic and easy-to-use platform for job management in collaborative scientific grid computing environments. Based on existing and emerging grid standards, the framework is comprised of a set of grid services offering functionality at multiple levels for job submission, control and management. The granularity of job control offered by the framework ranges from a top-level fire-and-forget perspective of batches of jobs to individual job micro-management. The GJMF is implemented using the Globus Toolkit version 4 (GT4) [10].

2 About this document

The aim of this document is to provide an introduction to the Grid Job Management Framework, its components and intended use cases. The target audience of this document (grid service developers familiar with the GT4 platform and / or other grid environments) will find it useful as a first introduction to the GJMF as well as a source of insight into the design rationales of the framework.

3 About GIRD

The Grid Infrastructure Research & Development (GIRD) [6] is a multi-project that develops standards-based infrastructure components for grid resource and project management. The long-term goal is to provide grid-enabled and grid-enabling tools for scientific computing applications.

The projects embraces service-oriented principles and leverages proposed grid and web services standards (WSRF, JSDL, GLUE, WSAG, OGSA, OGSA-EMS, etc) for improved cross-middleware interoperability, integration and customizability.

GIRD is located at the Department of Computing Science and affiliated with the High Performance Computing Center North (HPC2N) of Umeå University, Sweden.
4 Background & Motivation

The Grid Job Management Framework is part of a continuous effort to develop efficient job submission and management infrastructure components for collaborative scientific grid environments. The basic vision of the framework is to construct a service oriented architecture where jobs transparently can be submitted to grid-enabled machines and clusters, e.g. without knowledge of underlying middlewares or configuration issues.

Building on the experiences from the development of the Job Submission Service [4], the GJMF is designed to be composed of a layered set of small, easy-to-use components which each provide a (within the system) unique and specific functionality. This approach - offering many services which operate relatively independently of each other - allows system administrators to set up environments which offers only the functionality they are interested in. As an added benefit, this approach offers more ways to use the system than a conventional monolithic design with a preset range of operation modes. The granularity of the system functionality reaches from the bottom layer that provides an isolation from the underlying middlewares to the top layer where client APIs facilitates the use of the framework.

To summarize: the immediate goal of this work is to develop a functional infrastructure for job management in collaborative scientific grid environments. In the long-term perspective, techniques from this infrastructure are expected to be extended to more generic computing environments. These goals are pursued by the study, implementation and evaluation of emerging standards within the grid field.
5 Design criteria

In addition to the overall goal of creating robust and versatile software, a number of explicit design criteria has been formulated for the GJMF.

5.1 Small, single-purpose components

By the identification and isolation of small, single purpose components of the system, the design of the GJMF is directed towards a modular approach. The intent of this approach is to leverage the well known benefits of modularized software development; providing decreased development cycle lengths and yielding more robust software (with less need for maintenance and support). In theory at least, software components based on standards and designed with a limited scope of functionality should result in fewer bugs and longer software life spans.

5.2 Standards conformance

As the grid community evolves in a very rapid pace, building on existing and emerging standards is a necessity to achieve any type of software interoperability. Building of the Globus Toolkit and utilizing standards such as the WSRF [9], JSDL [3], GLUE [1], WSAG [2], OGSA [5], OGSA-EMS [5], etc and offering generic functionality, the GJMF is intended to be easily extensible to future middlewares and grid standards. An expressed part of this work is also to contribute implementations and evaluations of emerging standards and services.

5.3 Middleware abstraction

When developing software which is to be compatible with many underlying systems, it is natural to attempt to hide system specific behaviors under common abstraction layer. As will be seen in a later section, the GJMF attempts to encapsulate middleware dependent issues and details in a small set of services (the Job Control Service and the Resource Selection Service). This construct enables higher level services to focus on their respective responsibilities with complete disregard to which middleware is used by the set of resources the services interact with.
5.4 Ease of use

A fundamental goal of the GJMF project is to develop software that is efficient and highly functional, while at the same time easy to use, extend and administer. As an example of this effort can be mentioned that all services in the GJMF come with a pure Java client API in addition to the basic web service interfaces. This client API is intended to ease the transition into grid programming for Java developers with limited experience of Globus Toolkit or web services. All functionality offered by the services in the GJMF, including customization and extension of the framework itself, is exposed through the client API. Customization of services is done by implementing simple Java interfaces and providing those implementations to the container. Another example of this endeavor is that all services by default use only local services (i.e. services hosted in the same container as the service itself) if they are not explicitly configured to do otherwise.

