MSF: A Workflow Service Infrastructure for Computational Grid Environments

Seogchan Hwang[1] and Jaeyoung Choi[2]

[1] Supercomputing Center, Korea Institute of Science and Technology Information,
52 Eoeun-dong, Yusung-gu, Daejun 305-306, Korea
seogchan@kisti.re.kr

[2] School of Computing, Soongsil University,
1-1 Sangdo-5dong, Dongjak-gu, Seoul 156-743, Korea
choi@ssu.ac.kr

Abstract. Globus has become a standard in the construction of Grid computing environments. However, it still needs more work and research to satisfy requirements from various grid applications such as workflow services. We propose a Meta Scheduling Framework (MSF) for workflow service management based on the Globus toolkit. The MSF provides an XML-based Job Control Markup Language (JCML) for describing information and procedures of applications including dependencies of jobs, a workflow management engine for scheduling complicated job flow, and an execution environment without the need for code modification or wrapping.

1 Introduction

Grid computing [1] is a new infrastructure which provides computing environments for grand challenge problems by sharing large-scale resources. The Globus toolkit is a standard in constructing a Grid and provides essential grid services such as security, resource management, data transfer, information service, and so on. However, it still needs more work and research to satisfy the requirements of various grid applications. Workflow management is emerging as one of the most important grid services, yet it is difficult to use the grid resources for general applications because of various characteristics such as heterogeneity and dynamic organization. Numerous research groups have been working on workflow related projects.

GridFlow [2] is a workflow management system, which uses agent-based resource management and a local resource scheduling system, Titan. It focuses on the scheduling of time-critical grid applications in a cross-domain and highly dynamic grid environment by using a fuzzy timing technique and performance prediction of application. MyGrid [3] provides services such as resource discovery, workflow enactment, and distributed query processing for integration. It is a research project middleware to support biological environments on a Grid. And Condor [4] provides a workload management system for compute-intensive jobs and a scheduling of dependencies between jobs using DAGMan. This project provides similar functionalities but requires its own specific infrastructure.

In this paper, we introduce a system, called Meta Scheduling Framework (MSF), for grid computational environments. MSF provides a Job Control Markup Language that is able to specify the job flow for general applications which are not developed for grid environments. It also provides a workflow management service, based on Globus, to execute the job, and a graphical user interface to facilitate the composition of grid workflow elements and access to additional grid resources

In section 2, we illustrate the structure of the MSF and its components in detail. Section 3 describes the implementation of the system and examines a sample application, AutoDock. Finally, we discuss the conclusion of the research in Section 4.

2 Meta Scheduling Framework

Usually grid applications require complicated processing steps, for instance, when a task requires an input file which is the output of another task, and the two tasks are simultaneously running in separate computing nodes. Therefore, a job description language is essential to describe dependencies among jobs, and a management system is required to control the flow of tasks.

The goal of this research is to develop a framework to provide a workflow service to applications using the Globus. To accomplish this, we designed a workflow de

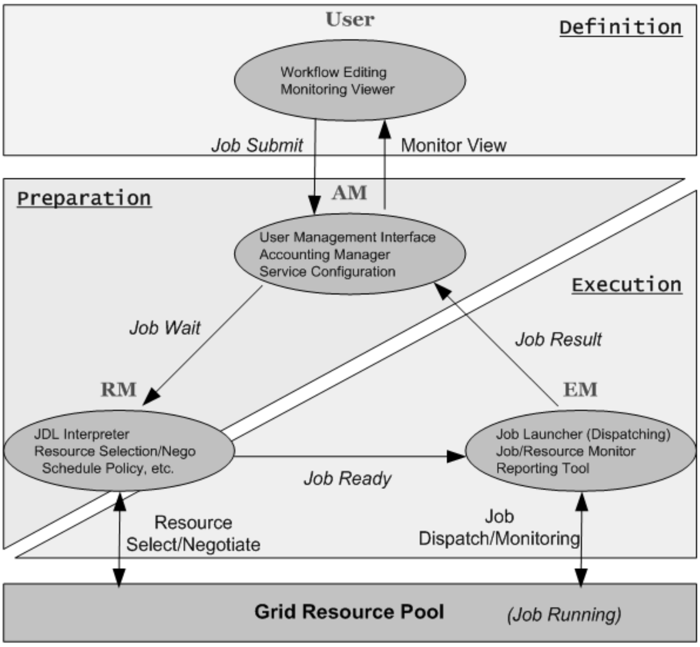


Fig. 1. Meta Scheduling Framework Architecture

scription language, called Job Control Markup Language (JCML), and a workflow management system. The JCML is designed to describe a process of tasks. And the workflow management system provides services to control the flow of tasks.

