Gridbus Workflow Enactment Engine

Jia Yu and Rajkumar Buyya

Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
Email: {jiayu, raj@}csse.unimelb.edu.au

1. Introduction

With the advent of Grid technologies, scientists and engineers are building complex and sophisticated applications to manage and process large data sets, and execute scientific experiments on distributed Grid resources [33]. Building complex workflows requires means for composing and executing distributed applications. A workflow expresses an automation of procedures wherein files and data are passed between procedures applications according to a defined set of rules, to achieve an overall goal [13]. A workflow management system defines, manages and executes workflows on computing resources. The use of the workflow paradigm for application composition on Grids offers several advantages [22] such as:

- Ability to build dynamic applications and orchestrate the use of distributed resources.
- Utilization of resources that are located in a suitable domain to increase throughput or reduce execution costs.
- Execution spanning multiple administrative domains to obtain specific processing capabilities.
- Integration of multiple teams involved in managing different parts of the experiment workflow – thus promoting inter-organizational collaborations.

Executing a Grid workflow application is a complex endeavor. Workflow tasks are expected to be executed on heterogeneous resources which may be geographically distributed. Different resources may be involved in the execution of one workflow. For example, in a scientific experiment, one needs to acquire data from an instrument, and analyze it on resources owned by other organizations, in sequence or in parallel with other tasks. Therefore, discovery and selection of resources for executing

workflow tasks could be quite complicated. In addition, a large number of tasks may be required to be executed and monitored in parallel and the location of intermediate data may be known only at run-time.

This chapter presents a workflow enactment engine developed as part of the Gridbus Project at the University of Melbourne, Australia [4]. It utilizes tuple spaces to provide an event-driven mechanism for workflow execution entities. The benefits of this design include the ease of deployment for various strategies of resource selection and allocation, and supporting complex control and data dependencies of tasks with scientific workflows.

2. Architecture

The primary components of the Workflow Enactment Engine (WFEE) [31] and their relationship with other services in the Grid infrastructure are shown in Figure 1. Workflow applications, such as scientific application portals, submit task definitions along with their dependencies, expressed in a workflow language, as well as associated QoS requirements to WFEE. WFEE schedules tasks through Grid middleware on the Grid resources.

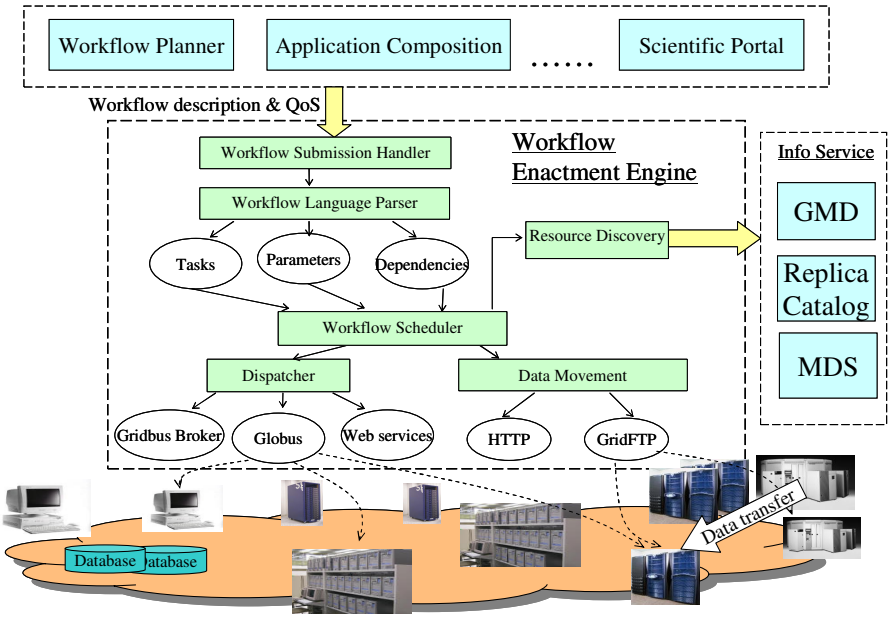


Figure 1: Architecture of WFEE.

The key components of WFEE are: workflow submission, workflow language parser, resource discovery, dispatcher, data movement and workflow scheduler.

- Workflow submission accepts workflow enactment requests from planner level applications.
- Workflow language parser converts workflow description from XML format into Java objects, *Task*, *Parameter* and *DataConstraint* (workflow dependency) which can be accessed by workflow scheduler.
- Resource discovery is carried out by querying Grid information services such as Globus MDS [12], directory service and replica catalogs, to locate suitable resources for the tasks.
- Dispatcher is used to access middleware. Resources may be Grid-enabled by different middleware such as Globus [12] or Web services [3]. WFEE had been designed to support different middleware by creating dispatchers for each middleware to support interaction with resources.
- Data movement system enables data transfer between Grid nodes by using HTTP and GridFTP [2] protocols.
- Workflow executor is the central component in WFEE. It interacts with *resource discovery* to find suitable Grid resources at run time; it locates a task on resources by using the dispatcher component; it controls input data transfer between task execution nodes through *data movement*.

3 Workflow Execution Management

The workflow execution is managed using a decentralized architecture. Instead of a central scheduler for handling whole workflow execution, a task manager is created for handling the processing of a task or a group of tasks, including resource discovery and allocation, task dispatcher and failure processing. Different scheduling strategies can be deployed in different Task Managers (TMs) for resource selection, QoS negotiation and data transmission optimization. The lifetimes of TMs, as well as the whole workflow execution, are controlled by a Workflow Coordinator (WCO).

As shown in Figure 2, dedicated TMs are created by WCO for each task group. Each TM has its own monitor which is responsible for monitoring the health of the task execution on the remote node. Every TM maintains a resource group which is a

set of resources that provides services required for the execution of an assigned task. TMs and WCO communicate through an Event Service Server (ESS).

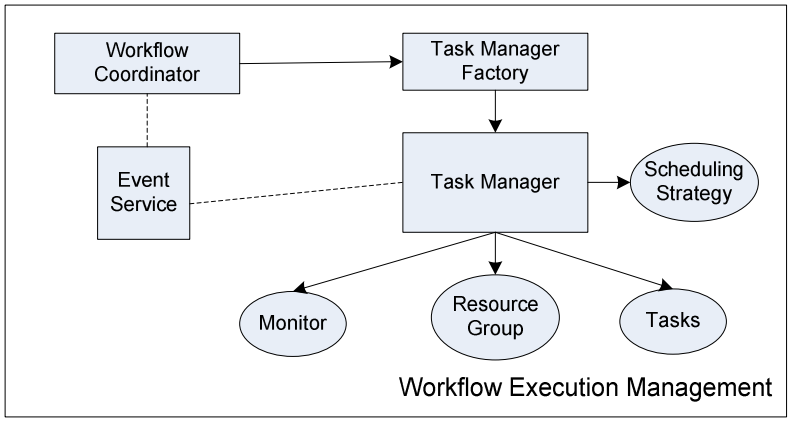


Figure 2: Execution management.

3.1 Communication Approach

A communication approach is needed for task managers. On one hand, every task manager is an independent thread of execution and they can be run in parallel. On the other hand, the behavior of each task manager may depend on the processing status of other task managers according to the task dependencies. For example, a task manager should not execute the task on a remote node if the input generated by its parent tasks is not available for any reason.

In addition, in a workflow, a task may have more than one input that comes from different tasks. Furthermore, the output of these tasks may also be required by other task managers as well. Hence the communication model between the task managers is not just one-to-one or one-to-many, but it could be many-to-many depending on task dependencies of the workflow.

Given this motivation, an event-driven mechanism with subscription-notification model has been developed to control and manage execution activities. In the system, the behaviors of task managers and workflow coordinator are driven by events. A task manager is not required to handle communication with others and only generates events according to a task's processing status. At the same time, the task managers take actions only depending on the events occurred without concern for details of other task managers. The benefit of this event-driven mechanism is that it provides

loosely-coupled control; hence the design and development of the system is very flexible and additional components can be easily plugged in.

.

