



**T A S K 3 . 2**  
**G R I D R E S O U R C E M A N A G E M E N T**  
**S R S**

**WP3 New Grid Services and Tools**

---

Document Filename: **CG-3.2-SRS-0010.doc**  
Work package: WP3 New Grid Services and Tools  
Partner(s): **UAB, DATAMAT, CSIC**  
Lead Partner: **UAB**  
Config ID: **Task 3.2 Grid Resource Management SRS**  
Document classification: Public

---

Abstract: This document provides a description of the initial specification and the requirements of Scheduling Agents for interactive parallel applications that are going to be developed in the framework of the CrossGrid project.



---

**Delivery Slip**

	<b>Name</b>	<b>Partner</b>	<b>Date</b>	<b>Signature</b>
<b>From</b>				
<b>Verified by</b>				
<b>Approved by</b>				

**Document Log**

<b>Version</b>	<b>Date</b>	<b>Summary of changes</b>	<b>Author</b>
0.1 DRAFT	18 Feb 2002	First Draft version	Miquel Senar, Elisa Heymann
0.2 DRAFT	9 Apr 2002	Updated according to new versions of WP1 SRS	Miquel Senar, Elisa Heymann
0.3 PUBLIC	7 May 2002	Added changes according to TAT comments	Miquel Senar, Elisa Heymann
1.0 PUBLIC	31 May 2002	Added changes according to comments made by external reviewers	Miquel Senar, Elisa Heymann, Stefano Beco

---

## CONTENTS

<b>1. INTRODUCTION</b> .....	<b>4</b>
1.1. PURPOSE .....	4
1.2. SCOPE .....	4
1.3. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS .....	5
1.4. REFERENCES .....	6
1.5. OVERVIEW .....	6
<b>2. OVERALL DESCRIPTION</b> .....	<b>7</b>
2.1. PRODUCT PERSPECTIVE .....	7
2.1.1. <i>System interfaces</i> .....	8
2.1.2. <i>User interfaces</i> .....	8
2.1.3. <i>Hardware interfaces</i> .....	9
2.1.4. <i>Software interfaces</i> .....	9
2.1.5. <i>Communications interfaces</i> .....	9
2.1.6. <i>Memory constraints</i> .....	10
2.1.7. <i>Operations</i> .....	10
2.1.8. <i>Site adaptation requirements</i> .....	10
2.2. PRODUCT FUNCTIONS.....	10
2.3. USER CHARACTERISTICS .....	14
2.4. CONSTRAINTS .....	14
2.5. ASSUMPTIONS AND DEPENDENCIES .....	15
2.6. APPORTIONING OF REQUIREMENTS .....	15
<b>3. SPECIFIC REQUIREMENTS</b> .....	<b>16</b>
3.1. EXTERNAL INTERFACES .....	17
3.2. FUNCTIONS .....	18
3.3. PERFORMANCE REQUIREMENTS .....	18
3.4. LOGICAL DATABASE REQUIREMENTS .....	18
3.5. DESIGN CONSTRAINTS.....	19
3.6. STANDARDS COMPLIANCE.....	23
3.7. SOFTWARE SYSTEM ATTRIBUTES .....	23
<b>4. APPENDIX A. DESCRIPTION OF COMPOSED JOBS USING CONDOR DAGS</b> .....	<b>24</b>
<b>5. APPENDIX B. RELATED WORK</b> .....	<b>26</b>
<b>6. INDEX</b> .....	<b>30</b>

## 1. INTRODUCTION

### 1.1. PURPOSE

The purpose of this document is to describe the requirements of Scheduling Agents for parallel applications submitted to a grid environment. Scheduling Agents are schedulers, they make decisions about where, when, and how to run parallel jobs on Grid resources as specified by a set of policies, priorities, requirements and limits. They are responsible of deciding the allocation of application tasks to computing resources in a way that tries to guarantee that applications are conveniently, efficiently and effectively executed. Agents will interact with other services developed in the CrossGrid project and this specification is aimed to be compatible with other on-going grid projects (namely, the Datagrid).

The intended audience of this document is the Task itself, Tasks 3.1, 3.3 and 3.4 regarding the interfaces and information required by this Scheduling Agents and WP1 regarding the set of requirements specified in their SRS that will be supported by SA.

### 1.2. SCOPE

Given a user's parallel application submitted to the grid, Scheduling Agents will be responsible for optimising scheduling and node allocation decisions. They are not grid resource managers, such as Condor-G, as they do not manage the queue of jobs that interact directly with Grid resources nor the local queues in the remote Computing Elements where the application is going to run. Their basic role is to tell the grid resource manager what to do, when to run jobs and where. Specifically they will carry out three main functions:

1. selection of the "best" resources that a submitted job can use. This selection will take into account the application requirements needed for its execution as well as some ranking criteria used to sort the available resources in order of preference.
2. perform the necessary steps to guarantee the effective submission of the job onto the selected resources. In general, Scheduling Agents will carry out this function by supplying a complete information about the job and the resources to be used to a suitable grid resource manager, which subsequently will be in charge of running the tasks of the job on the given set of resources.
3. monitor job execution and report on job termination. This function will be normally carried out by some sort of notification mechanism provided by the grid resource manager to which the job was submitted.

Scheduling Agents will not be in charge of providing a fault tolerant execution environment to the application. This service should be provided either by the particular grid resource manager to which the Scheduling Agents will submit the jobs or to the particular remote sites on which the application will be run. In case of partial or total failures of any Grid element that derive in an abnormal termination of the application, Scheduling Agents will simply cancel the job, put it back into the queue of pending jobs and notify the event to higher services with which users have direct interaction. These services should be responsible to keep the job into the queue for a later automatic resubmission by the Scheduling Agent or to return the job to the user. In the former case, the job will go through the three elements described above. In the former case, the job will simply be discarded.

### 1.3. DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

#### Definitions

Condor	Condor is a High Throughput Computing (HTC) environment that can manage very large collections of distributively owned workstations
Condor-G	Condor-G is a resource manager that is used as a front-end to a computational grid. It provides job monitoring, logging, notification, policy enforcement, fault tolerance and credential management.
Condor Glide-in	Mechanism that allows the temporary addition of a Globus resource to a local Condor pool. The addition is accomplished by installing and executing some of the Condor daemons on the Globus resource.
Globus	The Globus Toolkit is a set of software tools and libraries aimed at the building of computational grids and grid-based applications.

#### Acronyms/Glossary/Abbreviations

ClassAd	Classified Advertisement
DAG	Directed Acyclic Graph
DataGrid/EDG	The EU DataGrid Project IST-2000-25182
GIS	Grid Information Service, aka MDS
GRAM	Grid Resource Allocation and Management
job-ad	class-ad describing a job
JDL	Job Description Language
JSS	Job Submission Service
MDS	Metacomputing Directory Service, aka GIS
MPI	Message Passing Interface
NA	Not Applicable
RB	Resource Broker
R-GMA	Relational Grid Monitoring Architecture
SA	Scheduling Agents
TBD	To Be Defined
WMS	Workload Management System
WP	Work Package

## 1.4. REFERENCES

- [1] Definition of architecture, technical plan and evaluation criteria for scheduling, resource management, security and job description, DataGrid-01-D1.2-0112-0-3
- [2] Ten Actions for Superscheduling, Scheduling Working Group, Global Grid Forum, <http://www.cs.nwu.edu/~jms/sched-wg/WD/schedwd.8.5.pdf>
- [3] Task 1.1. Biomedical application - Software Requirements Specification Draft, CG-Task1.1-SRS-0001-0.1-DRAFT\_c.doc
- [4] Task 1.2. Flooding crisis team support - Software Requirements Specification Draft, CG-1.2-SRS-0001-1-0-DRAFT-A.doc
- [5] Task 1.3. Distributed Data Analysis in HEP - Software Requirements Specification Draft, task1[1].3srsv1.doc
- [6] Task 1.4. Air pollution, weather forecasting and sea wave modelling - Software Requirements Specification Draft, CG-T1.4-SRS-DRAFT.doc
- [7] Task 1.4b. Data Mining for Weather Forecasting - Software Requirements Specification Draft, task1[1].4.b-srsv1.doc
- [8] Advance Reservation API, Scheduling Working Document: 9.4, Global Grid Forum, June 2001. <http://www.cs.nwu.edu/~jms/sched-wg/WD/schedwd.9.4.pdf>

## 1.5. OVERVIEW

Section 2 provides an overall description of Scheduling Agents and section 3 details specific requirements and dependencies with other WPs or projects. Appendix A describes the Directed Acyclic Graph notation used in the Condor system. Appendix B contains a brief state-of-the-art related to scheduling and resource management in Grid environments.