The architecture of MSF is depicted in Figure 1. A user describes a job flow using the MSF console. The Access Manager (AM) provides services which include user authentication, environment setup, and a job submission service. The Resource Manager (RM) provides resource discovery and matching. The Execution Manager (EM) provides job launching, monitoring, and reporting. More details are described in Section 2.3.

MSF consists of three phases: definition, preparation, and execution. During the definition phase, jobs are defined to specify a Job Definition List (JDL), which describes a task flow using JCML. In this phase, users connect to the AM for authentication of the MSF and then the AM creates a user proxy for globus. In the preparation phase, resources are searched and assigned to the matched tasks. Next, the AM creates an agent to provide a proxy service to the user. The agent then passes the JDL and traces the status of the job. The RM receives the JDL from the AM and analyzes it. After finding appropriate grid resources for the job, the RM then assigns them to tasks and generates a worklist that includes information on activities and their executing order. Finally, during the execution phase, the tasks on the worklist are executed, the status is monitored, and the results are reported.

2.1 Job Control Markup Language

A job description language to specify the flow of an application task must provide a way to describe various grid environments and task information including execution environments, arguments, sequence, data dependency, prefetching, and so on. The JCML is a workflow description language based on the Graph eXchange Language (GXL) [5], which defines a graph-based XML representation for specifying the dependencies among components. The JCML consists of four major elements: Info, Resource, Component, and Dependency. Figure 2 shows the JCML structure.

Info: This element lists the document name, scope, target namespaces, authoring date, and so on.
Resource: This element describes the resources of hardware and software required to execute a job. The hardware includes architecture, CPU, memory, disk, and network bandwidth. The software includes an operating system, installed application, and local scheduler. The Time represents the deadline to be executed.
Component: This element lists all of the task-related information. A Node is an executing program or computer in the workflow. A node is classified into a task and a resource. A task node is an executing program in workflow, while a resource node is an assistant, which supports the task node, and represents the physical computing resources, such as storage and database. While a task node includes execution file, input, output, arguments, and resource configuration, a resource node includes data location and the access method. If it is necessary to refer to a series of nodes as a

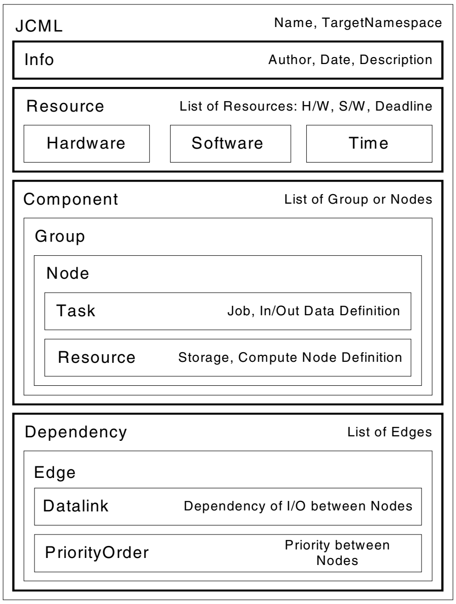


Fig. 2. JCML Structure

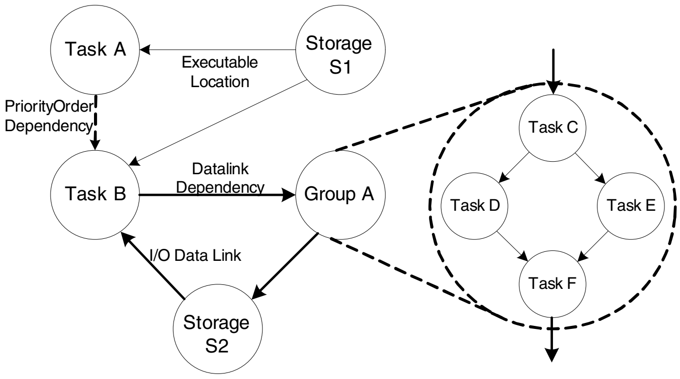


Fig. 3. Example model of JCML

single node according to a job flow logic, a group is logically a node. One group can include another group(s). Figure 3 illustrates an example model of JCML. A circle and a dotted circle represent a node and a group, respectively. The example is composed of two storage nodes, Storages S1 and S2; two task nodes, Tasks A and B; and a group, Group A, which includes four task nodes: Tasks A, B, C, and D.

