QoS for Service Based Workflow on Grid

L Guo, A S McGough, A Akram, D Colling, J Martyniak, M Krznaric
London e-Science Centre
Imperial College London
London, UK
{asm, aakram, liguo}@inf.ed.ac.uk
{d.colling, janusz.martyniak}@imperial.ac.uk

Abstract

As the main computing paradigm for resource-intensive scientific applications, Grid enables resource sharing and dynamic allocation of computational resources, promotes access to distributed data, operational flexibility and collaboration, and allows service providers to distribute both conceptually and physically to meeting different requirements. Large-scale grids are normally composed of huge numbers of components from different sites. This increases the requirements of workflows and Quality of Service (QoS) upon these workflows as many of these components have real-time requirements. In this paper, we describe a QoS-aware workflow management system(WfMS) from GridCC project[7] and show how our WfMS ensures workflows meet the pre-defined QoS requirements and optimise them accordingly.

keywords: *Grid, Quality of Service (QoS), Web Services, Service Oriented Architecture (SOA)*

1. Introduction

In order to perform constructive science a scientist will in general need to perform multiple tasks in order to achieve their goal. These may include such things as configuring an instrument, collecting and storing relevant data from the instrument, processing of this information and potentially further iterations of these tasks. This can be seen as a set of tasks which interact with each other in order to achieve the final result and can be considered as a workflows.

Workflow management systems (WfMSs) have been used to support various types of e-science workflows on Grid for a while and as the Grid has evolved into a Service-Orientated Architecture (SOA)[2] with Web Services[3] emerging as the de-facto communication mechanism. Worflow languages such as BPEL[4], WS-Choreography[5] are powerful languages for developing workflows based on Web Services. However, the development and execution of workflows within the Grid is a complex process due to the mechanisms used to describe them and the Web Services they are based upon. The selection of the best resources to use within the Grid is complicated due to its dynamic nature with resources appearing and disappearing without notice and the load on these resources changing dramatically over time. These are issues that the scientist will, in general, not wish to be concerned about. Quality of Service (QoS) constraints for interactions between the different Grid components thus becomes crucial to enable resources oriented workflows. In such workflow processes, service providers and consumers define a binding agreement or contract between the two parties, specifying quality of service (QoS) properties such as response times of particular resources such as instruments elements, available storage elements (disk space, memory) for certain tasks, etc. Management of such QoS directly impacts success of parties' participating in the coordination processes. A good management of quality leads to the creation of quality products and services, which in turn fulfills users' expectations and achieves their satisfaction. Therefore, when services are created or managed using workflow processes, the underlying WfMS should be able to accept, monitor, and control the QoS provided by users.

In this paper we present those parts of the GRIDCC[7]

architecture responsible for providing QoS support in WfMS to show how QoS component can be possibly used in Grid environment. In Section 2 we present related work. Section 3 gives the brief analysis of Grid Qos requirements. The architecture our QoS based WfMS is explained in Section 4. Section 4 also contains a more detailed breakdown of our system components: the workflow editor-section 4.1, performance repository-section 4.2, planner-section 4.3 and observer-section 4.4. We conclude our work in Section 5.

2. Related Work

Many Gird workflow systems and tools exist, such as: Askalon[8], DAGMan[9], GridFlow[10], GridbusWorkflow [11], ICENI[12, 13], Pegasus[14], and Triana [15]. Yu and Buyya[16] present a detailed survey of existing systems and provide a taxonomy which can be used for classifying and characterising workflow systems. We use elements of the taxonomy related to Quality of Service (QoS) and workflow modelling for comparing our approach to other systems. A prominent feature of adding instruments to the Grid is the need for real-time remote control and monitoring of instrumentation. Users and applications will be allowed to make Advance Reservation (AR) of computational resources, storage elements, network bandwidth and instruments. AR guarantees availability of resources and instruments at times specified[17]. In ICENI, the scheduling framework supports AR by using the meta-data of application components together with corresponding historical data stored for the purpose of performance analysis and enhancement. This approach is also used in Pegasus, GridbusWorkflow and Askalon. Some systems such as Askalon, DAGMan, ICENI, GridFlow and Gridbus Workflow allow users to define their own QoS parameters and to specify optimisation metrics such as application execution time, desired resources and economical cost. We are using a similar mechanism for performance enhancement and estimation taking into account the real-time QoS constraints.