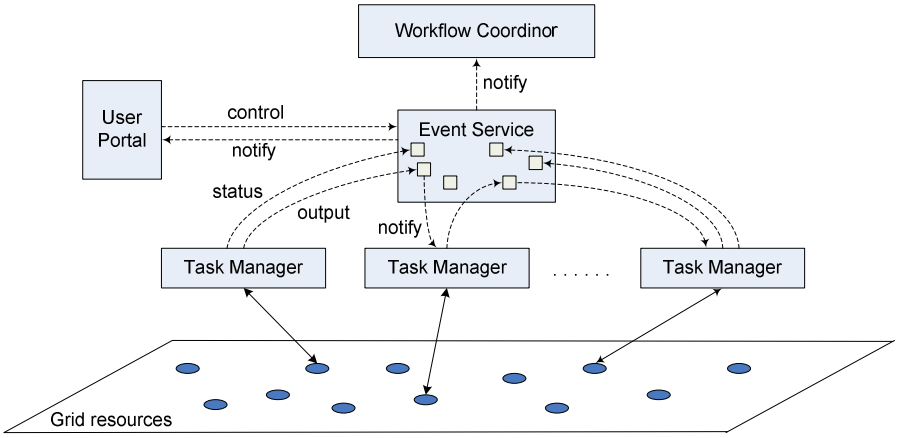


Figure 3: Event-driven mechanism.

The event notification is based on subscription-notification model. WCO and TMs just subscribe to events of interest after activation, and then are informed immediately when a subscribed event occurs. There are three basic types of events, *status events*, *output events* and *control events*. Status events are sent by the TMs to provide information on the status of task execution. Output events are sent by TMs to announce the task output is ready along with the location of its storage. Control events are used to send control messages, such as to *pause* and *resume* the execution, to task managers

As illustrated in Figure 3, TMs inform each other and communicate with the WCO through the ESS. For example, TMs put their task execution status (e.g. executing, done, failure) into the ESS, which notifies the WCO. If the output of a task is required by its child tasks, the task managers of the child tasks can subscribe to output events of the task. Once the task generates the required output, an output event is sent to the ESS, which notifies immediately, the child TMs that have subscripted to the output event. A user can control and monitor the workflow execution by subscribing to status events and sending control evens through a visual user interface.

3.2 State Transition

The state transition of WCO is illustrated in Figure 4. WCO registers with the ESS and start TMs of first level tasks, and then monitors activated TMs. Upon receiving execution status from a TM, WCO starts the TMs of its child tasks. If the WCO receives a status *done* event, it checks whether other TMs are still running. If so, WCO goes back to *monitoring*, otherwise it exits. If WCO receives a failed event from a TM, it proceeds to *failure processing*, and then ends.

The state transition of TMs is illustrated in Figure 5. The TM registers events, such as output events, status events, generated by its parent tasks and waits for the events to occur; when an event occurs, the TM goes to the *event processing* state. If all input data is available, it starts a new thread to process execution for a job; otherwise, the TM goes back to *wait* state. A job is a unit of work that a TM sends to a Grid node and one task may create more than one jobs. The job execution is started from *resource matching*, in which a suitable resource is selected from the resource group created by querying a directory service (see Section 4). If a suitable resource is available, the TM submits a job to the resource and then monitors the status of job execution on the remote resource. If the execution has failed, the TM goes back to *resource matching* and selects an alternative resource and then submits the job to it. If all parent tasks and execution jobs are completed, the TM ends.

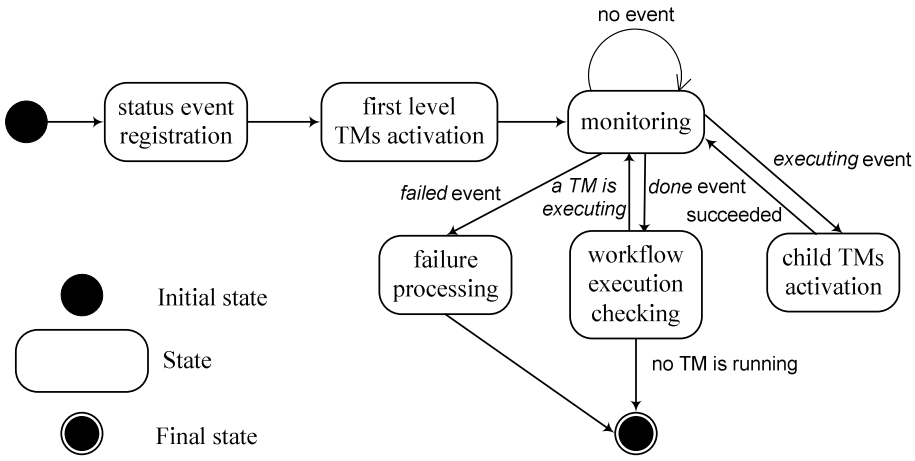


Figure 4: State transition of WCO.

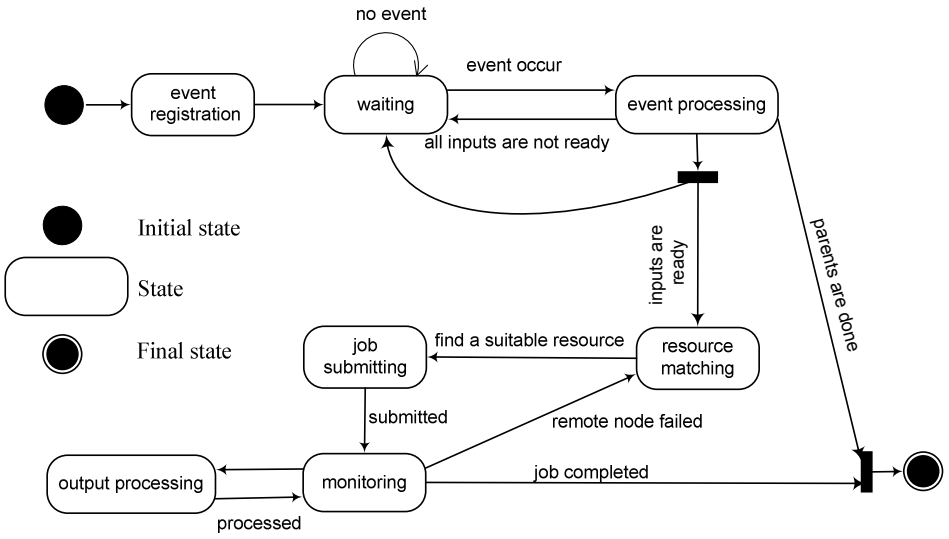


Figure 5: State transition of TM.

3.3 Interaction

The interactions between the WCO, TMs, ESS and remote resources are illustrated in Figure 6. First, the WCO needs to register to the ESS and subscribe to task status events. Then, the WCO activates task managers of first level tasks of which, in this example, there is only one TM1. After TM1 finishes the preprocessing for the task execution, it sends a message to ESS saying "*I am executing the task*". ESS informs the WCO and WCO activates TMs of the child tasks of TM1, namely TM2 and TM3, in this example.