## 2. OVERALL DESCRIPTION

### 2.1. PRODUCT PERSPECTIVE

Scheduling Agents, as defined in this document, constitute an extension of part of a Workload Management System (WMS) as described in the Datagrid project [1]. In particular, SA will automate most of the activities related with resource discovery and system selection, as described in [2]. As also pointed out in [2], SA differ from other common schedulers is that the SA “don’t own the resources and therefore don’t have total control over them. Furthermore, SA don’t have control over the entire set of jobs on the system, or even necessarily know about them, so decisions about an entire set of jobs to a resource cannot be made.”

SA will extend the basic services of the WMS to user jobs consisting of parallel applications that may, eventually, exhibit an interactive behaviour. Initially, support will be provided to parallel applications using MPI, as this is the original platform described in the project proposal which is also mentioned in most of the SRS documents (from the deliverables of month three) prepared by the corresponding tasks ([3, 4, 5, 6, 7]. Support for other parallel libraries (such us, PVM) or environments (Cactus) may be discussed in the future according to the special needs that WP1 partners will report. Currently, those environments are mentioned in some of the related documents only as tentative packages. We assume that the WP1 application developers will refine further their requirements as the project evolves. A specific MPI implementation has to be also agreed, although initial candidates, mentioned by most tasks, are MPICH and MPICH-G2.

The term job or application has a vague definition, according to the descriptions included in the SRS documents of WP1. Most of the applications described in these documents are composed of several independent but inter-operable and dependent components. All these components may run in parallel (see, for instance, the biomedical application), but they may also exhibit some sort of interdependency (for instance, flood team support application). Some of these components have a specialised function (visualisation, data acquisition, user interaction,...) that forces them to be run onto a particular resource, without any intervention of an SA because they are not supposed to run on grid resources but on local ones. Other components are identified (either implicitly or explicitly) as been candidates to run on grid resources. They consist of time consuming computations such us simulations (in the biomedical application, in the flood team support application and in the air pollution, weather forecast and sea wave modelling application), or data mining techniques (in weather forecasting and HEP applications). Finally, interactive HEP applications require also the eventual existence of a component that is responsible for allocate nodes and install data servers in them (which also implies data transfer before the interactive application starts). Thus, according to these descriptions, jobs submitted to SAs will be classified as:

- Simple jobs: consist of a single application (either sequential or parallel) that can be identified with a single executable binary. Each simple job will be described by means of a single file using the Job Description Language described in [1].
- Composed jobs: consist of a collection of single applications (each of which, can be either sequential or parallel) that exhibit some sort of inter-dependency. Each one of the applications will be described using JDL, as the previous case. In addition to that, dependencies between jobs will be described by a suitable language. Normally, a Directed Acyclic Graph (DAG) can be used to represent a set of applications where the input, output, or execution of one or more programs is dependent on one or more other applications. In this case, single applications are

nodes (vertices) in the graph, and the edges (arcs) identify the dependencies. A possible representation to be used for composed jobs could be the one adopted by Condor DAGman (a meta-scheduler for Condor jobs). A brief description of the DAG notation used by DAGman is included in appendix A.

Initially, SA will be designed to work with simple jobs. Additional support for composed jobs will be added at later stages of the project, once the proposed syntax to represent this kind of jobs has been discussed and approved with WPI developers. Description of scheduling actions to be performed on composed jobs will also be described in subsequent documents when further details about application requirements will be available. This document focuses on the initial support of SA to simple jobs.

Another relevant issue regarding the design of SA is related to the support of interactive jobs. Two main situations have been identified from the application specifications:

- The ability of the user to cancel a given job as a result of an inspection of the job output during the execution of the application. This situation requires, on the one hand, the addition of some simple command available at the SA to cancel a specific job. On the other hand, this cancel operation may need some additional services that fall out of the scope of SA, such as the ability to transfer outputs files during the execution of the application. This service must be provided either by the grid resource manager that controls the execution of the application or by the local resource managers on the remote sites where each application task is running. Further investigation has to be carried out in order to adopt a grid resource manager that provides such service.
- The ability of an SA to start immediately the execution of an interactive application in order to achieve a response time as low as possible. Again, this service will also rely on the ability of the underlying resource managers to provide facilities that can be exploited by the SA. The ability of making an advance reservation appears to be a valuable mechanism to support this requirement. Other useful mechanisms that will be investigated include the use of priorities between user jobs, which can be extensible to priorities between users. The priority schema would be used in conjunction with a job pre-emption mechanism in order to suspend an already running job and allocate an interactive job on the released resources. Again, at this stage of the project, the inclusion and the scope of this priority mechanism requires further elaboration and will not be included in early prototypes of the SA. They will eventually be introduced according to the comments and suggestions made by application developers and the Technical Architecture Team.

### **2.1.1. System interfaces**

Scheduling Agents will offer one interface to external Grid components. This interface consists of an API with several functions that allow a client application, such as a User Interface, to submit or cancel a job. Simple jobs will be described by a job-ad using a Job Description Language (JDL) that will specify the relevant characteristics of the job, its requirements and user specific needs in terms of scheduling options.

### **2.1.2. User interfaces**

No direct interface will be provided to users. All interactions should be carried out by means of intermediate client programs.

### 2.1.3. Hardware interfaces

No special interfaces with hardware components are considered. Specific use of communication ports will be described in future documents describing the design and implementation of Sas..

### 2.1.4. Software interfaces

Scheduling Agents will offer one external interface that, in principle, should be used by a client application (such as, the Web Portal or any alternative user interface). The main services provided by this API would be:

- **Submit a job:** a job described in a JDL file is passed to the SA. An identifier for the job (`job_id`) is returned once the SA verifies the correctness of the JDL file, which means that the job has been accepted by the SA. The `job_id` will be used by SA for later references to this job.
- **Cancel a job:** the SA will cancel the execution of the job identified by its `job_id`. It will issue a kill command to the resource manager and the job will be removed from the SA queue.

Although, these are the basic functions provided by SA, several variations will be later included to support some common user operations. Examples of useful variations would be, for instance, a command that cancels all the jobs of a given user.

Submitted jobs will be described by job-ads using the JDL or ClassAd terminology. Scheduling Agents will require also access to information about the available resources in the Grid environment that may be used to service each user request. Resource information will also be expressed in terms of ClassAds (as used in EDG). The necessary extensions will be added to the EDG's JDL to be able to represent the requirements of parallel applications and computing resources in order to carry out the selection process.

SA will have a second interface that will be used to pass the list of selected resources to the grid resource manager that is going to be responsible of the actual execution of the job on the selected resources. In particular, a first interface will be implemented to be compatible with the Datagrid's JSS to receive asynchronous notifications from the JSS about relevant events concerning a job, so that the SA can take appropriate actions.

### 2.1.5. Communications interfaces

Internal interfaces between the elements included in a Scheduling Agent will be described in the next document that will describe also their internal architecture. Basic mechanisms adopted will be shared memory data (for components of SA running as different threads) and through TCP/IP sockets (to communicate SAs with other elements related to job submission and grid resource management). According to future developments of Globus, communication interfaces of SAs will be compatible with Globus services based on the OGSA specification.

### 2.1.6. Memory constraints

No special constraints have been identified at this moment.

