Temporal Dimension for Job Submission Description Language

Omar Aldabbas, Mai Alfawair, Hussein Zedan, Antonio Cau
Software Technology Research Laboratory (STRL)
Faculty of Computing Sciences and Engineering
De Montfort University
The Gateway,Leicester LE1 9BH
UNITED KINGODOM
{aldabbas, mfawair, zedan, cau}@dmu.ac.uk

Abstract: A grid job's requirements description is very important, and raises interesting issues for resource management and scheduling within a grid environment. Several languages such as Globus Resource Specification language and European Data Grid JDL have been developed for describing grid jobs, the latest of which is Job Submission Description Language (JSDL). None of the current job description languages deal with applications or events, and all have limitation with regard to issues of time.
In this paper we propose an extension of JSDL to cope with jobs' granularity, their composition, event and time. This extension will enable JSDL to support the description of application and their jobs flow, and allow the system to interrupt job according to event or time. It will also give the user the flexibility to determine the job execution times that assist to complete his/her job; expectation for job execution time is very difficult especially in heterogeneous environment. The new extension enables user to define and describe; more than one job and their relationship at the same time, event occurrence at any time during the job execution time and handle it, and job time. Our extensions have been evaluated on a simulated grid of 100 nodes which has shown better performance and excellent scheduling results of the grid. The introduction of event has also increased the reliability aspect of the grid.

Key–Words: Grid Computing, JSDL, Grid Application, Event.

1 Introduction

The growing popularity of the Internet and the availability of powerful computers and high speed networks have led to the development of next generation computing, called grid computing [1]. Grid computing is a relatively new approach by which future ubiquitous high speed networks can be provided with seamless wide area access to high-end computational capabilities. A computational grid is a distributed computing infrastructure for advanced science and engineering applications[2]. The grid paradigm enables the coordination and sharing of a large number of geographically dispersed heterogeneous resources such as supercomputers, mainframes, storage systems and data sources, all of which behave as a network of computation[3].
Researchers in many disciplines rely on grids to collaboratively solve processing and data-intensive problems which are too large for the resources of a single institution. Grid infrastructures provide for more efficient utilisation of existing resources by combining heterogeneous, widely distributed computational resources into a cohesive computing infrastructure [4].
An application is a collection of jobs working together with relationships between these jobs. The execution of these should fulfil the general requirements of the application. The relationships between jobs may be classified into the three categories of serial flow, parallel flow and network flow. A job is single unit of work that is executed at an appropriate node on the grid, and is mainly described by a set of attributes that identify the task to be done and resource requirements.
The requirements description of an application and relationships between them is very important issue for resource management and scheduling for all distributed resources such as the grid environment, clusters and SMPs. It will help the resource broker to match available resources to the user's needs.
An event is a message indicating the occurrence of a situation. The distributed resources environment especially grid computing has various distributed entities including broker, and nodes that are employed to fulfil the general requirements of the application. The more distributed these resources, the more event occurrence such as process failure, system performance degradation and so on. This means that the event ratio will increase throughout the system, for many rea-

sons [5, 6]. Therefore, all descriptive languages have to support for the event handler linguistically by enabling the user to direct the resource broker to handle events. This direction either gives the resource broker the permission to handle events dynamically without user intervention or allow the user to consider the events and describe the handling of them with the job submission. Several languages for describing grid jobs, such as Globus Resource Specification language [7], European Data Grid JDL [8], have been developed. The latest one is called Job Submission Description Language (JSDL) [9], which is used to present the standardised job submission language. This language has been designed by the Working Group Global Grid Forum (GGF). JSDL is being used in many existent projects such as Unicore systems [10], the HPC-Europa project [11]and Job Scheduling Hierarchically (JOSH) [12]. JSDL is a good solution for describing individual jobs, but it has some deficiencies and limitations. The deficiencies are in application description and event handling, because the JSDL only allow the user to describe the individual jobs and it does not cover event handling. Regarding the limitation of time handling, JSDL only allows the user to describe fixed range execution time. The present paper addresses these issues. In particular, we introduce a concurrency construct which will aid in the description of large scale grid applications. In addition, we describe a powerful construct that relates both handling event and timing-constraints. This is useful for such as handling interrupt and timing-constraints. This paper is organised as follows: In section 2 we describe the architecture model, in section 3 we explain the linguistic support for job granularity, timing issues and classes of events that could happen in the grid's environment, in section 4 we describe the implementation of the extension of JSDL for job granularity, event and time, section 5 describe the experiment and result and finally we present the conclusion.