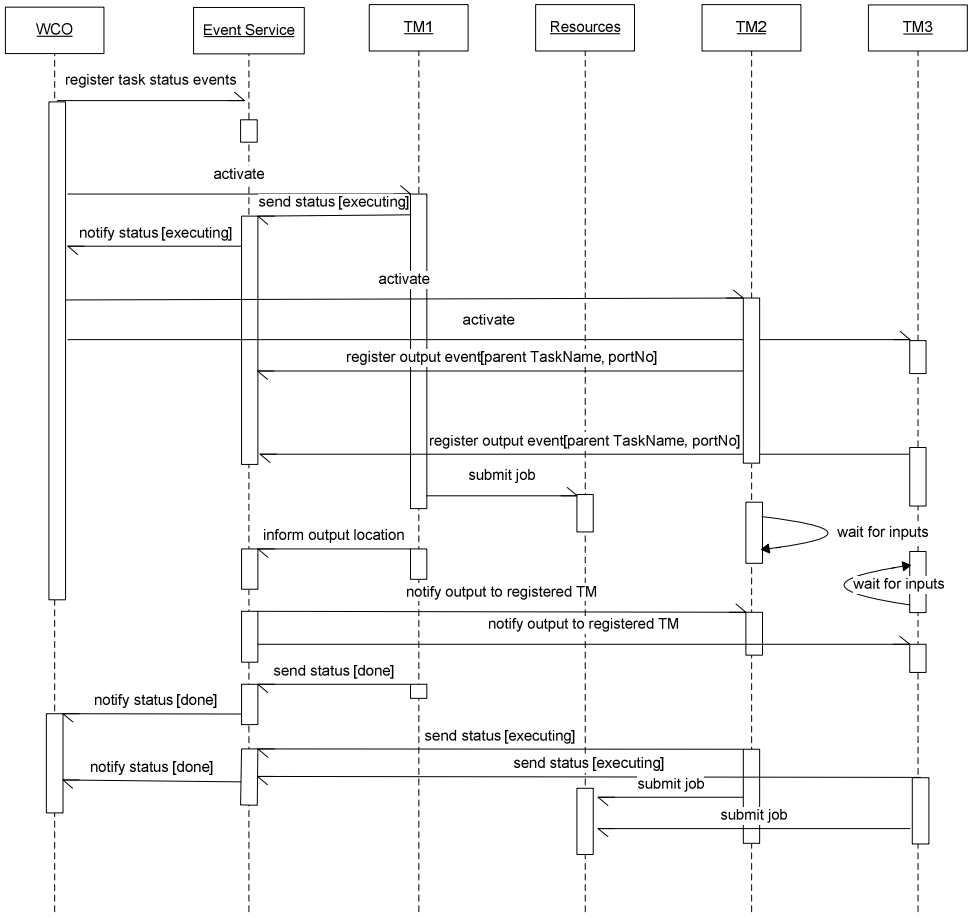


Figure 6: Interaction sequence diagram the WCO, TMs and ESS.

The inputs of the task managed by TM2 and TM3 rely on the output of the task of TM1, so TM2 and TM3 register to ESS and listen to its output events. Once TM1 identifies a suitable resource, it submits task to that resource. As soon as TM1 knows the output of the task, it informs TM2 and TM3 through ESS, saying *"my output of port No. x is ready and its location is xxxx"*. If all input data for TM2 and TM3 are ready, TM2 and TM3 reports execution status to ESS, and then proceeds to initialize the execution of their tasks. After WCO receives the notification of the execution of the tasks in TM2 and TM3, WCO will activate their child task managers, so that they can prepare for task execution. This process will be continued until the end of workflow execution.

4. Service Discovery

In a Grid environment, many services having same functionality and user interaction, can be provided by different organizations. In addition, a service may be replicated and deployed in many locations. From the user's point of view, it is better to use a service that offers a higher performance at a lower price. Therefore, a method is required to allow users to find replicated services easily.

A directory service, called *Grid Market Directory* (GMD) [33], has been developed to support service publication and discovery in a market-oriented Grid environment. GMD is an infrastructure that allows (a) the creation of one or more registries for service providers; (b) the service providers to register their resources/application services that they wish to provide; (c) users such as workflow engine to discover resources/services and their attributes (e.g., access price, location and usage constraints) that meet their QoS requirements.

Figure 7 illustrates service publishing and discovery in a Grid environment through GMD. Service providers' first register with the GMD and publish their static information such as location, service capability and access methods. A Grid user such as the workflow engine can query GMD to find a suitable service. After that, the user can also query and subscribe to the service provider directly to obtain more dynamic information such as service execution status.

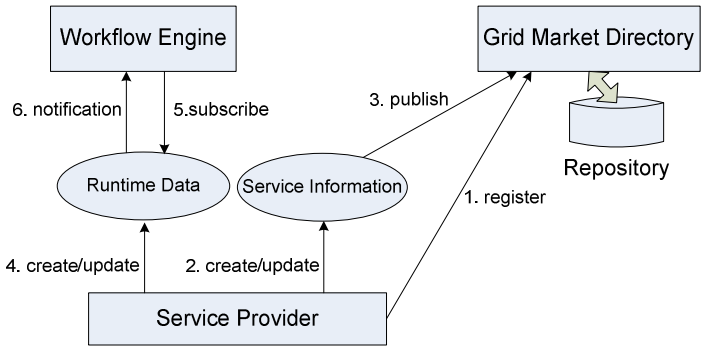


Figure 7: Service discovery using GMD.

A provider can provide their specialist applications for others to access remotely. Figure 8 shows an application service schema. An application service provider may also provide hosted machine information such as the host name and host public key for remote secure access. Service providers also need to indicate middleware through which Grid users can access the service.

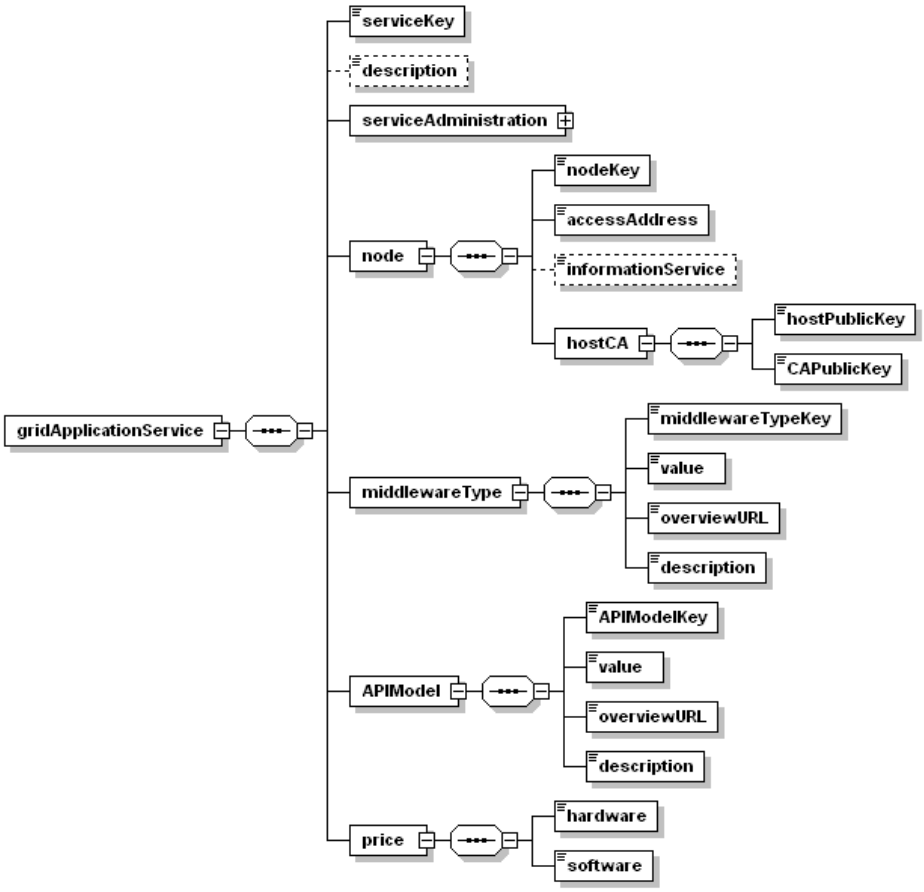


Figure 8: Grid application service schema.

Grid Application Model (GAM) is developed for application identification. GAM is a set of specifications and APIs for a Grid application. The GAM can be published by service providers within the GMD, and the users can search GMD for services conforming to a particular GAM. The applications with the same GAM name provide the same function and API. In the case of the workflow system, if users do not specify a particular service for a task in the workflow description, the scheduler uses the GAM name associated with the executable of the task to query the GMD. The GMD will return a list of services. These services are all able to execute the task.

5. Workflow Language

In order to allow users to describe tasks and their dependencies, a XML-based workflow language (xWFL) has been defined. The workflow language provides the means to build new applications by linking standalone applications.