Dependency: This element describes the dependencies of a workflow. Each line is an edge which represents an execution order and a dependency between two objects (node or group). It has two types of links, PriorityOrder and Datalink, which have a direction that expresses a starting point and an end point of linked nodes. The PriorityOrder just represents an execution order between the two linked nodes. The Datalink displays a flow of data which is used for the input or output file of each task.

As shown in Figure 3, each line represents an edge between the linked nodes, and all edges have directions. A PriorityOrder expressed in a dotted line is used for task nodes, but does not include resource nodes. Tasks A and B are linked to Storage S1 with solid lines, which imply the location of execution files. When the task is running, the execution file is transferred to a computing node selected by the scheduler. The edge which links Task B and Group A represents a dependency between two nodes, transferring the output of Task B to the input of Group A. The edge linking with Storage S2 shows that Task B reads the input from S2 and Group A writes the output to S2. Next, Task C of Group A receives the input from Task B and sends the output to Tasks D and E. Task F then receives two input files from Tasks D and E and finally writes the output to S2.

2.2 Workflow Management Systems

A workflow management system guarantees that a flow of tasks will be executed exactly. MSF provides a workflow service, which is a scheduler-based paradigm [6] based on a state transition machine.

Figure 4 illustrates the processing steps of a job in the workflow engine. A job is processed by a workflow management system as follows: analyzing the job, mapping resources, generating a worklist, and scheduling tasks. A user specifies a JDL to execute a grid application. After analyzing the JDL, the JDL Analyzer searches resources and assigns resources to the job, selected by the Resource Discovery and Selector, which uses an external information service such as MDS [7] and NWS [8], and then generates the worklist. The worklist has the execution information of the task

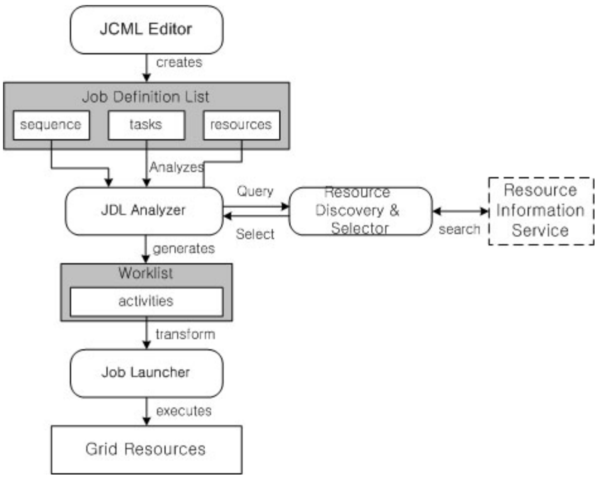


Fig. 4. Workflow Management Architecture

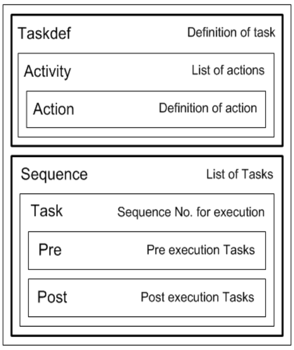


Fig. 5. Worklist Structure

and its execution order. Each task consists of activities. Figure 5 shows the worklist structure.