2 Architecture Model

Grid relies on advanced software that ensures seamless communication between nodes. It deploys a powerful mechanism which determines the locations of requisite data and the nodes necessary to carry them out, and it efficiently determines and exploits unused resources distributed over the grid and assigns them to computers who are currently in need of it. In Figure 1, the grid architecture consists of a user, broker and nodes.

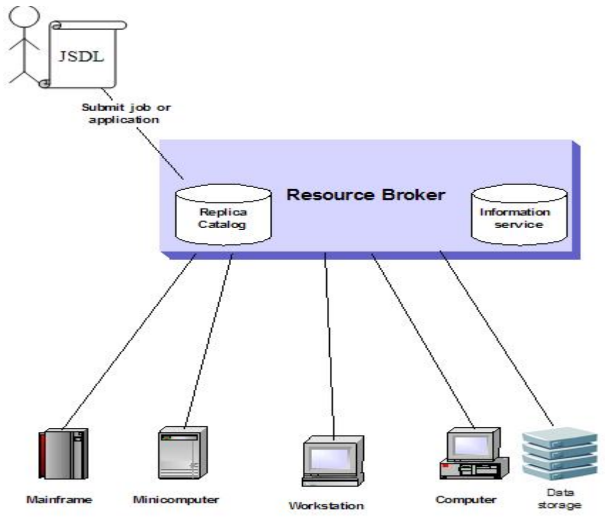

Figure 1: Grid Architecture

2.1 User

Users access the grid through a software interface running on their own computer. They are connected to the grid and are able to communicate with the resource broker. The user describes the application and job requirements by using JSDL which is in XML format, and send it to the resource broker.

2.1.1 Application Requirements

Grid applications are jobs that need to be executed on grid resources. These applications have requirements such as the names of all jobs, the relations between jobs, and individual job requirements. These application requirements must be considered by the resource broker which must therefore select optimal resources capable of meeting those requirements. This stage is pivotal, as it will influence the stages that determine the nodes where the application will execute and how events are handled.

2.1.2 Job Requirements

Users need to describe their job's requirements. These include job identification, software applications, data staging, event handling, execution start time and duration, deadline, resource specifications (CPU architecture, speed, file system, operating system, physical and virtual memory) bandwidth and security. This stage is equally vital, for the same reasons as the application requirements.

2.2 Resource Broker

A resource broker is a central component of a grid environment. The aim of the resource broker is to find, characterise, select and allocate the resources most suited to given jobs, and to handle events. After the resource broker receives a job or an application from the user it will communicate with the information service to query information regarding software, hardware and currently available resources, and with the replica catalogue to determine the location of all existing data. Once this information has been located, the resource broker will transfer the job to the resource to execute. If any event occurs while the job is being executed using the resource, the resource broker will handle this event and send the results back to the user. All these transitions are done transparently to the user, who views the grid as a single huge and powerful computer.

2.3 Node

The node is a shared entity providing some capabilities. These entities are the most basic component in the grid. Communication between nodes is achieved via network capabilities. We assume the resource broker does not mange this network. Each node has its own applications software, policy and data. A node is therefore able to execute a job. Nodes can be categorised according to their functionality in the grid. The most two common types of nodes are computational and storage nodes.
Computational nodes are the machines whose hardware and processing capabilities will be deployed and exploited. The node could be a cluster, mainframe, high performance computer or desktop. These capabilities will mainly be provided by the various kinds of processor architecture, in which each computation node will have its own processor architecture and its own hardware specifications such as speed, software platform compatibility and internal memory.

2.4 Data

Data is a piece of information stored in a grid node, and is used by application software to achieve a certain tasks. It may already be stored on one or more nodes, or it may come with the user job. The data is able to migrate from node to node depending on policy that enforced by the resource broker. Data could also be static or dynamic: static data contains domain-specific information that has no meaning or value outside the domain borders, is relatively long-lived in comparison to temporary data, and has a longer period of validity (such as genetic data), while dynamic data can change in size, information content and interactions with other distributed data structures during their lifetime.