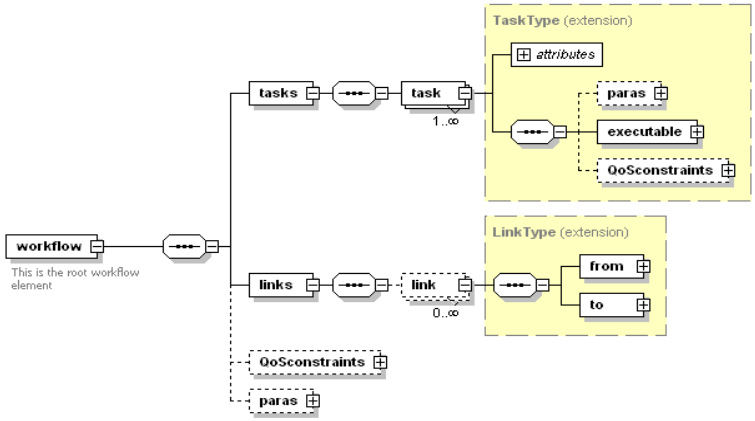


Figure 9: Structure of workflow language.

Figure 9 shows the basic structure of the workflow language. It consists of two parts: task definitions defined in <*tasks*>, data dependencies defined in <*links*> and QoS constraints defined in <*QoSconstraints*>.

5.1 Tasks

The element <*tasks*> is a set of definitions of tasks that are to be executed. Figure 10 shows the schema of task definition. A task can be a single task or a parameter sweep task. A parameter sweep task is able to process a set of parameters. The parameters are defined in <*paras*>. The detailed design of parameter tasks is introduced in Section 6. The element <*executable*> is used to define the information about the application, input and output data of the task. The workflow language supports both abstract and concrete workflows. The user or higher workflow planner can either specify the location of a particular service providing a required application in <*service*> or leave it to the engine to identify their providers dynamically at run-time. The middleware of the application is identified through a service information file by the GMD when dispatching tasks.

In the example that follows, task A executes dock.exe program on host *bellegrid.com* in the directory */services* and the executable *dock* has two input I/O ports: port 0 (a file) and port 1 (a parameter value). The example shows task *A* only has one output.

```
<task name= “A”>
  <executable>
    <name>dock</name>
    <service>
       <hostname=“bellegrid.com” />
        <accesspoint value=“/services/dock.exe” />
    </service>
     <input>
       <port num=0 type=“file” url=http://www.gridbus.org/dock.in
        value=“dock.in”/>
          <port num=1 type=“msg” value=1/>
    </input>
    <output>
          <port num=2 type=“file” value=“dock.out”/>
    </output>
  </executable>
</task>
```

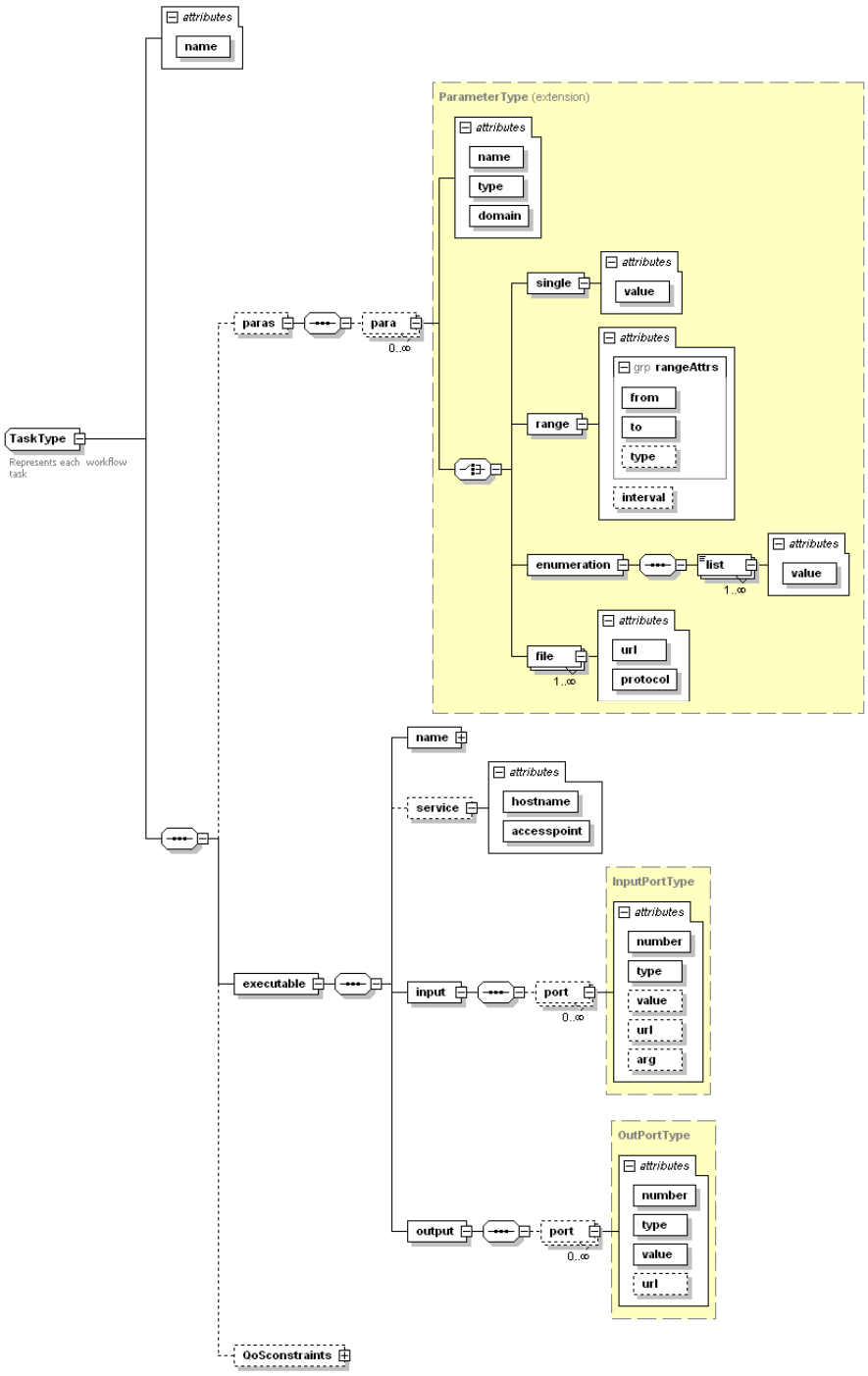


Figure 10: Schema of task definition.

5.2 Data Dependencies

A data link is used to specify the data flow between two tasks. The schema of the data link is defined in Figure 9. Figure 11 shows the example of a data flow description. The inputs of task *B* and task *C* rely on the output of *A*. The output of *A* needs to be

transferred to the node on which tasks *B* and *C* are executed. Input could be a file, parameter value or data stream.

```
<workflow>
  <tasks>
    <task name= "A">
          …..
    </task>
    <task name= "B">
       …....
    </task>
  <task name= "C">
   ……
  </task>
  <task name= "D">
   ……
  </task>
 </tasks>
 <links>
  <link>
   <from task="A" port=2 />
   <to task="B" port=0 />
  </link>
  <link>
   <from task="A" port=2 />
   <to task="C" port=0 />
  </link>
  <link>
   <from task="B" port=1 />
   <to task="D" port=0 />
  </link>
  <link>
   <from task="C" port=2 />
   <to task="D" port=1 />
  </link>
 </links>
</workflow>
```

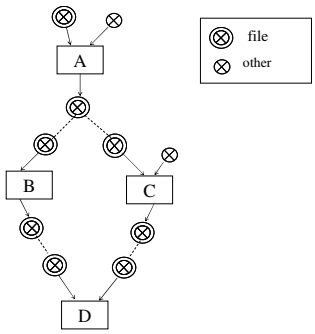


Figure 11: Flow diagram of task A, B, C and D.