### 2.1.7. Operations

Scheduling Agents will be running as permanent daemons. They will be started together with the rest of basic services included in the Grid middleware. They will react to any claim coming from client programs. Once a user job has been successfully submitted to a set of computing resources, they will react to events coming from inferior services of the Grid infrastructure.

### 2.1.8. Site adaptation requirements

TBD

## 2.2. PRODUCT FUNCTIONS

The basic functions involved in the management of user jobs include:

- **Matchmaking Process:** this activity consists on finding a match between the user application requirements and the resources available on the Grid. Characteristics of Grid resources are retrieved both from information services from the Grid that may include Replica Catalogues and resource directories (MDSs). A list of candidate resources will be generated by this Matchmaking Process that will include resources that fulfil the application requirements (including, for instance, resources that are available, in which the user has authorisation to run jobs and that match the architecture required by the application.
- **Resource Selection:** this process will be implemented by allocation heuristics that will be responsible for selecting the actual set of resources that is going to be used by the application from all the resources that fulfil the application requirements. The selection criteria will be guided by the user specific optimisation criteria and the use of additional information from the Grid resources provided by monitoring services. A ranking mechanism will be used to sort the resources filtered by the matchmaking process. Resources will be selected starting from the beginning of this sorted list until a number equal to the one specified in the job description file is obtained. Additional selection criteria may be applied if they are specified in the application requirements (for instance, resource selection may enforce that resources belong to the same farm). It is worth mentioning that the matchmaking process and the resource selection may be combined into a single service by combining the specification of requirements and the ranking criteria at the same time. Such combination is possible, for instance, if the Condor ClassAd library is used.
- **Submission Service:** This process performs the actual submission of a job to the resource manager of the Grid. It has to ensure that the resource manager ensures a simultaneous co-allocation of all the resources passed by the Resource Selection process. It will interact with the Resource Selection process if the resource manager is not able to allocate all the resources specified initially. The interaction will be repeated until a sufficient number of resources is allocated. Otherwise, an error will be reported. The Submission Service will be later responsible

for the actual submission of tasks to the appropriate resource. It will also monitor the application execution and inform to higher levels about any event that significantly affects the normal execution of the application (i.e., normal exit or abnormal termination). It will also implement cancellation request issued by the user through the suitable interface. The Submission Service will be a simple wrapper that will finally submit the job to a grid resource manager. Condor-G is going to be adopted as grid resource manager because it provides the desirable characteristics of reliability and persistency required to support temporal failures in different elements of the grid infrastructure (CEs, Globus gatekeepers,...). MPI applications will require the use of some additional features not available directly in the standard configuration of Condor-G. In particular, the glide-in mechanism will be evaluated as a candidate mechanism to support MPI applications distributed over multiple sites.

Figure 1. shows an initial design of SAs with its main components. According to the elements depicted in this figure we describe below how the above functions will be carried out and which element will be responsible for them.

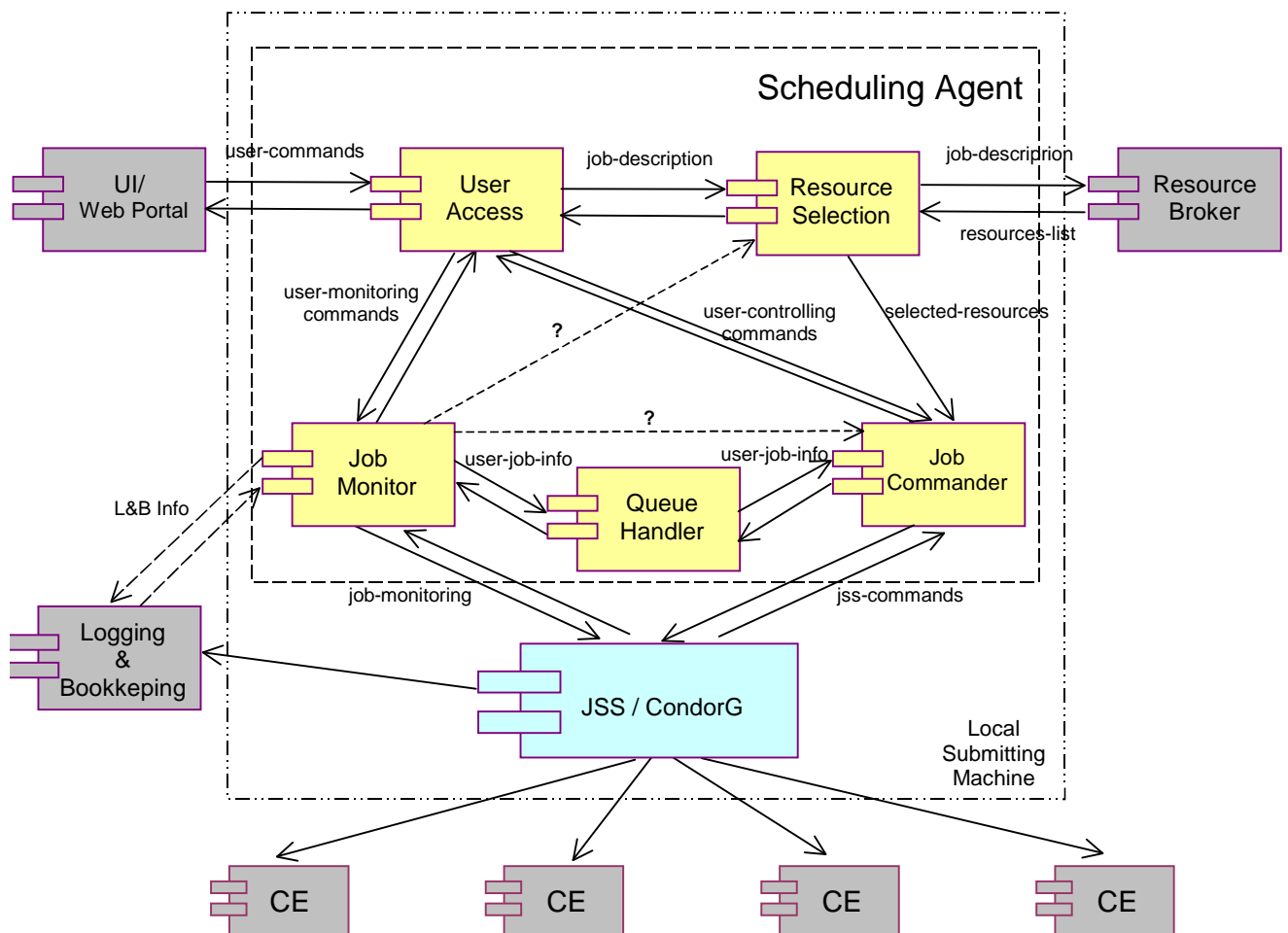


Figure 1. High level view of Scheduling Agent (SA) functions

In our design, Scheduling Agents are binded to a given user in a 1:1 relationship, and there are different copies of them for different users. Each user will know where his/her pair runs (machine, port) from his/her user profile. They will include the resource selection service and the submission service. The matchmaking process will be carried out by an external element: the Resource Broker. The Resource Broker will be interfaced to Grid Information Services (such as Globus MDS and Datagrid R-GMA). The Datagrid's Resource Broker functionalities will potentially be reused to implement this interface. Internally, the Resource Broker will be based on the Condor ClassAds library, which is able to provide single resource selection. It will be extended with new functionality so that it provides also multiple resource selection.

The SA can be composed by five main internal functions:

- User Access (UA);
- Resource Selection (RS);
- Job Commander (JC);
- Job Monitor (JM);
- Queue Handler (QH).