2.5 Application Software

The application software is a collection of software programs such as scientific application and multimedia application, which is installed on the grid nodes. A program is software that performs actions by executing a set of instructions to be carried out by a node. It is usually a process or set of processes that possibly run concurrently or with other processes. The basic function of application software is to use the grid environment to allow the job to run on an available resource on the grid.

3 Linguistic Support

In this section we describe our linguistic support for JSDL. In the extension, we introduce a concurrency construct which will aid in the description of large scale grid applications. In addition, we describe a powerful construct that relates both event and time. In what follows, we describe the role of each.

3.1 Job Granularity

An application is a collection of individual jobs working together to fulfill a task. The term "application" is used at the highest level of work on the grid. IBM [13] defines the grid application as *"a collection of work items to solve a certain problem or to achieve desired results using a grid infrastructure. For example, a grid application can be the simulation of business scenarios, like stock market development, that require a large amount of data as well as a high demand for computing resources in order to calculate and handle the large number of variables and their effects. For each set of parameters a complex calculation can be executed. The simulation of a large scale scenario then consists of a larger number of such steps. In other words; a grid application may consist of a number of jobs that together fulfill the whole task"*. A job is a single unit of work executed by an appropriate node in the grid; it may calculate something, execute one or more system commands, operate machinery or move or collect data. IBM [13] defines the job as being *"considered as a single unit of work within a grid application. It is typically submitted for execution on the grid, has defined input and output data and execution requirements in order to complete its task. A single job can launch one or many processes on a specified node. It can perform complex calculations on large amounts of data or might be relatively simple in nature"*. Sometimes the job has one of many other names such as "transaction", "work unit", "submission", and many others, all of which mean the same thing.

We use a notation for describing a parallel application, serial application and network application. The notation we use is:

```
<Application name> :
one or more characters.
<Job name>          :
one or more characters.
  '||' :  parallel composition.
  '; ' :  sequential/serial composition.
```

Grid application description has to contain the application name, job name and application flow. For example:

Weather simulation = (analyse data ; calculate data ; draw graphic)

The weather simulation is an application consisting of the sequentially arranged jobs involving analysing and calculating data and drawing a resulting graphic.

Application flow:
Application flow is the relationships between the jobs that fulfil the general requirements of the application. There are three types of application flow.

3.1.1 Parallel flow

Parallel applications contain several jobs that can all be executed in parallel. This type of application has many advantages for grid computing. For example:

Parallel Application = (job1 || job2 || job3)

In the above example parallel application consists of three jobs, all of which are executed in parallel.

3.1.2 Serial flow

Serial flow is a single thread of job execution where each job has to wait for its predecessor to finish and deliver output data as input to the next job. For example:

Serial Application = (job1; job2 ; job3)

In the above example, a serial application consists of three sequentially arranged jobs.

3.1.3 Network flow

A network application contains several jobs, some of which are executable in parallel while the other jobs are dependant on each other. Such an application contains mixed parallel and serial applications. For example:

Network Application = ((job1||job2) ; (job3||job4))

The above example contains two sequentially composed sub-applications, the first of which is a parallel application consisting of **(job1 || job2)**, while the second is also a parallel application, **(job3 || job4)**; these two applications present a network application.

3.2 Temporal Dimension

Time is an important and interesting issue in grid computing, for many reasons. The aim of grid computing is the exploitation of underutilised resources to achieve faster job execution times. Each job needs a period of time in which to be executed; most resources use time as a charging unit because time it is easily quantifiable. Users therefore have the possibility to provide expected times for job completion.
Users can provide waiting time, execution time, time to repair and deadline. Waiting time is the maximum delay for a job to be executed before which the job cannot be executed due to lack of suitable resource(s). In some cases the job may not be executed immediately after submission, instead being suspended until a suitable resource becomes available. Execution time is the required time for running a job using a resource or group of resources. Deadline is the time, specified by the user, by which execution of their job must be completed using these resources; it is equal to execution time plus waiting time, and can also include the time required to return the final results to the user. Time to repair is the maximum time the job can be delayed for compensating the user for failures to meet the deadline time.
In general, the formula {**job time-out t**} used to determine the execution time operation. This formula means "start job until t "time" finish then terminate job".
Several things complicate the determination of job execution time, including heterogeneous environments, data transfer speeds and communication delays. For this reason, in our approach we give users several ways to describe the time for their job to be executed.