6. Parameterization

Supporting parameterization in the workflow language is very important for scientific applications. It enables scientists to perform experiments across a range of different parameters without being concerned about the detailed workflow description. A parameter defined in each task type is called a *local parameter* ; when it is defined for the entire workflow, it is called a *global parameter*. As shown in Figure 9, multiple parameters types such as single, range, file and enumeration are supported. An example for a single parameter type and a range parameter type is given in Figure 12.

```
<paras>
     <para type= "single">
            <name>X</name>
            <value type=integer>10</value>
     </para>
     <para type= "range">
            <name>Y</name>
     <min>1</min>
```

```
            <max>20</max>
            <step>2</step>
        </para>
    </paras>
```

Figure.12: Single parameter and range parameter.

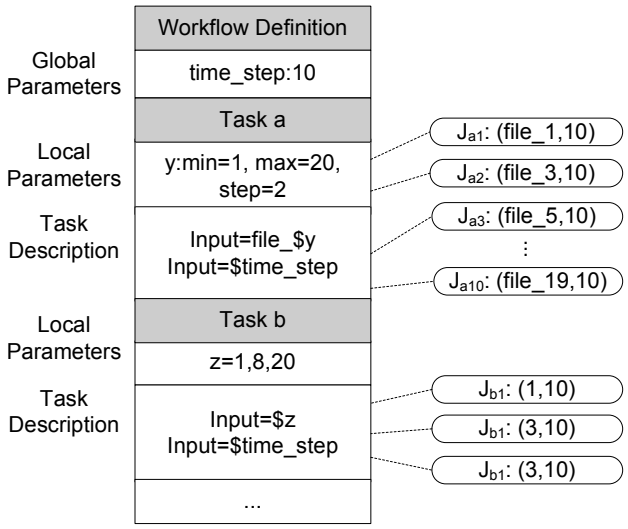


Figure 13: Illustration of workflow parameters.

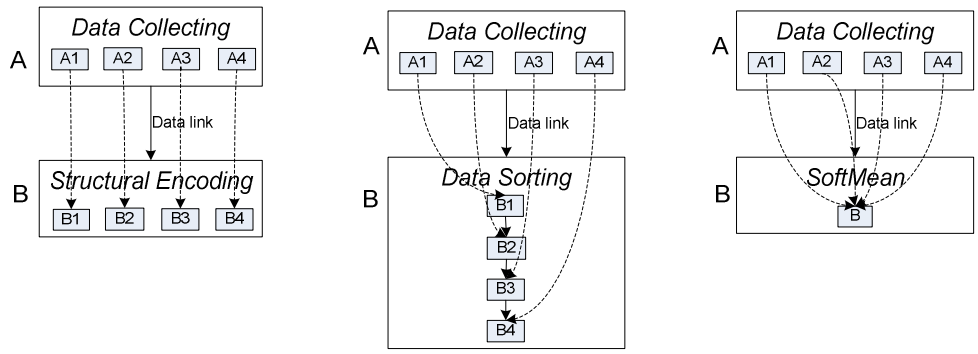
Among these parameter types, range and enumeration types are used to define range or a list of parameter valued which the task is required to be executed. This type of task is called as a *parameter-sweep* task [1] and is structured as a set of multiple execution jobs, each of which is executed with a distinct set of parameters. Figure 13 illustrates parameter sweep tasks. Two inputs are required by task *A* and Task *B*; one is a local parameter and the other is a global parameter. The local parameter of *A* is a range type parameter while the local parameter of *B* is an enumeration type. At execution time, the global parameter value is combined with each local parameter value and generates 10 sub-jobs for task *A* and 3 sub-jobs for task *B*.

7. I/O Models

As shown in Figure 11, a task receives output data from its parent as its input through a data link. However, for some tasks, more than one output could be generated by one output port. For example, such a task could be a data collecting task that continues to read information from a sensor device and generate corresponding output data or a parameter-sweep task that generates multiple data based on various sets of parameters. These outputs can be produced at different times. Some successor tasks may not require to be processed until all these output data are generated. It can process the

output once it is available. However, ancestor tasks can process the output data generated from a single parent differently, depending on their requirements.

Three I/O models have been developed in the workflow system to provide data-handling capabilities. These models are: *many-to-many*, *many-to-one* and *synchronization*. In Figure 13, there are two tasks: task *A* and task *B*. They are connected by a data link. There are multiple sub-jobs in *A* and each job produces an output. For the many-to-many model, task *B* starts to process data and generates an output once there is an input available on the data link. As shown in Figure 14a, four outputs generated by four sub-jobs of *A* are processed individually by four sub-jobs of *B*. For the many-to-one model, task *B* starts to process data once there is an input available, however, the result is calculated based on the result generated by earlier sub-jobs. As shown in Figure 14b, sub-job *B1* processes the output generated by sub-job *A1*. Once the output of *A2* is available, sub-job *B2* is created and processes the output of *A2* based on the output generated by *B1*. For the synchronization model, task *B* does not start processing until all the output is available on the data link. As shown in Figure 14c, there is only one sub-job in task *B* and it processes all outputs generated by sub-jobs of *A* at one time.



a) Many-to-Many b) Many-to-One c) Synchronization

Figure 14: Input/output models.

8. Fault Handling

Two fault handling mechanism are developed in the system, *retry submission* and *critical task replication.*

The retry submission mechanism reschedules a failed job onto a current available resource, and also records the number of failed jobs for each resource. Once the failed

job number exceeds a *warning threshold*, the scheduler decreases the number of jobs submitted to these resources. If the number of failed jobs exceeds a *critical threshold*, the scheduler terminates submission of jobs onto this resource.

The critical task replication mechanism replicates a task execution on more than one resource. The result produced earliest is then used for the rest of the workflow. This mechanism is designed to execute a long running task when there are multiple spare resources.

9. Implementation

The WFEE has been implemented by leveraging the following key technologies: (1) IBM TSpaces [30] for supporting subscription-notification based event exchange; (2) Gridbus broker [28] for deploying and managing job execution on various middleware; (3) XML parsing tools including JDOM [14]. The detailed class diagram and event server implementation are presented as follows:

9.1 Design Diagram

The class design diagram of the WFEE is shown in Figure 15. *XMLParsingToModel* parses XML formatted workflow description into Java objects which are instances of class of *Task*, *Port*, *DataConstraint*. These objects are passed on to *WorkflowModel*. *WorkflowModelToDiGraph* converts *WorkflowModel* into a directed graph represented by class *DiGraph* which encompasses many *GraphNode* objects. An instance of *GraphNode* contains a workflow task and the references of *GraphNodes* of its parent and child tasks. *WorkflowCoordinator* creates and controls the instantiation of *TaskManager* according to the graph node dependencies. *Job* class represents a unit of work assigned to a Grid resource. Every job has a monitor implemented by *JobMonitor* to monitor job execution status on the remote node. In order to extend WFEE to support multiple Grid middleware, we abstract class *Resource* and *Dispatcher* which provides interfaces that interact with Grid resources.

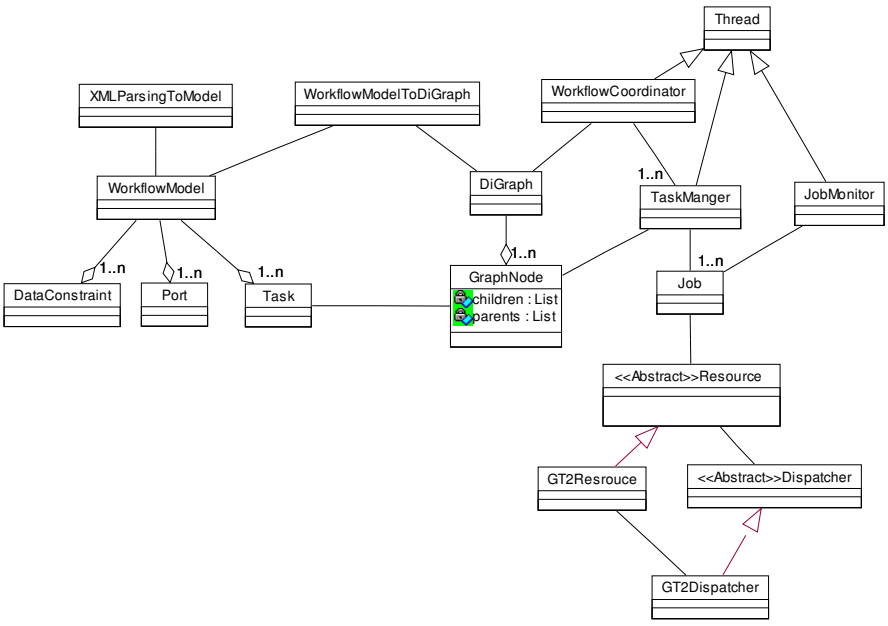


Figure 15: Class diagram of WFEE.