A typical scenario that illustrates the normal behavior of SA interacting with other middleware services to manage a particular parallel job is the following:

- The user contacts the SA through a remote connection using the Web Portal. The UA handles this connection and receives the user commands (like job-submit, job-cancel, job-status...).
- The user can submit his/her job providing to UA the job description in a JDL file. UA sends the job description to RS that will contact the Resource Broker (RB). The JDL file specifies all the information required to execute the application and includes also specific requirements and ranking information provided by the user. A minimal JDL file would look like:

```
Executable = apl.exe;      /* Name of executable */
Application_type = MPI;   /* It's an MPI application */
stdin = NULL;             /* No standard input is used */
stdout = output.txt;     /* standard output redirected */
stderr = error.txt;      /* standard error redirected */
CPUcount = 4;            /* the application requires 4 CPUs */
input_files = dat1;      /* the appl. uses one input file */
output_file = out1;      /* and generates one output file */
requirements = other.OpSys == "RH 6.2" && other.Arch ==
"INTEL"; /* the application has to run on machines with Red
Hat 6.2 as operating system and Intel processor */
```

- RB also receives from Grid Information Index and Replica Manager/Catalogue information about resources on the Grid. RB finds a match between the user application requirements and available grid resources (Matchmaking Process). Then, RB sends to the RS the list of found suitable resources.

The list of resources would be for instance:

```
quidam.cs.uab.es
saltimbanco.cs.uab.es
alegria.cs.uab.es
soleil.cs.uab.es
cirque.cs.uab.es
```

Each machine should have also a file with a description of its main characteristics (equivalent to the JDL file in the case of jobs)

- RS uses ranking mechanisms to select from this list the resources that will be used to run the submitted job. In our example, the first four machines could be the ones selected.
- The JC will issue a co-allocation command for all the resources selected. This co-allocation command will be based on the Condor glidein mechanism
- Once, the co-allocation command has succeeded, the JC uses the information about the selected resources and the user JDL file to create a command for JSS. The command is submitted to JSS and the QH stores in its queue information about submitted job. The JM checks the results of the submission operations by direct queries to the JSS.
- If the co-allocation command does not succeed or the grid resource manager is not able to submit the job to the resources, the JM (or the JC: see the two flows with question marks in the figure) asks RS for other resources matching the application requirements (specified by JDL file), to perform an automatic resubmission of the job.

In our example, this means that `cirque.cs.uab.es` would be tried first if one of the previous machines would not respond. If `cirque.cs.uab.es` also fails, the SA would ask for additional machines to the RB. If no more machines are provided by the RB, the SA deallocates any incomplete set of machines and puts the job into hold. It will periodically try to repeat the allocation until it succeeds or a maximum number of trials is reached (in the latter case a notification would be sent to the user, who can decide to cancel the job or keep it in the queue).

- All events generated by the different elements involved in the execution of the job are recorded in the Logging and Bookkeeping server (LB) through the JM. This server will be, in principle, the DataGrid Logging and Bookkeeping service, given that no specific service is being developed at the CrossGrid project. Nonetheless, the JM will store some minimum amount of information about the state of a running job.
- The user can check the status of the submitted jobs: JM receives the user-monitoring-commands by UA and gives back needed information stored in the LB. It has to be investigated whether there could be a direct access to LB server by Web Portal to query it.

Section 3 provides additional information regarding this design of SA and compares it with other alternative designs that have been adopted, for instance, in the DataGrid project.

The lack of a central control mechanism will imply that the allocation decisions made by the Scheduling Agents could be deadlock prone. The state of a particular resource may change in the time interval that goes from the output of the resource selection process to the actual claim of the resource by the grid resource manager. A simple mechanism based on time-outs will be used initially to prevent deadlock situations in the allocation of multiple resources. The allocation of all the resources for a

single job will finish with an error if a certain time-out expires before all the resources are effectively allocated. In this case, the job will be passed back to the Matchmaking Process to find a different set of available resources.

Selection of resources will be based on a set of usual steps as described in [2]. A list of significant factors that can affect scheduling decisions may be found also in [1]. They include, for instance, the location of data, requirements of job, user preferences, user priority, status of available resources, etc. Some of these factors will be used as excluding criteria. They will serve to discard resources that are not suitable for a given application. Other factors will be used to provide a ranking of resources that will be used to do the final selection. Heuristic scheduling strategies responsible for the generation of ranked lists of resources will be included and tested during the project.

It is worth mentioning also that the use of the Condor glidein mechanism has two purposes. First, it is used as a co-allocation service that guarantees that all the required resources are available before the application is transferred there for execution. Second, it will be used also to support interactive high priority applications when no resources are available but lower priority applications are already running. In this case, the SA will suspend or kill some of the low priority applications and allocate those resources to the high priority application. The glidein mechanism will guarantee that the resources will not be lost between the suspension of the low priority jobs and the starting of the high priority job. If a high priority job wanted to avoid the overhead involved in the matchmaking and resource selection processes, a similar mechanism could be applicable.

## 2.3. USER CHARACTERISTICS

As was mentioned in section 2.1.2 this package belongs to middleware, thus these components essentially do not have direct contact with the user. All user interactions with SAs will be carried out through the web portal.

## 2.4. CONSTRAINTS

- a) Regulatory policies: none.
- b) Hardware limitations (e.g., signal timing requirements): none.
- c) Interfaces to other applications: The Resource Broker has to be interfaced to Globus MDS and/or DataGrid R-GMA.
- d) Parallel operation: no special constraints.
- e) Audit functions: no special constraints.
- f) Control functions: TBD
- g) Higher-order language requirements: C and C++.
- h) Signal handshake protocols (e.g., XON-XOFF, ACK-NACK): none.
- i) Reliability requirements: Very High
- j) Criticality of the application: Very High

k) Safety and security considerations: SA have to follow the same security, encryption and authentication mechanisms commonly used by services and tools of CrossGrid.

## 2.5. ASSUMPTIONS AND DEPENDENCIES

Scheduling Agents assume that basic elements and components developed at the Resource Management workpackage of the Datagrid project will be reused. In particular, the same JDL based in ClassAds will be used with the necessary extensions the represent parallel jobs. They assume also the existence of Resource Brokers and Job Submission Services for sequential jobs, and basic middleware such us Condor-G, and Globus GRAM and DUROC modules. SA will take advantage of these available services in order to provide an early prototype that could be evaluated by the application developers in WP1. Later extensions or improvements would be added into SAs according to their feedback.

Linux is assumed to be the actual operating system. C and C++ will be the default implementation languages.

## 2.6. APPORTIONING OF REQUIREMENTS

A common scripting language will be chosen, if necessary.

All requirements are prioritized so that features with the highest priority will be developed first. Refer to next section for the detailed table with requirements and their priorities.

### 3. SPECIFIC REQUIREMENTS

Scheduling Agents will rely, in general, on EDG middleware, which also relies on Globus and Condor services. They will also rely on basic services used in the Workload Management Service of EDG, such as, the Job Description Language, the Condor ClassAd library and the Condor-G grid resource manager. Moreover, we expect to reuse some elements developed in the Workload Management Service of EDG. The use of some of these elements is mandatory because there is no equivalent task in CrossGrid to deal with them (for instance, the maintenance of the information about grid resources in ClassAd format). The use of common services should guarantee also, on the one hand, the compatibility of CrossGrid Scheduling Agents with DataGrid jobs (i.e. any DataGrid jobs submitted to a CrossGrid Scheduling Agent should run on a CrossGrid testbed just as any CrossGrid job). On the other hand, this approach will allow a fast development of initial prototypes, which will be a valuable tool to evaluate the features needed by application developers that have to be included in later versions.

Section 2.1 has described the rationale and the overall perspective of this package. Some of the main requirements for this package were presented by giving references to the particular application that use this package. A more detailed list of requirements follows. The table includes a number that identifies the requirement, a brief description and its priority (which can be understood as the severity of the requirement: high priority corresponds to features that are crucial to have in the first prototype, while medium corresponds to features that are nice to have or that are going to be included in later prototypes).