The Taskdef describes the behavior of a task, or a list of activities (actions) such as file copy or running programs, which all have an execution order. The Sequence describes the sequence information of Taskdef. The tasks, each with their own sequence numbers, are executed according to their sequence number and their causal ordering specified in Pre and Post in Task.

```
<node id="d" type="applicationJob">
  <excutable name="mkdpf3" location="/usr/local/autodock/share/" arguments=""/>
  <application name="autodock" version="3.0.1" />
  <configuration>
    <env name="workDir" value="/home/seventy9/queue/job"/>
    <envname="PATH" value="/bin:/user/bin:/bin/usr/local/autodock/bin:/usr/local/
 autodock/share"/>
  </configuration>
  <data>
    <in id="dia" name="l.pdbq" location="" cmdArg="" link="yes"/>
    <in id="dib" name="p.pdbqs" location="" cmdArg="" link="yes"/>
  </data>
</node>
<edge id="bd" from="b" to="d" direction="directed" type="datalink"></edge>
<edge id="de" from="d" to="e" direction="directed" type="datalink"></edge>
```

(a) Job Definition List

```
<action id="3">
  <execution>
    <resource>
      <host name="ironfly.ssu.ac.kr" port="2119"/>
      <workDir path="/home/seventy9/queue/job"/>
      <env name="PATH" value="/bin:/usr/bin:
          /bin/usr/local/autodock/bin:/usr/local/autodock/share"/>
    </resource>
    <command>
      /usr/local/autodock/share/mkdpf3 l.pdbq p.pdbqs
    </command>
  </execution>
</action>
```

(b) Worklist

```
&(directory="/home/seventy9/queue/job")
 (arguments="l.pdbq""p.pdbqs")
 (executable = "/usr/local/autodock/share/mkdpf3")
 (environment=("PATH" "/bin:/user/bin:/bin/usr/local/autodock/bin:
  /usr/local/autodock/share"))
```

(c) Globus RSL

Fig. 6. Three Job Description Formats in MSF

The Job Launcher executes activities according to the worklist. At the workflow engine, it is required to transform activities in the worklist to RSL [9] type in order to execute the real task in a local grid scheduler. Figure 6, for example, shows the three job description formats in MSF: (a) a partial definition of a node and an edge in JDL, (b) an action of a Taskdef in worklist, and (c) RSL command.

2.3 Components Architecture

The MSF, as shown in Figure 7, has four major components: (a) Access Manager (AM) manages the user connection and requests, (b) Resource Manager (RM) is in charge of resource discovery and assignment, (c) Execution Manager (EM) controls workflow, and (d) Information Manager (IM) provides all event data to other system components.

The main purpose of AM is to manage the connection between clients and the MSF. If a client connects to the MSF, then the AM accepts it and performs authentication and authorization of the user. If the client is verified, the Master Agent creates a new User Agent. The User Agent manages the client session and performs some functions: submitting a job, tracking the status of a client request, and providing a service configuration.

RM provides two functions, resource discovery and allocation. The JDL Interpreter analyzes the JDL received from AM and forwards the resource information to the Resource Collector, which searches resources using an external Grid Information

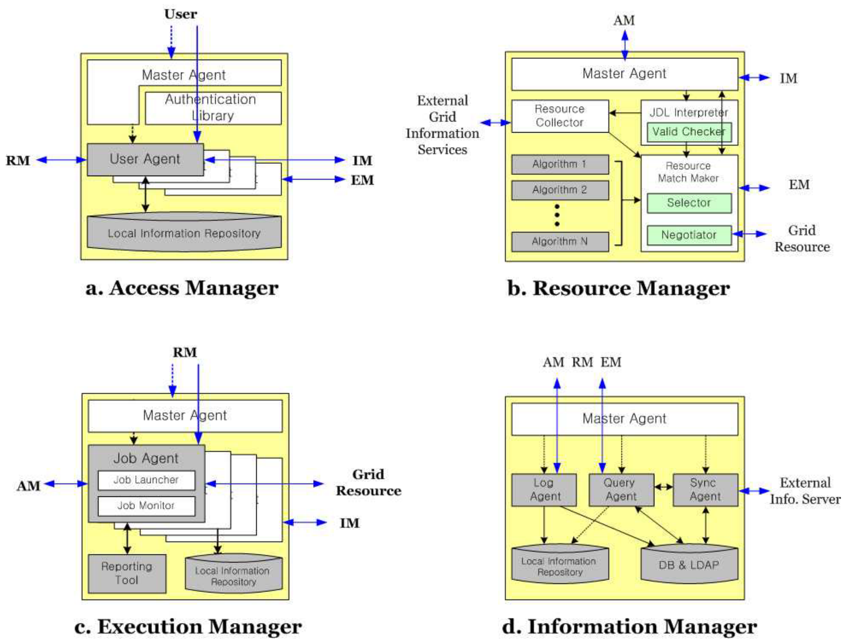


Fig. 7. MSF Components Architecture

Service. The Resource Match Maker selects appropriative resources for the job using a matchmaking algorithm, which can be replaced with another in a plug-in manner. The selected resources are then negotiated by the Negotiator. Finally, the RM generates a worklist, and passes it to the EM.