First method: Users determine three times for each individual job to be executed: optimal, preferred and worst execution times. The optimal execution time is the time needed to finish the whole job in an optimal environment and without delay using a resource or a group of resources. The preferred execution time is the time required to finish a whole job using a resource or group of resources. The worst execution time is the time required to finish a whole job in a bad environment or with delays, using a resource or a group of resources. {**job execution-times t1, t2, t3** }**:** this operation constitutes the first method. It allows users to determine three periods of execution time, these being t1 (optimal execution time), t2 (preferred execution time) and t3 (worst execution time). This operation means "start job until t1 finish, if job is finished stop job, if not wait until t2, if job is finished stop job, if not wait until t3, whether or not job is finished stop it". This method yields the following advantages:

- It gives users more flexibility to determine the execution time for the job.
- It achieves good scheduling.
- It reduces communication between the resource broker and resources.
- It returns the result to the user in a short time.
- It saves resource time (i.e. it utilises resources on other jobs if the present job is finished in optimal or preferred execution time) and reduces cost.

Second method: Users determine three times for each individual job: Lowerbound and Upperbound execution time and increment step. Lowerbound execution time is the time required for finishing a whole job in an optimal environment and without any delay using a resource or a group of resources. Upperbound execution time is the time required to finish the whole job in a bad environment or with delay using a resource or a group of resources. Increment step is constant time unit used in addition to the LowerBound until either the job is finished or the UpperBound is achieved. This ratio must be less than the Upperbound and give the Upperbound when added continuously to the Lowerbound.
{**job execution-step-time t1, t2, c**}**:** this operation constitutes the second method. It allows users to determine the Lowerbound, UpperBound and increment step of execution time. t1 represents Lowerbound and t2 Upperbound execution times, and c is increment step. This formula means "start job until t1, if job is finished stop job, if not wait until t2 plus increment step, if job is finish stop job and so on until the UpperBound execution time is achieved, whether or not it is finished stop it." This method uses a large range of expected execution time jobs, because they are dependent on several resources or external data. In some cases, it is difficult for users to determine the job execution time exactly within a tiny range.

3.3 Event

An event is a message to indicate that something has happened at run time in systems (grid computing). An event in grid environment will assist in managing and controlling job execution using distributed resources. Open grid services architecture defines an event as a *"representation of an occurrence in a system or application component that may be of interest to other parties. Standard means of representing, communicating, reconciling, and recording events are important for interoperability"* [12]. It is an important aspect of the overall management of jobs; therefore, all descriptive languages have to be able to cope with (i.e. identify and handle) events. JSDL does not deal with events. Since events are concerned with jobs after submission, describing them will be the concern of lifetime management.
In general, events are generated by users, resource brokers or resources. To describe any event, three things must be identified: ***message events, condition and handling event*** .
Message events User events are messages allowing users to control job execution during run time; they include terminate job, reduce or decrease time. These messages are sent from users to resource brokers. Resource broker events are messages instructing a change in job execution during run time; they include process failure, system performance degradation, network failure and new machines. Resource events are messages indicating that something has happened to resources, such as hardware or software failure and resource unavailability. These messages are sent from resource to resource broker.
Condition Conditions are one or more constraints associated with events, which cause particular handling events to happen. Conditions may include job status.
Handling Event Handling events are responses related to events with their associated conditions. Handling events may be initiation, termination, re-start or completion of one or more jobs. The event operation formula is {**job1 event-condition job2**}**,** where e is the event and c is the condition. This operation means "start job 1 and check the condition, if the event occurs, stop job1 and start job2, if the event does not occur complete job1, don't start job2.

4 XML Encoding of extended JSDL

We follow the same format and the same normative and notational conventions as the original specification of JSDL. JSDL provides two mechanisms for its extension: the use of new attributes or new elements[9]. The namespace prefix used for this schema in the specification is "jsdl-Application". Since this is a proposal, the normative namespace for this schema is not given.
In our proposal, we exploit JSDL capability to de-

scribe applications, times and events. For these features we need to add elements. The applicationjob element, which describes the application flow between jobs, will be added as a child element of the jobdefinition element; the AllEvents Elements, which describes the event that happened in the grid environment, will be added as a child element of the JobDescription element; and the Time Elements (WaitingTime, TimeToRepair, Deadline, ExecutionTimeConstant and ExecutionIncrementStep), describing the time required using resources to execute and complete the job, will be added as a child element of ResourceElements.