ID	Requirement Description	Priority
R-001	The Scheduling Agent shall receive commands from the Web Portals and send results back to them.	High
R-002	The Scheduling Agent shall send command output/results back to the Web Portals.	High
R-003	The Scheduling Agent shall send to the Resource Broker the description of the job to be submitted using a description language (e.g. JDL from EU DataGRID).	High
R-004	The Scheduling Agent shall receive from the Resource Broker the list of available resources matching the user request. It should be noted that the RB works both with lists of resources and with individual resources.	High
R-005	The Scheduling Agent shall submit commands to the JSS to run jobs over the selected resources.	High
R-006	The Scheduling Agent shall poll the status of running jobs querying the JSS.	High
R-007	The Scheduling Agent shall receive from JSS the status of running jobs.	High
R-008	The Scheduling Agent shall support parallel applications using MPI as communication library.	High
R-009	The Scheduling Agent shall log all the information related to the status of a running job onto the Logging & Bookkeeping server.	Medium

R-010	The Scheduling Agent shall be able to retrieve detailed info related to running jobs from the Logging & Bookkeeping server.	Medium
R-011	The Resource Broker shall get information from the information system about all available replicas of data and give it back to the Scheduling Agent, who will use it in the resource selection process.	Medium
R-012	The Scheduling Agent shall support different priority schemes for interactive (or high priority) applications and batch (or low priority) applications.	Medium
R-013	The Scheduling Agent shall support resource co-allocation for parallel jobs.	High
R-014	The Scheduling Agent shall perform the resource selection considering the network bandwidth, given that most of the applications state in their SRS that performance of network resources is critical.	Medium
R-015	The Scheduling Agent must support applications that require a large number of CE for executing. These CEs could belong to different sites.	Medium
R-016	The Scheduling Agent shall support composed jobs. Single components will be either sequential or parallel MPI applications. Components may exhibit ordering dependencies in terms of data files produced by one component that are used by other components.	Medium
R-017	The Resource Selection component of the Scheduling Agent shall consider that the application may require the reservation of unique resources such as medical scanners and 3D visualisation environments (required by the biomedical application).	Medium
R-018	The Scheduling Agent should allow advance resource reservation (required by the weather forecasting and air pollution modelling application).	Medium

### 3.1. EXTERNAL INTERFACES

The external interfaces of SA with other components of CrossGrid middleware are the following:

- 1. Interface with User Portal.** Jobs will be submitted to SA using JDL. A job identifier will be returned back if the job is accepted by the SA. This job identifier will be used later for any future operation regarding that particular job (in particular, it will be used to cancel the job).
- 2. Interface between SA and RB.** The job, described by its JDL file, will be passed by the SA to the RB. A list of CEs will be returned by the RB if the suitable resources are found. The RB will retrieve resources from the information services available in the VO (for instance, Globus MDS, EDG Ftree and EDG R-GMA).
- 3. Interface between SA and JSS.** The job, the list of resources and a submission attempt timeout will be passed by the SA to the JSS, who will carry out the actual submission of the job to the remote resources. All this information will be included in the JDL file of the job.

### 3.2. FUNCTIONS

As described in section 2.2, the main functions provided by this package are matchmaking, resource selection and submission service. A more detailed list of functions that are going to be supported by the package is included in the following table.

FUN-001	The Scheduling Agent shall manage user commands according to their types (monitoring or controlling commands).
FUN-002	The Scheduling Agent shall parse the job description submitted by the user.
FUN-003	The Scheduling Agent shall identify the constraints and the requirements related to a user job description and use them to rank/select the available resources.
FUN-004	The Scheduling Agent shall prepare commands for the JSS to run jobs over the selected resources.
FUN-005	The Scheduling Agent shall maintain a minimum amount of information about the state of a running job submitted by itself.
FUN-006	The Scheduling Agent shall maintain a permanent queue for all submitted jobs of the user it refers to.
FUN-007	The Scheduling Agent shall monitor the status of submitted jobs.
FUN-008	The Scheduling Agent shall automatically redo the resource selection, according to instructions contained in the user job description, in case the grid resource manager is not able to allocate all the resources specified initially.
FUN-009	The Resource Broker shall carry out single resource and multiple resource selection.

### 3.3. PERFORMANCE REQUIREMENTS.

- SA should be highly scalable and the same applies to the RB. Scalability refers both to the number of users and the number of jobs submitted by each individual user.
- The system comprised by SA and RB should submit jobs very quickly when resources are available and the rest of Grid components are working without failures and no significant delays are experienced in the network infrastructure. Otherwise, the system has to work with a reasonable delay. The term reasonable is necessarily ambiguous as it depends on the perception of the final user. Experience with first prototypes will be used to evaluate whether the normal performance of the system is acceptable or additional mechanisms are required in the case of Grid congestions.

### 3.4. LOGICAL DATABASE REQUIREMENTS

NA

### 3.5. DESIGN CONSTRAINTS

The CrossGrid project is supposed to adopt in the future Globus version 3.0. Thus, SA design should be compatible with OGSA specification and Web Services.

Furthermore, UML has been established as a standard modeling language in WP3.

The architecture design of Scheduling Agents is an open issue at this stage of the project. To some extent, their design is related to the overall architecture adopted at the CrossGrid project in terms of how Virtual Organisations are defined. Two are the main options that we plan to investigate.

1. **Tight design.** In this case, all the components of a SA are tightly integrated into a single component. This approach is close to the one adopted by the EDG project in which the Resource Broker is the central element that has the responsibility for all the allocation decisions and is also responsible for maintaining a queue with all the submitted jobs. Whether there is only one central RB for the entire EDG project, one for each VO or one for each user is still an open issue in the EDG project. The RB has been designed to be able to work with all configurations, but further research is required to find the right trade-off between the performance bottleneck problems that a central RB may suffer, and the lack of fair allocation that the use of a large distribution of RBs may exhibit.
2. **Modular design.** In this case, the components of a SA (which corresponds to a RB in the EDG case) will be split into separate subcomponents and their number and distribution within a VO is not going to be uniform. One of the approaches that we are going to investigate consists on having one centralised module at each VO responsible for the matchmaking and resource selection processes (in figures 2 and 3, we refer to this element as RB, but it is worth pointing out that it includes only part of the functionality of a RB according to EDG definitions. In addition to this RB, there will be separate copies for each user in the VO of the SA components that maintain the queue of jobs for that user and interacts with the personal copy of the grid resource manager. In figures 2 and 3, these two elements are identified as SA (the Scheduling Agent daemon that keeps track of all the jobs submitted by a given user, enforcing his policy criteria) and JSS/Condor\_G (the local resource manager service that interacts with the remote resources). This design will try to provide a fair allocation of resources between all users of the same VO and will try to avoid the problems related to a failure of a monolithic RB, as in the case of EDG. In the modular case, jobs already running will not be affected by a failure of the central matchmaking process. Only new jobs will be affected as users would not be able to submit because they could not be matched with available resources. In the tight case with a central RB per VO, jobs already running would eventually be affected by a failure of the central RB (and may finally fail or end up in an inconsistent way) as long as the grid resource manager is not going to be able to keep track of them. Figure 3 illustrates this design.