5.5 Software versatility

As previously mentioned, some services in the GJMF provide a set of customization points where third parties can implement customization plug-ins to alter the behavior of services. These plug-ins strictly offer monitoring and advisory functionality and should help make the GJMF more flexible and versatile for those who wishes to extend the default functionality. Another design feature of the GJMF that targets versatility is that each service that uses another service in the GJMF can be configured to use a set of these services at once. This construct, which allows a service to simply try another when a targeted service becomes unavailable, allows for increased robustness in terms of service redundancy as well as offering the administrator to construct a virtual infrastructure where services may join and leave the infrastructure at will. The services in the GJMF by default attempts to make local calls, bypassing the web service interface and calling the service back-end directly whenever possible. I.e. this is possible when the caller and callee service are hosted in the same container. This local call mechanism (which is entirely transparent to the end user) greatly reduces service call overhead by avoiding unnecessary message serializations and credential delegations.
5.6 Layers of functionality

As end users requirements of a system varies broadly, one of the main purposes of the GJMF is to identify different generic types of use cases for a job management environment. Rather than attempting to provide a single software component that services all of the end users needs, the functionality of the framework has instead been separated into different levels of granularity; each offering the end user a different set of functionality. The GJMF offers, from a conceptual point of view, five distinct levels of functionality.

![Diagram of GJMF functionailty layers]

Figure 1: GJMF functionality layers
From the ground and up, the conceptual functionality layers of the GJMF are:

**Middleware Abstraction Layer**
Similar to the hardware abstraction layer of an operating system, the middleware abstraction layer provides the functionality of a set of middlewares while encapsulating the details of these underlying middlewares. This construct allows higher layers to use the middleware services without any knowledge of their actual implementation details.

**Basic Job Submission Layer**
The services in the basic job submission layer offers basic job submission functionality without any type of error recovery. Analogous to the fire and forget model of the reliable job submission layer, the functionality provided by this layer is more of a single shot nature; i.e. should an error occur in the job submission process, the problem will have to be handled by the user.

**Reliable Job Submission Layer**
The reliable job submission layer provides an easy to use, fire and forget type of job submission in the GJMF. The services in this layer enables a user to submit jobs (or groups of jobs) which are then processed without any further input from the user. The services in this layer handles re-submission and other failure handling issues according to an established best-effort type of protocol.

**Advanced Job Submission Layer**
As the name implies, the services located in the advanced job submission layer provides job submission functionality of a more non-trivial nature. Currently on the drawing board, a workflow execution engine service is being planned as part of the advanced job submission layer.

**Application layer**
Residing at the top of the GJMF, the application layer contains the GJMF client API, specialized software clients (such as grid or application portals) and command-line tools provided with the framework. Software residing in this layer typically provides usage scenario specific access to the GJMF and are generally not considered to be an architectural part of the GJMF.
6 Architecture

Built on the service-based platform Globus Toolkit 4, the Grid Job Management Framework offers a set of grid services which combined constitutes a multi-tiered job submission, control and management architecture.

6.1 Task Group Management Service (TGMS)

The Task Group Management Service (TGMS) [13] provides a high-level service for autonomous processing of groups of jobs. Using the fire-and-forget metaphor, the user of the service submits groups of jobs which are processed until all jobs are completed (successfully or otherwise) without any further user interaction. The TGMS relies on the TMS for actual job submission and is itself primarily targeted towards end users, although it is likely that it also will be used as a job submitter for job groups by workflow services. Conceptually, the TGMS can be thought of as a batch processing system for grids.

6.2 Task Management Service (TMS)

Like the TGMS for job groups, the Task Management Service (TMS) [14] offers a fire-and-forget reliable job submission solution for individual jobs. If job submission or execution would fail, the TMS simply resubmits it to the BSS until the job succeeds or a maximum number of retries has been reached.

6.3 Brokering & Submission Service (BSS)

The Brokering and Submission Service (BSS) [11] provides primarily a functionality abstraction for job submission - it receives an abstract job specification (task) as input, retrieves an execution plan (that is, a list of updated job descriptions including references to computational resources suitable for job execution) from the RSS for the job, and the submits the job to a computational resource (via the JCS) using an updated job description from the execution plan.