9.2 Event Messages

We have utilized tuple spaces to exchange events. The tuple space model is originated at Yale with the Linda system [6]. A tuple is simply a vector of typed values (Fields). A tuple space is a collection of tuples that can be shared by multiple parties by using operations such as read, write and delete. In our work, we have leveraged IBM's recent implementation of tuple spaces called TSpaces [30] to be the event server.

Table 1: Format of events.

Event name	Field1	Field2	Field3	Tuple template for registration
task status event	task No.	"status"	value	new Tuple(new Field(String), status, new Field(String))
output event	task No.	port No.	value	new Tuple(taskNo, portNo, new Field(String))
job status event	job No.	task No.	value	new Tuple(new Field(String), taskNo, new Field(String))

There are three types of event tuples whose format is shown in Table 1, *task status event*, *output event* and *job status event*. Task status event is sent by TMs and WCO use it to control TMs activation. The first field is the ID of the task, the second field is string "status" to indicate the type of tuple for the registration purposes, and the third field gives the value of status.

Child TMs need output events sent by the parent TMs to be informed if their inputs are available. The events have three fields: the task ID is given in the first field, the second field is port numbers, and the third field is the location of output.

One task can have multiple jobs. Job status events are sent by the job monitor. Every job status event provides a job ID and its task ID with status value. TMs make decisions according to the job events. For example, when a job has failed, the TM can reschedule it on another resource in the resource group.

The tuple templates are used for subscribing to the corresponding event. For example, task status events can be received by a tuple template with the second field as a String called “status”.

10 A Case Study in fMRI Data Analysis

Magnetic Resonance Imaging (MRI) [18] uses radio waves and a strong magnetic field to provide detailed images of internal organs and tissues. Functional MRI (fMRI) [15] is a procedure that uses MRI to measure the signal changes in an active part of the brain. fMRI is becoming a major diagnostic method for learning how a normal, or a diseased brain is working. fMRI images obtained by scanning the brains of subjects as they perform cognitive tasks. A typical study of fMRI data consist of multiple-stage processes that begin with pre-processing of raw data and conclude with a statistical analysis. Such analysis procedures often require upon hundreds or even thousands of images [35].

10.1 Population-based Atlas Workflow

Population-based atlas [27] creation is one of the major fMRI research activities. These atlases combine anatomy imaging data from healthy and diseased populations. These describe how the brain varies with age, gender, and demographics. They can be used for identifying systematic effects on brain structure. For instance, they provide a comprehensive approach for studying a particular subgroup, with a specific disease, receiving different medications, or neuropsychiatric disorder. Population-based atlases contain anatomical models from many subjects. They store population templates and statistical maps to summarize features of the population. They also

average individual images together so that common features of the subgroup are reinforced.

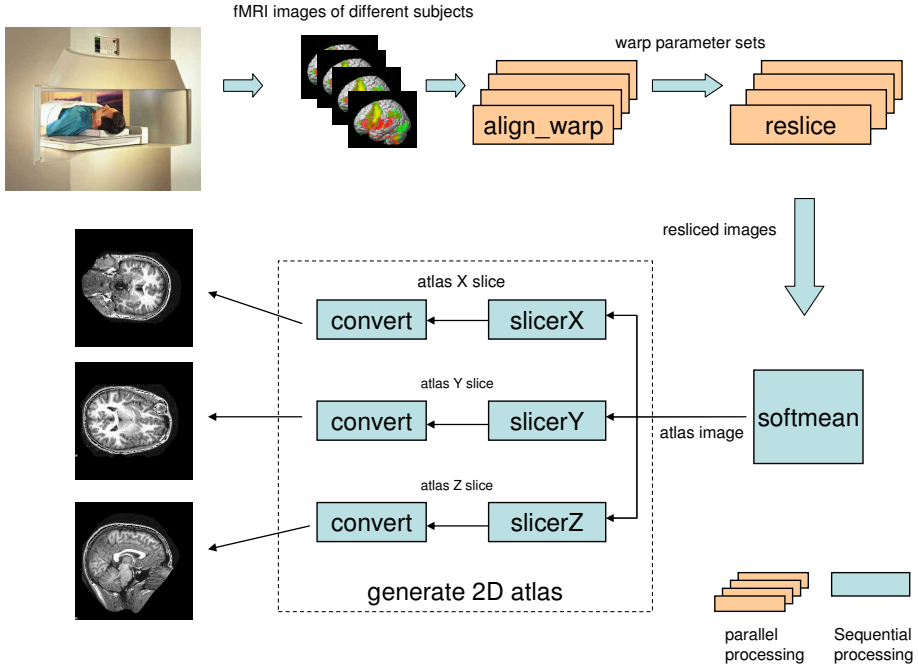


Figure 16: Population-based atlas workflow.

Figure 16 shows a workflow that employs the Automated Image Registration (AIR) [29] and FSL [23] suite for creating population-based brain atlases from high resolution anatomical data. The stages of this workflow are follows:

a) The inputs to the workflow are a set of brain images which are 3D brain scans of population with varying resolutions and a reference brain image. For each brain image, *align_warp* adjusts the position and shape of each image to match the reference brain. The output of each process is a *warp parameter set* defining the spatial transformation to be performed.

b) For each warp parameter set, *reslice* creates a new version of the original brain image according to the configuration parameters defined in the warp parameter set. The output of each reslice procedure is a resliced image.

c) *softmean* averages all the resliced images into one single atlas image.

d) The averaged image is sliced using *slicer* to give a 2D atlas along a plane in three dimension (x, y and z), taken through the centre of the 3D image. The output is an atlas data set.

e) Finally, each atlas data set is converted into a graphical atlas image using *convert*.

10.2 Experiment

Table 2: Applications configuration of Grid sites.

Node / Details	Applications	Location
Manjra.cs.mu.oz.au	AIR FSL Convert	University of Melbourne, Australia
vgtest.vpac.org	AIR	VPAC, Australia
Vgdev.vpac.org	AIR	VPAC, Australia
Brecca-1.vpac.org	AIR	VPAC, Australia
Brecca-2.vpac.org	AIR	VPAC, Australia
karwendel.dps.uibk.ac.at	FSL	University of Innsbruck, Austria
uuuu.maekawa.is.uec.ac.jp	FSL	University of Electro-Communications, Japan
walkure.maekawa.is.uec.ac.jp	FSL	University of Electro-Communications, Japan

* AIR package includes software for executing the task

Table 3: Resource attributes.

Node / Details	CPU (type/#/GHz)	Middleware
manjra.cs.mu.oz.au	4/Intel Xeon/2.00GHz	SSH/GT2
vgtest.vpac.org	1/Intel Xeon/3.20GHz	SSH/GT4
ngdev.vpac.org	1/Intel Xeon/3.20GHz	SSH/GT4
brecca-1.vpac.org	Intel Xeon/4/2.80GHz	SSH
brecca-2.vpac.org	4/Intel Xeon/2.80GHz	SSH
karwendel.dps.uibk.ac.at	2/AMD Opteron 880/2.39 GHz	SSH/SGE
uuuu.maekawa.is.uec.ac.jp	1/Intel Xeon/2.80 GHz	SSH/GT4
Walkure.maekawa.is.uec.ac.jp	1/Intel Xeon/2.80 GHz	SSH/GT4

The experiment was conducted using the testbed provided by the University of Melbourne (Australia), Victorian Partnership for Advanced Computing (VPAC) (Australia), University of Electro-Communications (Japan), and University of Innsbruck (Austria). The configuration of all resources is listed in Table 2 and Table 3. Table 2 shows the application software available on every resource. All application software cannot be installed on every resource, due to their varied capability and administration policy. The AIR application required for executing procedure *align_warp*, *reslice* and *softmean* is installed on the sites of VPAC and the University of Melbourne, while the FSL application required for executing procedure *slicer* is installed on other sites. The Convert application required to execute procedure *convert* is only available on the site of the University of Melbourne. Table 3 shows processor cabability and supporting middleware of each resource.