The EM carries out job launching, monitoring, and reporting. A Master Agent creates a Job Agent corresponding to the job. The Job Agent then assigns the activities of the tasks to selected resources according to the order of the worklist, which monitors the status of running tasks and reports of the result of each task.

The IM controls the event log data from the other components and consists of three agents: the Log Agent which records all events from other components; the Query Agent which searches for the requested query; and the Sync Agent which synchronizes the data with other IM, if necessary.

3 Implementation and Examination

We developed a prototype of MSF using pure Java (JDK 1.4.0) and Java CoG Kit (version 0.9.13) on Globus 2. We also implemented and executed a Virtual Screening on MSF. A Virtual Screening is one of the design methods, called *docking* in Bioinformatics, which combines one protein with many, sometimes hundreds of thousands of ligands to discover candidates for new drugs. In order to execute docking, the format of the material must be changed. We chose the AutoDock [10] application for this experiment. Figure 8(a) shows the processing steps involved in transforming the format of the protein and the ligand for docking in AutoDock. The dotted square represents a process for protein transformation, which is executed just once during the virtual screening. Figure 8(b) illustrates the JCML process to make a workflow used by MSF for AutoDock processing. A circle represents a program which transforms each format according to the AutoDock processing steps. The programs of babel, mol2topdbq, mkgpf3, and mkpdf3, are commands which transform to the format of mol2, PDBQ, GPF, and DPF, respectively. And finally autogrid3 and autodock3 calculate the docking energy of the two materials, the protein and the ligand.

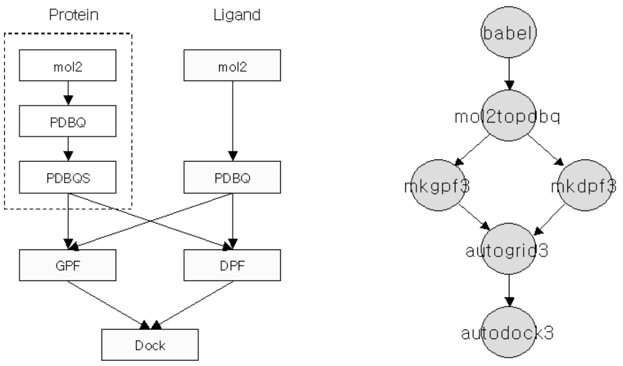


Fig. 8. (a) AutoDock Procedure Steps (b) JCML Process Model

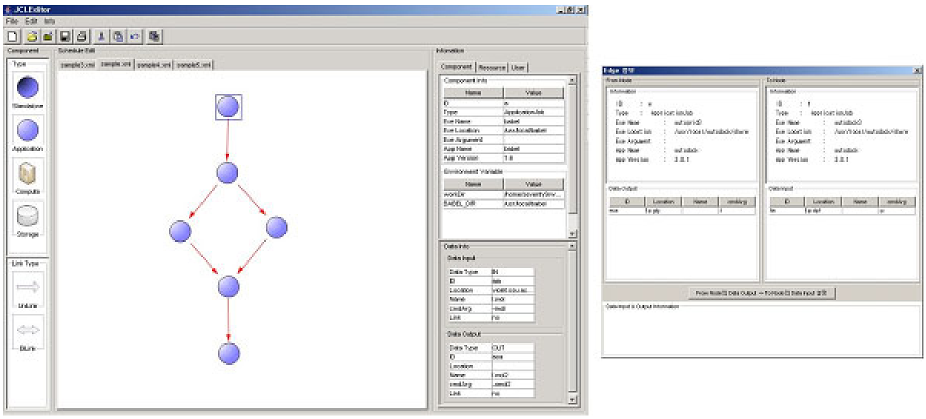
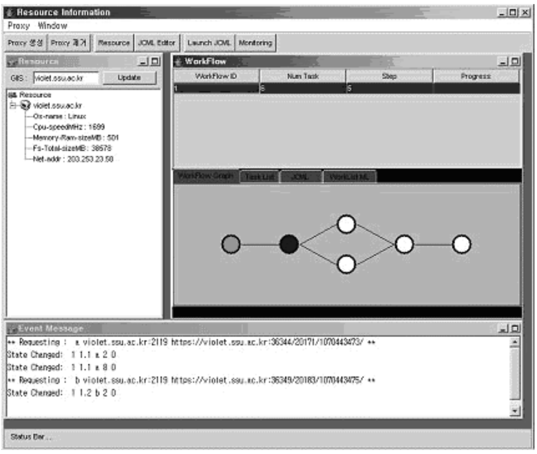