4.1 ApplicationJob Element

Definition: This element describes the application identification and application flow between jobs. It contains two main elements which are applicationIdentification and JobsFlow. If this is not present than it is not defined and that mean this is an individual job, in other word this description for individual job.
Pseudo Schema:

```
<ApplicationJob>
        < ApplicationIdentification ...  />
        < JobsFlow .../>
        <xsd:any##other/>*
</ApplicationJob>?
```

4.2 ApplicationIdentification Element

Definition: This element contains all elements that identify the application: ApplicationName, Description, ApplicationAnnotation, and ApplicationProject.
Pseudo Schema

```
<ApplicationIdentification>
     <ApplicationName .../>
     <JobAnnotation .../>*
     <ApplicationProject .../>*
     <Description />*
     <xsd:any##other>*
</ApplicationIdentification>
```

4.3 ApplicationName Element

Definition: This element is a string that specified by a user to determine the name of the application. **Pseudo Schema**

```
<ApplicationName>
          xsd:string
</ApplicationName>
```

4.4 ApplicationAnnotation Element

Definition This element is a string that MAY be specified by a user to annotate the application. If this element is not present then it is not defined. In contrast to the Description element, JobAnnotation MAY contain information that is intended for use by the consuming system.
Pseudo Schema:

```
<ApplicationAnnotation>
          xsd:string
</ApplicationAnnotation>*
```

4.5 ApplicationProject Element

Definition This element is a string specifying the project to which the application belongs. The project could be used by accounting systems or access control systems. The interpretation of the Application Project elements is left to the implementation of the consuming system. If this element is not present then it is not defined.
Pseudo Schema:

```
<ApplicationProject>
          xsd:string
</ApplicationProject>*
```

4.6 JobsFlow Element

Definition This element is a string specified by a user to determine the application flow. The user will enter the notation for describing a parallel application, serial application and network application.
Pseudo Schema:

```
<JobsFlow>xsd:string</JobsFlow>
```

4.7 WaitingTime Element

Definition This element is a positive integer that specifies the maximum seconds for job to wait until it can be executed on a suitable resource(s). In some cases, the job may not be executed immediately after it is being submitted. Thus, it needs to be suspended until a suitable resource becomes available. If this is not present then it is not defined and the consuming system MAY choose any value.
Pseudo Schema:

```
<WaitingTime>
          jsd:nonNegativeInteger
</WaitingTime>?
```

4.8 TimeToRepair Element

Definition This element is a positive integer that specifies the maximum seconds for repair job. If this is not present then it is not defined and the consuming system MAY choose any value.
Pseudo Schema:

```
<TimeToRepair>
        jsdl:nonNegativeInteger _Type
</TimeToRepair>?
```

4.9 Deadline Element

Definition This element is a time that specifies the Deadline for submission job result to the user or finished the job. If this is not present then it is not defined and the consuming system MAY choose any time.
Pseudo Schema:

```
<Deadline>
        jsdl:nonNegativeInteger
</Deadline>?
```

4.10 ExecutionTimes Element

Definition This element allows User to determine three times for each job execution time which is the optimal execution time, preferred execution time, and worst execution time. If this is not present then it is not defined and the consuming system MAY choose any value.
Pseudo Schema:

```
<ExecutionTimes>
  <OptimalExecutionTime>
      jsdl:nonNegativeInteger
  </OptimalExecutionTime>
  <PreferredExecutionTime>
      jsdl:nonNegativeInteger
  </PreferredExecutionTime>
  <WorstExecutionTime>
     jsdl:nonNegativeInteger
  </WorstExecutionTime>
</ExecutionTimes>?
```

4.11 ExecutionIncrementStep Element

Definition: This element allows User to determine the minimum and maximum times, and the increment step for each job execution time. If this is not present then it is not defined and the consuming system MAY choose any value.
Pseudo Schema:

```
<ExecutionIncrementStep>
        <LowerBoundExecutioTime>
                jsdl:nonNegativeInteger
        </LowerBoundExecutioTime>
        <UpperBoundExecutionTime>
                jsdl:nonNegativeInteger
        </UpperBoundExecutionTime>
        <IncrementStep>
                jsdl:nonNegativeInteger
        </IncrementStep>
</ExecutionIncrementStep>?
```