In the first experiment, the impact of the number of Grid sites is investigated for the various numbers of subjects. Figure 17 shows the total execution times using 1-5 Grid sites for generating atlas of 25, 50 and 100 subjects. The size of image file associated with each subject is around 16 to 22 MB. We can see that the total execution time increases as the number of subjects increases. Additionally, the larger the number of Grid sites, the faster execution time is achieved. For example, the total execution time of generating an atlas of 100 subjects using one Grid node is 95 minutes; however, it only takes 45 minutes using five Grid nodes. The speedup rate is over 50%. It shows the performance of conducting fMRI data analysis can be significantly improved by using the Grid.

Figure 18 shows the execution progress for processing 50 subjects. At the beginning of the workflow execution, 50 *align_warp* jobs are generated for the first step, and each job processes one subject image. Once a job in the step one is completed, the task manager of the step two is notified by the output event of this job. It then generates a new *reslice* job of the step two. Therefore, the number of waiting jobs does not continuesly decrease when *align_warp* jobs are completed. In Figure 18, we can obverse that the number of waiting jobs is remained around 50 until the 50 jobs of the step one are completed. All the results of the step two are processed once by the *softmean* task, and the completion of *softmean* generates three *slicer* jobs to produce 2D images along three dimensions. Therefore, there is only small number of waiting jobs after 600 seconds.

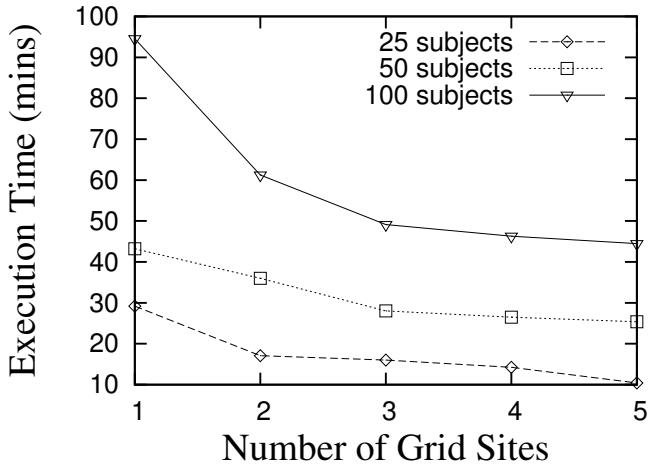


Figure 17: Total execution times of processing 25, 50 and 100 subjects over various Grid sites.

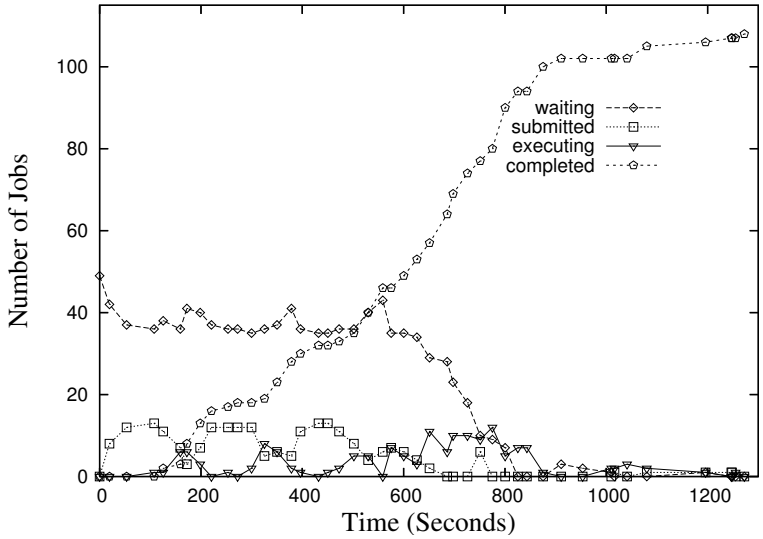


Figure 18: Execution progress for processing 50 subjects.

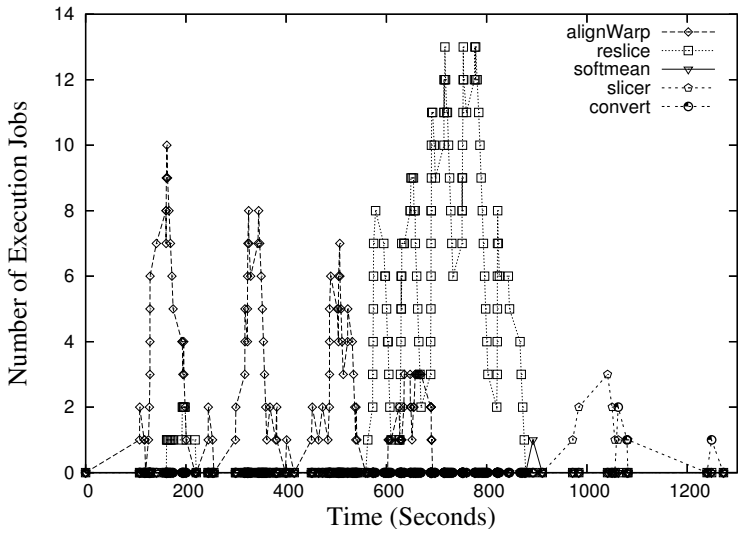


Figure 19: Execution tasks for processing 50 subjects.

Figure 19 shows the number of jobs of each task running over the execution. Available jobs of the step one and step two can be executed in parallel. However, the jobs of the step one have a higher priority than those of the step two, when they are compete for resources. As we can obverse the most execution jobs during 0-580 seconds are produced by *alignWarp* and only a few of the jobs of *reslice* were executed.

Table 4: Detailed execution times of the tasks for processing 100 subjects.

Task	**Start Time (min)**	**End Time (min)**	**Duration (min)**
align_warp	0	22.82	22.82
reslice	2.65	28.50	25.85
softmean	28.92	43.08	14.17
sliceX	43.1	44.1	1
sliceY	43.1	44.13	1.03
sliceZ	43.1	44.07	0.97
convertX	44.4	44.72	0.32
convertY	44.42	44.75	0.33
convertZ	44.07	44.43	0.37

Table 4 shows the start and end time for each task in the workflow for 100 subjects. The start time we measured is the time when the stage-in of input data to the remote resource is started and the end time is the completion time of a task. As we can see from Figure 16, there are multiple sub-tasks in task *align_wrap* and *reslice*. The

task *reslice* process was started after *align_wrap* process, once an output produced by a sub-task of *align_wrap* was available. However, the task *softmean* is started to process after all sub-tasks of align_wrap have been comeleted, because it requires all results produced by the sub-task of *align_wrap* to generate a mean image. Theoretically, after task *softmean* finishes, its child tasks should be submitted immediately, however, there are some time intervals between the parent tasks' end time and child tasks' start time. That gap can be attributed to the overhead of running WFEE, including time involved in processing event notifications, resource discovery and remote resource submission. However, compared to the running time of tasks, this gap is insignificant and less than 2%.

11. Related Work

The workflow engine presented in this chapter is an independent workflow execution system and takes advantage of various middleware services such as security, Grid resource access, file movement and replica management services provided by the Globus middleware [10][12], and multiple middleware dispatchers provided by the Gridbus Broker.

Many efforts toward grid workflow management have been made. DAGMan [26] was developed to schedule jobs to the Condor system in an order represented by a DAG and to process them. With the integration of Chimera [11], Pegasus [7] map and execute complex workflows based on full-ahead-planning. In Pegasus, a workflow can be generated from metadata description of the desired data product using AI-based planning technologies. The Taverna project [20] has developed a tool for the composition and enactment of bioinformatics workflow for the life science community. The tool provides a graphical user interface for the composition of workflows. Other workflow projects in the Grid context include UNICORE [21], ICENI [19], Karajan [17], Triana [24] and ASKLON [9].