Fig. 9. JCML Editing Windows: (a) Main Window, (b) Edge Window



GIS : violet.ssu.ac.kr

```
Resource
  violet.ssu.ac.kr
    Os-name : Linux
    Cpu-speedMHz : 1699
    Memory-Ram-sizeMB : 501
    Fs-Total-sizeMB : 38578
    Net-addr : 203.253.23.58
```

WorkFlow ID	Num Task	Step	Progress
1	6	5	

```
** Requesting :  a violet.ssu.ac.kr:2119 https://violet.ssu.ac.kr:36344/20171/1070443473/ **
State Changed:  1 1.1 a 2 0
State Changed:  1 1.1 a 8 0
** Requesting :  b violet.ssu.ac.kr:2119 https://violet.ssu.ac.kr:36349/20183/1070443475/ **
State Changed:  1 1.2 b 2 0
```

Fig. 10. MSF main console

After JCML modeling, it is required that the job flow be specified. Figure 9 displays an editing window for the AutoDock. We drew a workflow for AutoDock using the task and edge icons, and specified the task information in the right side of the main window. We also set up the dependencies among nodes in the edge window.

Figure 10 displays the main console which monitors the status of the job flow. The main console has three windows: a resource window which shows the current state of the selected resource, a workflow window which illustrates the status of each task in the workflow, and an event window which displays the event from the task.

4 Conclusion

In this paper, we described the architecture of a Meta Scheduling Framework. The main purpose of a MSF is to provide a workflow service for general applications in a Grid environment. Therefore, we designed and implemented a workflow description language, JCML, to describe the flow of application, including complexity and dependencies, and a workflow management system to execute and monitor this flow.

Currently, we are working to extend the architecture in order to enhance efficiency and availability and to describe jobs in greater detail with JCML. In addition, a new version of Globus 3.0, OGSA [12], has been released, which integrates scientific and enterprise environments based on web service. The OGSA platform will be supported by MSF in the near future.

References

1. I. Foster and C. Kesselman, ed., The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, (1998).
2. J. Cao, S. A. Jarvis, S. Saini and G. R. Nudd, GridFlow: Workflow Management for Grid Computing, 3rd International Symposium on Cluster Computing and the Grid, (2003), 12-15.
3. R. Stevens, A. Robinson and C. Goble, myGrid: personalized bioinformatics on the information grid, Bioinformatics, 19(1), (2003), 302-304.
4. M. Litzkow, M. Livny and M. Mutka, Condor - A Hunter of Idle Workstations, 8th International Conference of Distributed Computing Systems, (1998), 13-17.
5. A. Winter, B. Kullbach and V. Riediger, An Overview of the GXL Graph Exchange Language, Software Visualization. Lecture Notes in Computer Science, Vol. 2269, (2002), 324-336.
6. A. Cichocki, A. S. Helal, M. Rusinkiewicz and D. Woelk, eds., Workflow and Process Automation: Concepts and Technology, Kluwer, (1998).
7. K. Czajkowski, S. Fitzgerald, I. Foster and C. Kesselman, Grid Information Services for Distributed Resource Sharing, 10th International Symposium on High-Performance Distributed Computing, (2001), 181-184.
8. R. Wolski, N. Spring and J. Hayes, The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, Future Generation Computing Systems, 15(5/6), (1999), 757-768.
9. The Globus Resource Specification Language, http://www.globus.org/gram/rsl_spec1.html.
10. AutoDock, http://www.scripps.edu/pub/olson-web/autodock/.
11. Python Molecule Viewer, http://www.scripps.edu/~sanner/python/pmv/webpmv.html.
12. I. Foster, C. Kesselman, J. Nick and S.Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG. Global Grid Forum, (2002).