4.12 IncrementStep Element

Definition This element specifying increment step uses to addition to the LowerBound until either finish the job or achieve the UpperBound. This Increment Step must be less than the UpperBound and give the UpperBound when add it continuously to the LowerBound.
Pseudo Schema:

```
<IncrementStep>
        jsdl:RangeValue_Type
</IncrementStep>
```

4.13 AllEvents Elements

Definition This element describes the events. It contains all event elements, that the user can define them. If this is not present then it is not defined and the resource broker MAY handle the event.
Pseudo Schema:

```
<AllEvents>
        <Event />+
        <xsd:any##other>*
</AllEvents>?
```

4.14 Events Element

Definition This element describes the event occurs, event handlers and conditions. It contains MessageEvent, Condition and EventHandler.
Pseudo Schema:

```
<Event>
        <MessageEvent /> +
        <Condition />*
        <EventHandler ./>+
</Event>+
```

4.15 MessageEvent Element

Definition This element describes the type of event.
Pseudo Schema:

```
        <MessageEvent>
                xsd:string
        <MessageEvent/>
```

4.16 Condition Element

Definition This element describes the condition of event
Pseudo Schema:

```
<Condition>xsd:string<Condition/>
```

4.17 EventHandler Element

Definition This element describes the handle of event
Pseudo Schema:

```
    <EventHandler>
            xsd:string
    <EventHandler/>
```

5 Experiments

We conducted the experiments in our simulation with different grid environments, jobs and application jobs. Each grid environment has at least 100 nodes and each node has different types of application software, data and polices. We built our grid simulation called STRL Grid Simulation on java and oracle, to simulate the resources, data, polices of grid and resource, application software, grid portal, grid resource broker, information service and replica catalogue together with all their functions. The goal of our simulation is to analyse the impact of application jobs, job execution times and events on the grid environment.

5.1 Application Job

Regarding application jobs impact on the grid environment, the simulation shows the time saving in completion of the application job and the time spent by the user in submitting the application job to the grid. Our extension allows the user to submit several jobs (together known as the application job) to the grid in a batch instead of separately. As a result, all jobs in the application job will have the same priority for execution. If they were submitted separately, the deadline for complete of the application job would be greater. The user would also be kept constantly busy with these jobs, especially in the case of serial jobs. The time taken for the application job to be submitted is less than would be required for each job to be submitted separately. Figure 2shows the advantages of our application job extension. It illustrates the time saving for different application jobs and compares it with the traditional way of submitting the jobs separately and sequentially).

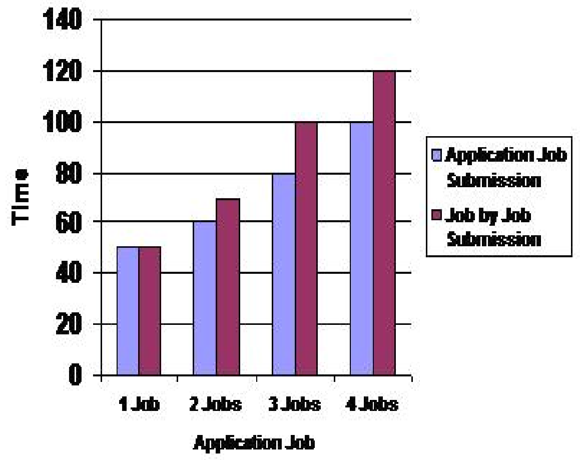

Figure 2: Application job and job by job submission

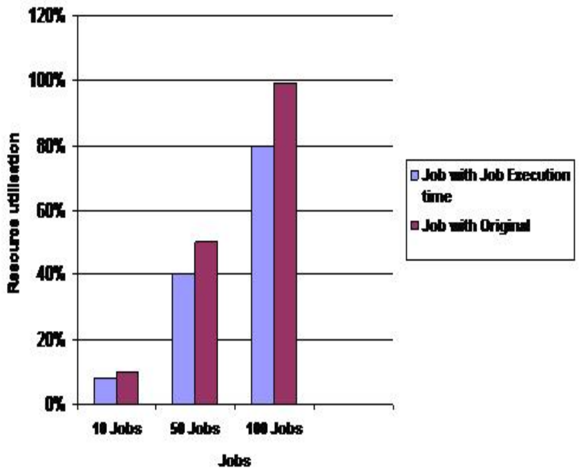

Figure 3: Resource utilisation by job execution time

5.2 Job Execution Time

The new extension for JSDL allows the user to describe more than one set time by which the job must be executed, in order to check if the job is finished or whether it still requires longer. This means that the job may be completed within one of these stipulated times (i.e. early), in which case the grid can exploit the resources previously needed to execute this job in order to carry out another. Figure 3shows the advantages of our Job Execution Time extension. It illustrates the resource utilisation, returns the result and compares it with the original JSDL.