6.4 Job Control Service (JCS)

Like a virtual machine, the Job Control Service (JCS) [8] provides a functional abstraction of the underlying middleware(s) and offers a platform independent job submission and control interface. The JCS can be used independently but requires the caller to provide a complete execution plan, i.e., a complete job specification in JSDL [3], including a reference to a computational resource to whom it will submit the job.
6.5 Resource Selection Service (RSS)

The Resource Selection Service (RSS) utilizes the grid information system to provide a mapping from abstract job specifications (tasks) to suitable computational resources currently available on the grid. The RSS is comprised of two underlying standardized services - the Candidate Set Generator (CSG) [5] and the Execution Planning Service (EPS) [5]. As the name implies, the Candidate Set Generator locates a set of computational resources that fulfills the requirements of the job. Given such a candidate set, the Execution Planning Service formulates what is known as an execution plan - a prioritized list of the computational resources in the set (in the form of ranked and updated job descriptions). This prioritization is created according to a previously known or explicitly provided set of rules. In summary, the RSS is used (in the GJMF primarily by the BSS) to retrieve host mappings for tasks, i.e. matchings from tasks to computational resources.

6.6 Log Accessor Service (LAS)

The Log Accessor Service (LAS) [12] provides a reliable logging system for the stateful services in the GJMF. From the LAS an end-user can retrieve detailed information pertaining to the submission and execution of jobs, tasks and task groups alike. Although some of this information can be ascertained directly from the originating service in the form of status notifications, the LAS is the only place in the GJMF from where a complete log of system events - that is, a log including resulting system events in services used by the currently queried service can be found. The LAS is also the only service in the GJMF from which a user can retrieve information about tasks, task groups and jobs which has been completed and removed from the GJMF.

6.7 Dynamic Configuration Service (DCS)

Currently on the drawing-board, the DCS is intended as a solution for dynamic configuration of the GJMF. The expected benefits of such a service includes having a single point of configuration for multiple services and facilitation of a simple way of constructing virtual networks of GJMF services.
6.8 Data Management Service (DMS)

A service called the Data Management Service, is being considered for the GJMF. The GJMF currently assumes that all files required for the execution of a job on the grid will be available to the computational node via GridFTP. The need for a service that somehow handles the GridFTP URLs (and possible the files themselves) has however been observed, especially for use cases where the user needs transfer files to (and store files on) the grid for future job executions. Whether this will be a standalone grid service or a front-end to an existing storage solution remains to be investigated.

6.9 Client API

Albeit not a service, the GJMF Client API is an integrated part of the Grid Job Management Framework as it provides utility libraries for creating tasks and task groups, performing credential delegations and service level APIs for accessing all components in the Grid Job Management Framework. The expressed purpose of the GJMF Client API is to provide easy-to-use Java-only programmable access to all parts of the Grid Job Management Framework.
7 Use cases

As the Grid Job Management Framework by its nature offers a multitude of potential use cases we will here limit the discussion to a few scenarios which hopefully captures enough of the general idea to illustrate the individual role of each component in the GJMF. Note that in all of these scenarios, a task or task group specification is already assumed to be constructed (using for example, the client API); i.e. when a task or task group is referenced, we refer to an abstract task definition formatted in the JSDL format. Also note that for all scenarios, files required for the execution of tasks on the grid are assumed to be available to the computational resource using GridFTP and that status notifications are available for tasks, task groups and jobs from the service to whom it was submitted.

7.1 Job control

Using the JCS to submit and control a job executing on a specific computational resource constitutes the simplest scenario for using the GJMF.

1 The user submits a complete job specification to the JCS, including a named computational resource to execute on. Note that using the JCS service interface the user has the option to stop (and additionally also remove) the job at any time.

2 The job is dispatched to the specified computational resource by the underlying middleware.

3 Files are staged in and the job is queued for execution on the computational resource.

4 The job executes and resulting files are staged out, progress reports in the form of status notifications are delivered to the JCS (and in turn to the client).

5 The user removes the job from the JCS. If not explicitly removed, the job will be automatically removed at a later time by the WSRF lifetime mechanism in the Globus Toolkit.
7.2 Basic job submission

For the case when a job specification is available, but the end user does not care to manually determine on which computational resource a the job should be executed, the BSS can be used to submit the job.