Compared with the work listed above, the workflow engine provides a decentralized scheduling system by using tuple spaces model, which facilitates deployment of different scheduling strategies to each task. It also enables resources to be discovered and negotiated at run-time.

A number of workflow languages [3][8][16] have been developed and most of them focus on the composition of web services. However web services are not the

standard middleware used by the majority of today's scientific domains [34]. The workflow language proposed in this chapter is middleware independent and also supports parameterization [1], which is important to scientific applications.

12. Summary

In this chapter, a workflow enactment engine is introduced to facilitate composition and execution of workflows in a user-friendly manner. The engine supports different Grid middleware as well as run-time service discovery. It is capable of linking geographically distributed standalone applications and takes advantage of distributed computational resources to achieve high throughput.

The event-driven and subscription-notification mechanisms developed using the tuple spaces model make the workflow execution loosely-coupled and flexible. Supporting parameterization in the workflow language allows users to easily define a range and list of parameters for scientific experiments to generate a set of multiple parallel execution jobs. The engine has been successfully applied to an fMRI analysis application. The engine presented in this chapter facilitates users to build workflows to solve their domain problems and provides a basic infrastructure to schedule workflows in Grid environments.

References

[1] D. Abramson, J. Giddy, and L. Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?. *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, Cancun, Mexico, IEEE CS Press, Los Alamitos, CA, USA, May 1-5, 2000.

[2] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke. Data Management and Transfer in High Performance Computational Grid Environments. *Parallel Computing Journal*, 28(5):749-771, May 2002.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana. Business Process Execution Language for Web Services Version 1.1, 05 May 2003, http://www-128.ibm.com/developerworks/library/ws-bpel/ [Feb 2005]

[4] R. Buyya and S. Venugopal. The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report. *Proceedings of the First IEEE International Workshop on Grid Economics and Business Models* (GECON 2004, ISBN 0-7803-8525-X, IEEE Press, New Jersey, USA), April 23, 2004, Seoul, Korea.

[5] J. Cardoso. Stochastic Workflow Reduction Algorithm. Technical Report, LSDIS Lab, Department of Computer Science University of Georgia, 2002.

[6] N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32:444-458, April 1989.

[7] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. H. Su, K. Vahi, M. Livny. Pegasus: Mapping Scientific Workflow onto the Grid. *Proceedings of the Across Grids Conference 2004,* Nicosia, Cyprus, 2004.

[8] T. Fahringer, S. Pllana, and A. Villazon. AGWL: Abstract Grid Workflow Language, *Proceedings of the International Conference on Computational Science, Programming Paradigms for Grids and Meta-computing Systems*. Krakow, Poland, Springer-Verlag, Heidelberg, Germany, June 2004.

[9] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. S. Jr, and H. L. Truong. ASKALON: a tool set for cluster and Grid computing. *Concurrency and Computation: Practice and Experience*, 17:143-169, Wiley InterScience, 2005.

[10] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, 11:115-128, 1997.

[11] I. Foster, J. Vöckler, M. Wilde, Y. Zhao. Chimera: A Virtural Data System for Representing, Querying, and Automating Data Derivation. *Proceedings of the 14th International Conference on Scientific and Statistical Database Management (SSDBM)*, Edinburgh, Scotland, UK, July 24-26, 2002.

[12] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. *IFIP Internatiaonl Conference on Network and Parallel Computing*, 2006.

[13] D. Hollinsworth. The Workflow Reference Model, Workflow Management Coalition, TC00-1003, 1994.

[14] JDOM. http://www.jdom.org [December 2004]

[15] J. Van Horn. Online Availability of fMRI Results Images, *Journal of Cognitive Neuroscience*, 15(6):769-770, 2003.

[16] S. Krishnan, P. Wagstrom, and G. v. Laszewski. GSFL: A Workflow Framework for Grid Services, Argonne National Laboratory, Technical Report Preprint ANL/MCS-P980-0802, Aug 2002.

[17] G. von Laszewski, M. Hategan. Java CoG Kit Karajan/GridAnt Workflow Guide. Technical Report, Argonne National Laboratory, Argonne, IL, USA, 2005.

[18] J. Mattson and M. Simon. The Pioneers of NMR and Magnetic Resonance in Medicine: The Story of MRI. Jericho & New York: Bar-Ilan University Press, 1996.

[19] S. McGough, L. Young, A. Afzal, S. Newhouse, and J. Darlington. Workflow Enactment in ICENI. *Proceedings of the UK e-Science All Hands Meeting*, Nottingham, UK, IOP Publishing Ltd, Bristol, UK, Sep. 2004; 894-900.

[20] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver and K. Glover, M.R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045-3054, Oxford University Press, London, UK, 2004.

[21] M. Romberg, The UNICORE Architecture Seamless Access to Distributed Resources, *Proceedings of the 8th IEEE International Symposium on High Performance Computing*, Redondo Beach, CA, USA, 1999, pp. 287-293.

[22] D P. Spooner, J. Cao, S. A. Jarvis, L. He, and G. R. Nudd. Performance-aware Workflow Management for Grid Computing. *The Computer Journal*, Oxford University Press, London, UK, 2004.

[23] S. Smith, P. Bannister, C. Beckmann, M. Brady, S. Clare, D. Flitney, P. Hansen, M. Jenkinson, D. Leibovici, B. Ripley, M Woolrich, and Y. Zhang. FSL: New tools for

functional and structural brain image analysis. *Proceedings of the 7th International Conference on Functional Mapping of the Human Brain*, June 10-14, 2001, Brighton, UK.

[24] I. Taylor, M. Shields, and I. Wang. Resource Management of Triana P2P Services. *Grid Resource Management*, Kluwer, Netherlands, June 2003.

[25] I. Tayler, E. Deelman, D. Gannon, M. Shields (Editors). *Workflows for E-science: Scientific Workflows for Grids*, Springer-Verlag London Ltd, London, UK, Dec 2006.

[26] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons, NJ, USA, 2003.

[27] P. Thompson, M. S. Mega, and A. W. Toga, Sub-Population Brain Atlases, *Brain Mapping: The Methods (2nd Edition)*, A. W. Toga and J. C. Mazziotta, Eds., 2002.

[28] S. Venugopal, R. Buyya and L. Winton. A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids. *Concurrency and Computation: Practice and Experience*, 18(6): 685-699, Wiley Press, New York, USA, May 2006.

[29] R. P. Woods, S. R. Cherry, J. C. Mazziotta. Rapid automated algorithm for aligning and reslicing PET images. *Journal of Computer Assisted Tomography*, 16:620-633, 1992.

[30] P. Wyckoff. TSpaces, *IBM Systems Journal*, 37, 1998.

[31] J. Yu and R. Buyya. A Novel Architecture for Realizing Grid Workflow using Tuple Spaces. *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)* , Nov. 8, 2004, Pittsburgh, USA.

[32] J. Yu, S. Venugopal, and R. Buyya. A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and their Services. *The Journal of Supercomputing*, 36(1): 17-31, ISSN: 0920-8542, Springer Science+Business Media, Berlin, Germany, April 2006.

[33] J. Yu and R. Buyya. A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing*, 3(3-4): 71-200, Springer Science+Business Media B.V., New York, USA, Sept. 2005.

[34] Y. Zhao, M. Wilde, I. Foster, J. Voeckler, T. Jordan, E. Quigg, and J. Dobson. Grid Middleware Services for Virtual Data Discovery, Composition, and Integration, *Proceedings of the 2nd Workshop on Middleware for Grid Computing*, Toronto, Ontario, Canada, 2004.

[35] Y. Zhao, J. Dobson, I. Foster, L. Moreau, and M. Wilde, A Notation and System for Expressing and Executing Cleanly Typed Workflows on Messy Scientific Data, *ACM SIGMOD Record*, 34, September 2005.