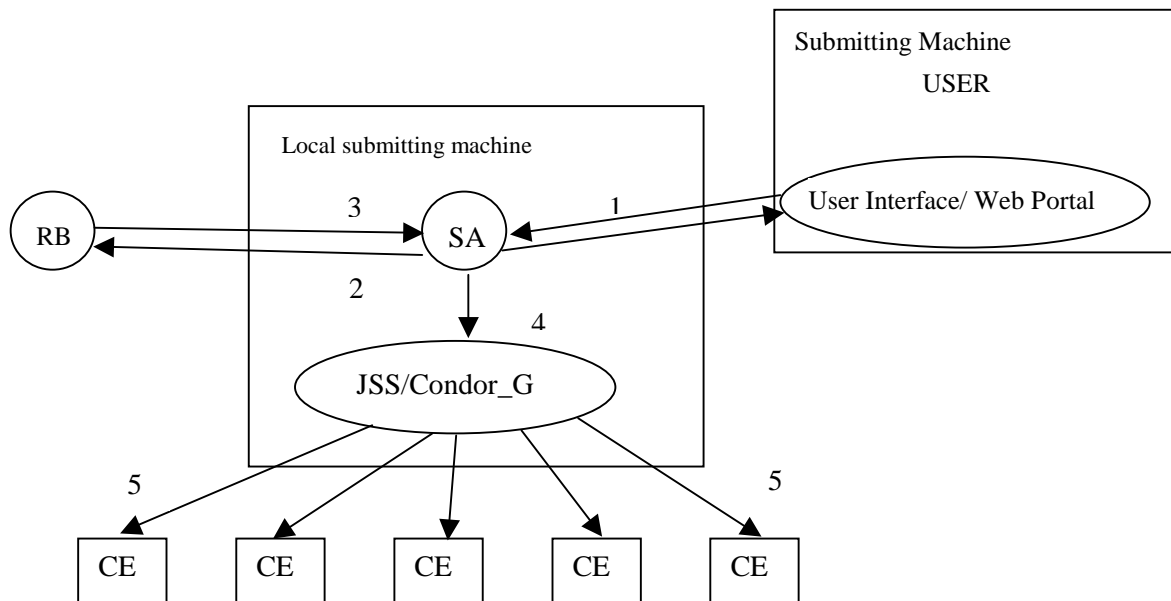


Figure 2. Grid resource management based on a modular design

In this scenario, the sequence of steps in the execution of a job is as follows:

- Step 1: The user will contact the SA through a remote connection (using the web portal interface, for instance) and will submit the job described in a JDL file.
- Step 2: The SA will pass the job description to the central RB
- Step 3: The RB and will answer with a list of suitable resources
- Step 4: The SA will take the list of resources passed by the RB. It will combine them with the JDL file and will create a suitable command that will be subsequently passed to the JSS service. In the case of MPI applications, SA will issue a set of Condor glidein commands before the job is finally passed to the JSS/Condor\_G service.
- Step 5: The JSS takes the responsibility to contact with the Computing Elements (CE) specified by the RB to start the application there. While the application is running, JSS will carry out also any further interaction with CEs or other intermediate elements in the Grid, such as Globus gatekeepers.

Users can control the execution of jobs by issuing cancellation commands to the SA at any moment. The response back from the SA to the User Interface is symbolised in figure 1 by the arrow going from SA to the User Interface module.

JSS and SA are coupled to a given user and there will be different copies of them for different users. Each pair of SA/JSS may run onto a different machine (which would normally correspond to the user's permanent machine where binaries and output files would be stored), but multiple users may share the same machine (although different instances of SA/JSS will be run for each one). Events generated by the different elements involved in the execution of a job will be recorded using the Logging and Bookkeeping service implemented in EDG. Users will be able to retrieve information about the current state of their jobs by issuing the appropriate command in their user interface, which will get the information for the Logging and Bookkeeping service. SA will carry out a simple monitoring of the job by issuing direct queries to the JSS. This simple monitoring will be used, for instance, to perform an automatic resubmission of the job in the case of a failure, if the failure is due to the Grid infrastructure and not the job itself. Automatic resubmission will be expressed as a user level option included in the JDL to control, for instance, how many automatic resubmissions the system has to try before the job is given back to the user.

It is worth mentioning that status of Grid resources will be always changing and, in principle, there is no guarantee that the list of resources selected by the RB will be available later on when either the SA or the JSS contacts them. Therefore, SA will set up different timeouts to deal with eventual problems that may occur if the status of a CE has changed or the information held by the RB was outdated. If the timeout expires SA will repeat steps 2 to 5 until success (the number of trials will be also a user option). In the case of resubmission, the SA will give to the RB information about the particular CEs that did not respond, so that the RB could take into account this information for latter matches.

As mentioned in section 2.1, advance resource reservation [8] would be a valuable service to guarantee a low response time for interactive applications that require a near real-time response. SA will take advantage of any available service of advance reservation provided by the underlying resource manager system. Depending on the reservation services it provides, we would investigate also the implementation of a reservation service at the RB level during the second year of the project. This service is not going to guarantee the availability of reserved resources because the RB does not have direct control over CEs. It would basically prevent reserved resources to be used as candidate resources in the matchmaking process. Reserved resources would be internally marked as unavailable in the RB until the reservation expiration time is reached. As the RB does not have direct control over CEs, this reservation mechanism would necessarily have a reduced functionality to avoid the introduction of significant inefficiencies in the usage of Grid resources.

A second design that can be used in latter stages of the project will include more than one RB within a Virtual Organisation (VO). They will be interconnected to exchange updated information about the state of Grid of resources, although this co-operation at the RB level will not preclude the use of timeouts at the SA level. Independently of the number of RBs, every user will have its own copy of the SA and JSS daemons. And every SA will interact only with a single RB. This design would also allows RBs from different VOs to exchange information about resources shared by multiple VOs as Figure 3 shows. In this example, there are two Virtual Organisations ( $VO_1$  and  $VO_2$ ) with their corresponding elements (CEs, RBs, and SA/JSS pairs). Some of the CEs are shared between  $VO_1$  and  $VO_2$ , which means that those resources could be allocated to jobs coming from either Virtual Organisation (in the case of simultaneous submissions of these resources, the local resource manager of each CE will allocate the CE either to a job from  $VO_1$  or from  $VO_2$ , according to the local policy of the CE). Figure 3 shows also the interconnection between RBs and the eventual existence of more than one RB per VO, but SA connected to only one RB within a VO ( $SA_1$  and  $SA_2$  are connected to  $RB_1$ , while  $SA_3$  is connected to  $RB_2$ , all in  $VO_1$ ).