1. The user submits a job (i.e. an abstract task definition without an execution plan / named computational resource) to the BSS.
2. The BSS elicits an execution plan from the RSS
3. The BSS updates the job with a complete job specification (found in the execution plan) and, finally, submits the updated job to the JCS.
4. The JCS returns an End Point Reference (EPR) to a job resource in the JCS. Using this EPR the user can control the job just as if it would have been submitted directly to the JCS.
5. The JCS processed the job as described in section 7.1.

7.3 Reliable job submission

If the user likes to submit a job for execution, without having to handle re-submissions and other failure handling issues, the TMS is a suitable choice.

1. A task is submitted to the TMS.
2. The TMS adds the task to an internal queue.
3. The TMS continuously traverses its task queue and submits the tasks one by one to the BSS. Submission and execution failures are handled by rescheduling the task (up to a maximum number of times).
4. The BSS processed the tasks as described in section 7.2.
7.4 Reliable submission of multiple jobs

When a number of mutually independent jobs are available for submission, the TGMS is a suitable choice.

1. A task group is created and submitted to the TGMS.
2. The TGMS adds the tasks in the task group to an internal queue.
3. The TGMS continuously traverses its task queue and submits the tasks one by one to the TMS. Errors reported to the TGMS by the TMS are treated as failures.
4. The TMS processed the tasks as described in section 7.3.

7.5 Tracking execution progress

When the user is interested in more detailed execution information, for example when tracking where a specific job has executed, the LAS can be used.

1. A task group is submitted to the TGMS.
2. The TGMS processes the task group as described in section 7.4.
3. Using the unique identifier of the task group, the user retrieves the task group log from the LAS. Using the identifiers for the tasks in the task group the user can retrieve the individual task logs as well as the logs of all resulting jobs.

7.6 Interactive jobs

When running a job which will interact with another job or system, for example a rendering pipeline which will send data to a visualization engine, the status notifications of the GJMF services can be used for (coarse) program synchronization.

1. A job is submitted to the BSS.
2. The BSS processed the jobs as described in section 7.2.
3. Status notifications are delivered from the job and the client uses them (in conjunction with the IP address of the job found in the EPR) to trigger the initiation of TCP/IP communication.
8 Terminology

Task - an abstract task specification.
As the term "job" has been found to be ambiguously defined in different contexts we here define a task to be an abstract task specification expressed in the Job Specification Description Language (JSDL) [3].

Job - a task instantiation on a computational resource.
Instantiations of tasks on specific computational resources (as submitted by the Job Control Service) are referred to as jobs. The distinction between a task and its job(s) is made as the GJMF may generate several jobs for each task (e.g. upon job submission or execution failures). Technically, a job is represented by a tuple containing a reference to the original task, the job specification, the (translated) native job description and a reference to the computational resource which has been matched to the task.

Task group - a collection of autonomous tasks.
Finally, a task group is here defined as a set of autonomous tasks without any ordering, prioritization or other types of internal relationships.
9 State models

The state model of the GJMF defines states for task groups, tasks and jobs. Task group states are states for task groups in the TGMS, task states are states for tasks in the TMS and, finally, job states are states for jobs in the JCS.

9.1 Job states

As defined by the JCS, a job is defined to at all times reside in one of five valid states. For interoperability reasons, these are chosen to be identical to those of the Basic Execution Service (BES) [7]:

Pending
The job has been submitted to the JCS, but have not yet begun to execute on the computational resource. In addition to the actual job submission, this step also entails file staging and credential delegations.

Running
The job has begun executing on the computational resource.

Canceled
The job execution have been explicitly canceled by the owner or a systems administrator, and all related activities have been terminated.

Failed
The job execution (or resulting file stage outs) has met irrecoverable difficulties and has thus terminated erroneously.

Finished
The job execution has ended successfully. File staging have been completed and any resulting output data is available where specified in the JSDL.

Figure 2: Job states
9.2 Task states

A task is defined to have six valid states:

Idle
The task is available and waiting for submission (i.e. not yet submitted or active).

Submitting
The task is currently being submitted to a host and is therefore not available for submission (to other hosts).