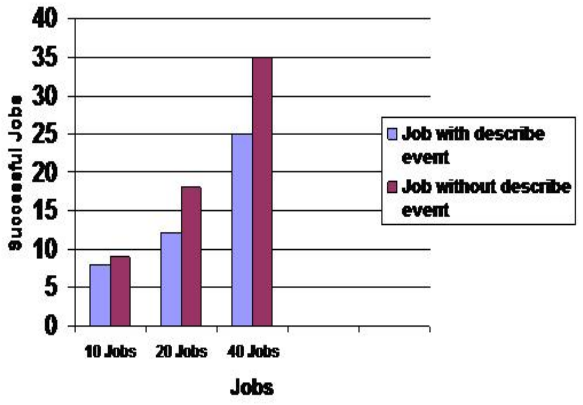

Figure 4: Successful Jobs with event

5.3 Events

In JSDL events allow the user to control the job after submission (i.e. at run time). The event will increase the number of successful jobs shown in Figure 4. For example, one of the tasks in job A is to read the file from hard disk D, but that hard disk has failed, and consequently the grid can not read the file (which means that the job has failed). However, through the event the user can specify the handler for each event, if a particular event occurs, steps (in this case, if hard disk D fails, read from hard disk B).

6 Conclusion

Several languages for describing grid jobs have been developed, such as JSDL. But, these languages have some deficiencies and limitation regarding to large scale application, event, concurrency and time issues. In this paper, we presented a solution to deal with these issues. We introduced concurrency construct which aided the description of large scale grid application. In addition, we described a powerful construct that relates both event and time.We are currently porting our simulation to a physical grid which we have built[14].

References:

[1] M. Baker, R. Buyya, and D. Laforenza, “Grids and grid technologies for wide-area distributed computing,” *Softw. Pract. Exper.*, vol. 32, no. 15, pp. 1437–1466, 2002.

[2] C. K. Ian Foster, *The Grid: Blueprint for a New Computing Infrastructure*, C. K. Ian Foster, Ed. Morgan Kaufmann Publishers; 1ST, 1998.

[3] I. Foster, C. Kesselman, and S. Tuecke, “The anatomy of the grid: Enabling scalable virtual organizations,” *Int. J. High Perform. Comput. Appl.*, vol. 15, no. 3, pp. 200–222, 2001.

[4] M. Li and M. baker, *the grid core technologies*. john wiley & sons, 2005.

[5] P. Stelling, C. DeMatteis, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski, “A fault detection service for wide area distributed computations,” *Cluster Computing*, vol. 2, no. 2, pp. 117–128, 1999.

[6] H. Lee, K. Chung, S. Chin, J. Lee, D. Lee, S. Park, and H. Yu, “A resource management and fault tolerance services in grid computing,” *J. Parallel Distrib. Comput.*, vol. 65, no. 11, pp. 1305–1317, 2005.

[7] *Globus resource Specification Language RSL v.1.0.*, http://www.globus.org.

[8] F. Pacini, *JOB DESCRIPTION LANGUAGE HOWTO*, Datamat SpA, http://server11.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2-Document.pdf.

[9] A. Anjomshoaa, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva, *Job Submission Description Language (JSDL) Specification, Version 1.0*, Global Grid Forum, http://www.gridforum.org/documents/GFD.56.pdf.

[10] *Unicore Web Site.*, http://www.unicore.org.

[11] *HPC-Europa Web Site.*, http://www.hpc-europa.org/.

[12] *Job Scheduling Hierachically (JOSH)Web Site.*, http://gridengine.sunsource.net/josh.html.

[13] B. J. L. F. N. B. C. G. J.-Y. G. R. S. S. S. Yu, *Enabling Applications for Grid Computing with Globus*, International Technical Support Organization, ibm.com/redbooks, 2003.

[14] M. Alfawair, O. Aldabbas, P. Bartels, and H. Zedan, “Grid evolution,” *IEEE International Conference on Computer Engineering and System.*, 2007.