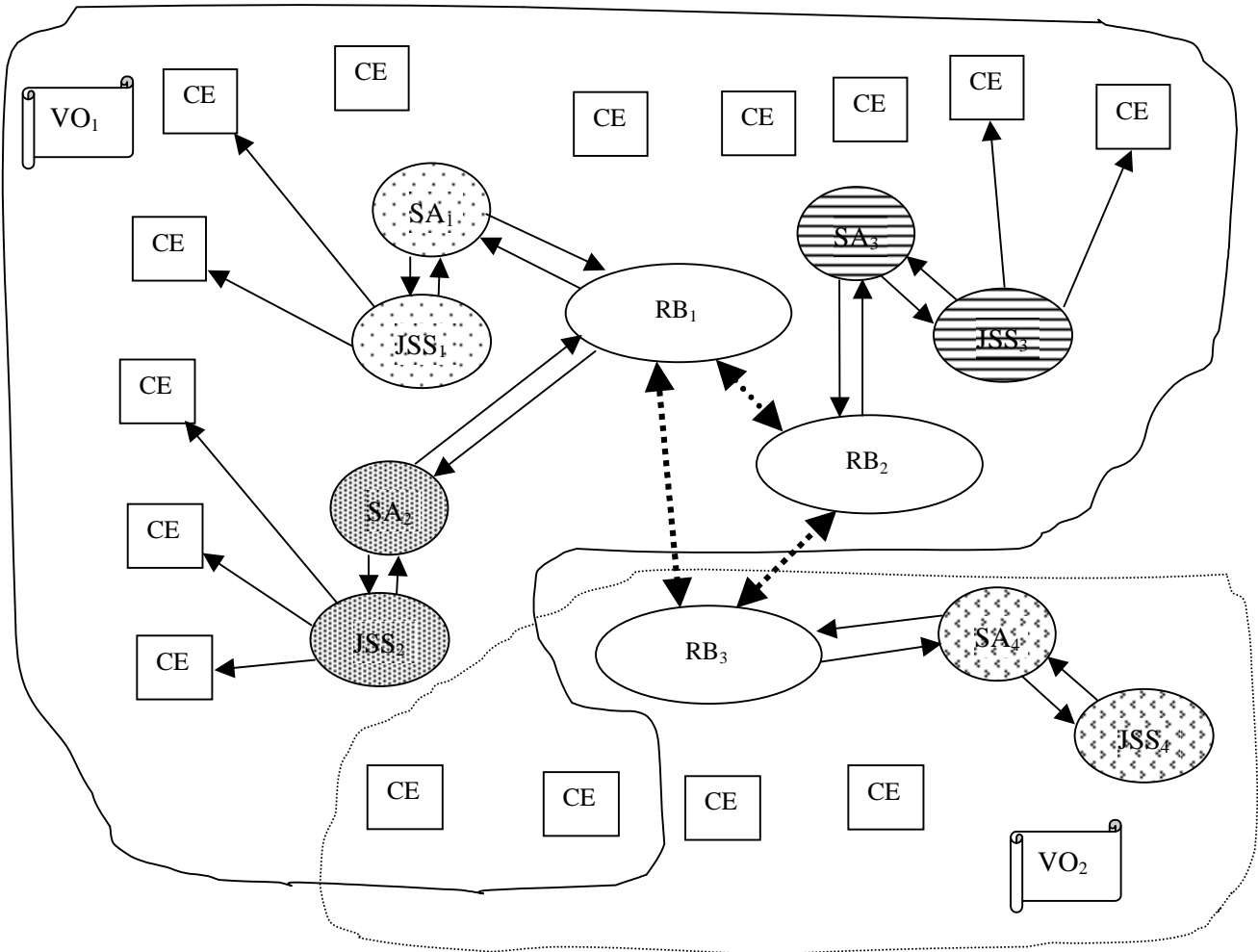


Figure 3. Organisation and interaction of SAs and RBs across multiple VOs.

Initial prototypes of SA, however, will be based on a tight design, as this approach is usually less error prone. The design will evolve to the modular option as the project progresses, basic services become more stable and the overall architecture of the CrossGrid is further refined.

### 3.6. STANDARDS COMPLIANCE

The system should be compliant with the following standards:

- UML
- SOAP
- WSDL

### 3.7. SOFTWARE SYSTEM ATTRIBUTES

The system has to exhibit the following attributes (the priority of these requirements is shown in brackets) :

- **Robustness** (high priority): there must be no single point of failure and non-critical failures in the rest of Grid components should have minimal impact. This means, for instance, that new jobs may not be accepted by a SA, but already running jobs would eventually keep running until completion. In the case of failure of the machine where the SA is running, jobs should be kept in a permanent list and they have to be reassumed (or resubmitted) once the machine is recovered.
- **Documentation must exist** (high priority). This includes a user guide, an installation guide, and a system guide with the information needed to continue software development.
- **Ease of use** (medium priority): it should be possible for an experienced system manager to install, configure and troubleshoot the software in a reasonable amount of time (a few hours).
- **Extra facilities** (medium priority): the system should have a command line interface for manual submission of jobs (without using the web portal).

## 4. APPENDIX A. DESCRIPTION OF COMPOSED JOBS USING CONDOR DAGS

This appendix contains a brief example of an Input File describing a DAG as used by Condor DAGman and described in the Condor Manual (v6.3). The whole description of a DAG includes additional features not presented here for the sake of simplicity. This appendix has been included for illustration and evaluation purposes, and to start an open discussion about the need and convenience for adopting a similar notation to represent CrossGrid composed jobs. It is expected that WP1 developers should read it and make comments about the suitability of this syntax to express their composed jobs. To our knowledge, the EDG project is also evaluating the use of some kind of DAG notation for DataGrid jobs that can be easily decomposable into independent jobs with interdependencies between them.

A full description of the syntax will be provided when a final notation will be agreed.

The input file used of a DAG specifies three items:

1. A list of the programs in the DAG. This serves to name each program and specify each program's Condor submit description file.
2. Processing that takes place before submission of any program in the DAG to Condor or after Condor has completed execution of any program in the DAG.
3. Description of the dependencies in the DAG.

An example input file for DAGMan is

```
# Filename: diamond.dag
#
Job A A.condor
Job B B.condor
Job C C.condor
Job D D.condor
Script PRE A top_pre.csh
Script PRE B mid_pre.perl
Script POST B mid_post.perl
Script PRE C mid_pre.perl
Script POST C mid_post.perl
Script PRE D bot_pre.csh
PARENT A CHILD B C
PARENT B C CHILD D
```

Each program (single application according to notation of section 2.1) is described by a single line called a Job Entry, identified by the JOB keyword.

The `SCRIPT` keyword identifies scripts that have to be processed in the submission machine either before a program is submitted to Condor (PRE script) or after a program successfully completes its execution under Condor (POST script).

The third type of item in the DAG input file describes the dependencies within the DAG. Nodes are parents and/or children within the DAG. A parent node must be completed successfully before any child node may be started. A child node is started once all its parents have successfully completed.

## 5. APPENDIX B. RELATED WORK

Grid infrastructure projects have provided many of the services required for applications that can benefit from the rich set of resources available in Grid environments. However, these efforts have shown also that taking advantage of these resources still requires extensive support from the middleware layers. Resource management in Grid environments is one of the challenging problems that has received special attention in the past years. There are different activities that are related with the term *resource management*. They include resource discovery, process scheduling and job execution. Different Grid projects provide different solutions to these issues. An overall overview of all these solutions can be obtained in [10]. In this appendix, we review the most relevant efforts that have been carried out in terms of process scheduling. However, we also describe some systems such as Condor-G which are not schedulers from a strict point of view, but systems that manage the job execution.

The **AppLeS** [1] framework guides the implementation of application-specific scheduler logic, which determines and actuates a schedule customised for the individual application and the target computational Grid environment. In contrast to the CrossGrid approach, the user has to provide a model that describes the main characteristics of the application from the performance perspective. This model is later combined with information regarding the future performance of Grid resources. And, as a result, a whole scheduling is obtained, which can be used to control the execution of the application at run-time. In particular, AppLeS has been normally used together with the NWS (Network Weather Service) [14] which is a service that periodically monitors and dynamically forecasts the performance various network and computational resources can deliver over a given time interval. Most remarkably AppLeS efforts have targeted master-slave applications [7,11].

**ARMS** [4] is an Agent-based Resource Management System for grid computing, which couples the performance prediction of the PACE toolkit with a scheduling algorithm designed to manage a local grid resource. ARMS uses agents which are viewed as representative of grid resources. The information of each resource can be advertised within an agent hierarchy and agents can cooperate with each other to discover available resources. This resource discovery mechanism significantly differs from other grid approaches (such as those based on Globus) that utilise query-based mechanisms for resource discovery and advertisement, and that have been adopted, for instance, by the CrossGrid and DataGrid projects.

The **Condor-G** system [8] is a grid resource manager that provides an API and command line tools that allow the user to perform the following job management operations:

- submit jobs, indicating an executable name, input/output files and arguments;
- query a job's status, or cancel the job;
- be informed of job termination or problems, via callbacks or asynchronous mechanisms;
- obtain access to detailed logs, providing a complete history of their jobs' execution.

It has been designed primarily as a reliable front-end to a computational grid. It provides job monitoring, logging, notification, policy enforcement, fault tolerance, credential management, and it can handle complex job-interdependencies. Condor-G acts as a grid resource manager by exploiting GRAM, GASS, MDS-2, and GSI services included in the Globus toolkit. Condor-G

functionalities are provided by a personal desktop agent and supported in a Grid environment, while guaranteeing fault tolerance and exactly-once execution semantics. It is important to point out that Condor-G does not address the important question of how to determine where to execute user jobs. This role is left to a higher level authority. CrossGrid Scheduling Agents will be designed to address this problem and will use Condor-G as a reliable grid resource manager.