Active
The task has been successfully submitted to a computational host (i.e. there exists a corresponding job in a JCS).

Canceled
The task has been canceled as a result of the system encountering invalid user credentials (i.e. the task owners proxy have expired). Note that, in addition to a proxy renewal, the task will have to be explicitly removed from the TMS before it can be successfully resubmitted.

Completed
The task has been successfully completed (i.e. has a finished job).

Failed
The task has failed (i.e. the task has reached its maximum number of submission or execution attempts).

Figure 3: Task states
9.3 Task group states

Finally, a task group may be found in any of the three valid states:

**Active**

The task group is residing in the TGMS and contains tasks that are available for submission to the TMS.

**Suspended**

The task group have been explicitly or implicitly suspended. Suspending a task group allows the currently active tasks to continue running, but delays any future job submissions until the task group has been explicitly resumed. Implicit suspension is most likely the result of the system detecting expired proxy credentials. The task group may contain tasks that have not yet been completed but these are not available for submission until the task group has been explicitly resumed. Note that a suspended task group also may contain active tasks (i.e. tasks that have queued or running jobs), this implies that a suspended task groups state may transit to done (if these jobs are successful and no tasks remain to process).

**Done**

All tasks in the task group have been processed (successfully or otherwise).

![Figure 4: Task group states](image)
10 Development history

Work on the Grid Job Management Framework began with the development of the first prototypes of the Task Management Service and the Job Control Service. During the development of the Job Submission Service it had become apparent that smaller, more focused services more easily could be adapted to emerging standards such as the Basic Execution Service (BES) and the Candidate Set Generator. The Job Control Service was designed with a BES-style of abstraction layer and the Task Management Service as a reliable job submitter which could handle most of the mundane tasks of re-submissions and statistics gathering.

During development, the TMS functionality was extended to handle groups of independent tasks (task groups, i.e. it targeted embarrassingly parallel tasks) and the service was therefor renamed the Task Group Management Service in its initial functional version. This, first version of the TGMS incorporated complete log information for every task group, task and all resulting jobs, a choice that gave the service an undesirably large memory footprint. Further study also showed that this design was poorly matched to the WSRF persistence mechanism (as entire task groups was deserialized and subsequently serialized for each log update notification from a job) and thus the TGMS was redesigned.

In the second generation of the TGMS, the logs were extracted and placed in their own service - the Log Management Service. This allowed for making the service state driven rather than log driven, and to facilitate the use of WSRF persistence the TGMS was further split into two services - the Task Group Management Service and the Task Management Service. The Log Management Service was later refactored to a more flexible, plug-in based architecture and renamed the Log Accessor Service in the process.

A notable change in terminology is that the earlier versions of the Brokering and Submission Service was named Job Submission Service after its predecessor. But as the BSS does not offer the same level of functionality (the JSS being far more advanced than the BSS) it was renamed in order to reduce confusion on the topic.
11 Future work

Dynamic configuration
Providing a dynamic configuration model to the GJMF would allow for both a hot-swap type of reconfiguration of the system as well as a single point of configuration for the entire framework.

Coallocation and reservations
Acknowledging the need for scheduled and cooperating jobs, services for advance reservation and co-allocation are being considered.

Workflows
As some computational jobs are hierarchical in nature, a workflow service providing a structured sequential approach to job submission is being considered. Whether this will be a full service in itself or an interface to an existing workflow engine remains a topic for investigation.

BOINC front-ends
Enabling a BOINC-based front-end to submit jobs via the Grid Job Management Framework could be transparently realized by developing a translation service that interacts with the DMS and the TGMS / TMS.

BOINC back-ends
Developing a BOINC-based back-end for the Grid Job Management Framework would provide the end-user with a convenient way to submit jobs to a CPU-cycle harvesting framework. Clearly there are types of computations that are well suited for the BOINC type of a more dynamically distributed computational environment.

Log4Grid
A general purpose log service, with Log4Java interoperability would make a very useful tool in grid development. Developing it in the form of a general XML logger would provide a generic service which could be reused for the LAS and a resource property state monitoring service.

Generic application front-ends
Designing and prototyping general forms of application front-ends to the GJMF would most likely give insight into both the types of problems facing developers of such systems. Experience gained from such work would benefit both developers of clients as well as developers of the core of the GJMF.
References