The **Datagrid Workload Management System** [9] is a whole system currently under development in the framework of the DataGrid project that comprises several elements responsible for resource selection and scheduling. CrossGrid Scheduling Agents have several connection points with it and we will mention them below. The whole system comprises four components: the User Interface, the Resource Broker, the Job Submission Service, and the Logging and Bookkeeping Service. There are several similarities between DataGrid's approach and CrossGrid approach to solve the management of jobs and resources. Both projects base their resource management services on two key components: ClassAds and Condor\_G. The ClassAds library is the foundation to represent jobs (in JDL), resources and carry out the matchmaking between jobs and resources. Condor\_G is the core element in the Job Submission Service. Condor\_G is responsible to manage the job submission, interfacing with Computing Elements, and providing several recovery mechanisms from failures in different Grid components. The Resource Broker is the core element in the workload management system. It finds a Computing Element for a submitted job and passes it to the JSS. It also keeps a permanent queue with all handled jobs. In the CrossGrid project these functions the functionality of EDG Resource Broker have been split into the Scheduling Agent and the Resource Broker which are also organised in a different way. The CrossGrid Resource Broker is only responsible for matchmaking and there are very few copies in a given VO. Another difference with respect to EDG is that matchmaking in the CrossGrid case will require to extend the Condor matchmaking framework to support both single resource and multiple resource selection. Regarding SAs, there is one of them per user. Each SA will keep the permanent information about all user jobs and it will ensure that all necessary resources are co-allocated before passing a parallel job to Condor\_G. It will also be responsible to manage the available resources in a way that guarantees that high priority jobs (i.e. interactive) are able to run as soon as possible, while less priority jobs are either temporarily vacated or suspended for a latter resubmission. The EDG User Interface is command-line based, while CrossGrid will be based on a web interface. However, as no Logging and Bookkeeping service is being developed in CrossGrid, SAs will generate logging messages compatible with the EDG Logging and Bookkeeping service.

**NetSolve** [5] is a client-server system that enables users to solve complex scientific problems remotely and it . The system allows users to solve particular problems by accessing remote libraries installed **across** a network. NetSolve client library is linked in with the user's application and then the application makes calls to NetSolve for specific service. NetSolve searches for computational resources on a network with the appropriate services installed and runs them with user supplied input parameters. In contrast to the Crossgrid project, Netsolve does not address the problem of submitting arbitrarily user applications to run on a grid environment. Only preinstalled libraries can be accessed through Netsolve and there is some limited support for parallel applications, which is based on a set of simple API functions and an adaptive scheduling mechanism for task-farming applications [6].

**Nimrod-G** [2] is a tool for automated modelling and execution of parameter sweep applications (parameter studies) over global computational grids. It provides a simple declarative modelling language for expressing parametric experiments. One of the key components of Nimrod-G is the

Nimrod-G Resource Broker. It is basically based on a Task-Farming Engine (TFE) that constitutes the co-ordination point for processes performing resource trading, scheduling, data and executable staging, remote execution, and result collation, and a Scheduler that performs resource discovery, trading and scheduling. In addition to the current Scheduler, TFE enables also "plugging" of user defined schedulers.

Resource management and scheduling algorithms in Nimrod-G are based on economic principles [3]. Specifically, it requires that each user job has a user-defined deadline and budget constraints for schedule optimisations. With this information, for instance, the broker may try to meet the deadline (completion time) using cheap resources as much as possible. More expensive machines might be added later if the broker predicts that the deadline would not be met with the current set of resources. This economy based scheduling approach, however, is only feasible in the presence of highly predictable farmer applications that can be decomposed in a fixed number of small tasks each of which exhibiting a similar execution time. In general, CrossGrid applications don't fulfil these requirements: some of them can not be expressed as sweep parameter applications, and no information is provided by the users about the expected execution time of each application.

**Prophet** [12,13] is a run-time scheduling system that, similarly to the AppLeS approach, exploits application structure and system resource information to promote application performance. It was demonstrated originally in heterogeneous, local-area networks, although its concepts could be adapted to Grid environments. It's a scheduler tight with parallel applications written in the Mentat programming language, a fact that limits its usability. Prophet was initially used with both SPMD applications and applications based on task-parallel pipelines.

## References:

1. Fran Berman, Richard Wolski, Silvia Figueira, Jennifer Schopf, and Gary Shao "*Application-Level Scheduling on Distributed Heterogeneous Networks*", Proceedings of Supercomputing (1996).
2. Rajkumar Buyya, David Abramson, Jonathan Giddy, "*Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*", The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), Beijing, China. IEEE Computer Society Press, USA, 2000.
3. Rajkumar Buyya, David Abramson, and Jonathan Giddy, "*A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker*", Future Generation Computer Systems (FGCS) Journal, Elsevier Science, The Netherlands, 2002 (to appear).
4. J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd, "*ARMS: and agent-based resource management system for grid computing*", To appear in Scientific Programming, Special Issue on Grid Computing, IOS Press (2002).
5. Henri Casanova and Jack Dongarra. "*NetSolve: A Network Server for Solving Computational Science Problems*", The International Journal of Supercomputer Applications and High Performance Computing", Volume 11, Number 3, pp 212-223, 1997.
6. Henri Casanova, Myung-ho Kim, Jim Plank and Jack Dongarra, "*Adaptive Scheduling for Task Farming with Grid Middleware*", The International Journal of Supercomputer Applications and High Performance Computing, Vol. 13, pp. 231-240, 1999.
7. H. Casanova, G. Obertelli, F. Berman and R. Wolski, "*The AppLeS Parameter Sweep Template: User-level middleware for the Grid*". In Proceedings of Supercomputing (November 2000).

8. James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke, "*Condor-G: A Computation Management Agent for Multi-Institutional Grids*", Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10) San Francisco, California, August 7-9, 2001.
9. Francesco Giacomini and Francesco Perlz, "*Definition of Architecture, Technical Plan and Evaluation Criteria for Scheduling, Resource Management, Security and Job Description*", DataGrid Deliverable D1.2-0112-0-3, September, 2001.
10. Klaus Krauter, Rajkumar Buyya, and Muthucumar Maheswaran, "A *Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing*", Software: Practice and Experience (SPE) Journal, ISSN:0038-0644, Volume 32, Issue 2, 2002, Wiley Press, USA, February 2002.
11. G. Shao, R. Wolski and F. Berman, "*Master/slave computing in the Grid*", In proceedings of the 9<sup>th</sup> Heterogeneous Computing Workshop (May 2000).
12. J. Weissman, "*Prophet: Automated scheduling of SPMD programs in workstation networks*", Concurrency: Practice and Experience, 11, 6 (1999).
13. J. Weissman and X. Zhao, "*Scheduling parallel applications in distributed networks*", Journal of Cluster Computing 1, 1 (1998) 109-118.
14. Rich Wolski, Neil Spring, and Jim Hayes, "*The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*", Journal of Future Generation Computing Systems, Volume 15, Numbers 5-6, pp. 757-768, October, 1999.

## 6. INDEX

### *C*

ClassAd .....	5, 9, 10, 12
Computing Elements .....	4, 13
Condor.....	4, 5, 8, 10, 11, 12, 13, 17, 18
Condor-G .....	4, 5, 10, 11, 12

### *D*

DAG.....	5, 7, 17, 18
DataGrid.....	5, 12, 17
Directed Acyclic Graph.....	5, 7

### *E*

EDG .....	5, 9, 12, 14, 16, 17
-----------	----------------------

### *G*

GIS .....	5
Globus .....	5, 11, 13, 16
GRAM.....	5, 11

### *J*

JDL.....	5, 7, 9, 11, 13, 14, 16
Job Description Language.....	5, 7, 8, 12. <i>See JDL</i>
Job Submission Service.....	5
job-ad .....	5, 8
JSS.....	5, 9, 12, 13, 14, 16

### *M*

<b>Matchmaking Process</b> .....	10, 11
MDS .....	5, 16
MPI.....	5, 7, 11, 13

### *R*

RB .....	5, 12, 13, 14, 16
Resource Broker .....	5, 11, 12
resource manager .....	4, 5, 8, 9, 10, 11, 12, 14
<b>Resource Selection</b> .....	10

### *S*

SA .....	4, 5, 7, 8, 9, 11, 12, 13, 14, 15, 16
Scheduling Agents .....	1, 4, 5, 6, 7, 8, 9, 10, 11, 12
<b>Submission Service</b> .....	10

### *W*

WMS .....	5, 7
-----------	------