Worflow languages such as BPEL are powerful language for developing workflows based on Web Services. But it doesn't provide a mechanism for describing QoS requirements. Our approach to overcome this has been to develop a partner language to use with BPEL. Instead of defining a new language for workflows with QoS requirements we use a standard BPEL document along with a second document which points to elements within the BPEL document and annotates this with QoS requirements. This allows us to take advantage of standard BPEL tooling for execution and manipulation as well as provide QoS requirements. As we use XPath[18] notation to reference elements within the BPEL document our approach can be easily adopted to other languages based on XML.

3. Workflow with Quality of Service Requirements on Grid

Quality of Service (QoS) is a broad term that is used in this context to denote the level of performance and service that a given client will experience at a specific point in time, when invoking a given specific operation on a Web service instance. QoS support refers to the possibility that a service instance is capable of offering a performance level which satisfy the requirements of a client. QoS is particularly relevant to many of the Grid applications, which are based on collaborative tools and on the "real-time" interaction with Instrument Elements.

QoS support is paramount given the inherent shared nature of Grid services, and the limited capabilities (in terms of hardware and software resources) that are typically available to satisfy a client's request, especially for those more desirable services. In this environment, an aggressive best-effort usage of services can consequently cause denial of service problems.

We believe that the possibility of supporting client requirements according to two complementary approaches:

- **loose (soft) guarantees:** The provisioning of loose guarantees consists of the capability of clients, to select service instances that can potentially provide a best-effort QoS profile meeting the client's requirements. This is based on previous measurements of that service on similar machines. From these previous measurements it may be possible to estimate performance values for the service. Loose guarantees are delivered on a best-effort basis, and for this reason, they generally do not require the prior establishment of a Service Level Agreement (SLA) and the consequent negotiation of a contract. Loose guarantees are suitable to those application that can perform adaptation and self-healing operations to cope with requirements that are not met in practice e.g. When a client can simply adapt by switching to an alternative service, or for situations where a statistical approach to QoS is sufficient e.g. 60% of the sensors must switch on with 20s.
- **strict (hard) guarantees:** Strict QoS requires the certainty of the delivery of the prescribed level of service, which needs to be agreed upon through signaling and negotiation processes involving both the client and the service provider. Strict guarantees require an enforcement through fabric-layer mechanisms that are service-specific. Reservation of a given amount of resources (for example, RAM memory, disk space, network bandwidth, etc.) is one of the most popular mechanisms for the support of strict guarantees. The reservation service provider is responsible of keeping information about resource availability over time, of ensuring that the total amount of resources allocated does

not exceed the maximum amount of resources that can actually be locked, and of supporting resource-specific locking mechanisms to guarantee exclusive access to reservation users.

QoS provisioning of our work relies on both strict and loose QoS guarantees. While hard QoS requires the making of reservations on the resources to be used, soft QoS requires the user (or planner acting on the users behalf) to model the execution of the services they require. These models may vary from from the simple, when only a single service is required to the extremely complex when several services are required as part of a workflow. In such cases it may be required that a reservation is required even to satisfy loose QoS constraints.

4. Architecture of QoS Aware WfMS

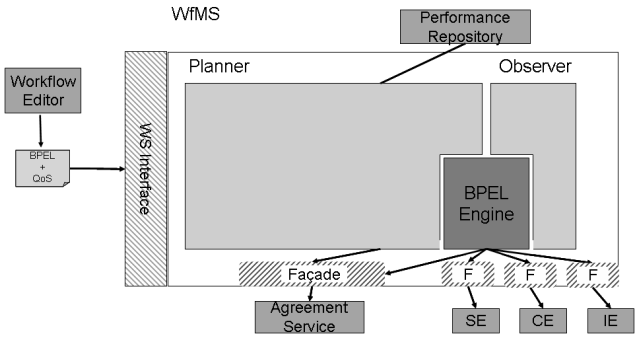

Figure 1: Components of QoS Aware Workflow Management System

Figure 1 shows the overall architecture of the QoS based WfMS on. The Workflow Editor is the user interface for producing BPEL workflows and QoS documents. The Workflow Management Service contains the Workflow Planner and Observer along with the Workflow Engine. The Workflow Engine may communicate with various Grid services such as Compute Elements (CE), Storage Elements (SE) and, as defined within the GRIDCC project, the Instrument Elements (IE)Ca Web Service abstraction of an instrument along with the Agreement Service (AS) for brokering reservations with other services. The Performance Repository (PR) contains information about previous executions of tasks on the Grid.

On receiving a workflow and QoS document produced by the workflow editor, the WfMS must decide, based on information from the PR along with an understanding of the workflow, if the submitted request can be achieved within the provided QoS constraints. In order to achieve this the WfMS may choose to manipulate the workflow. This may include making reservations for tasks in the workflow and/or changing the structure of the workflow. The Workflow Engine is invoked and is responsible for the ordered execution of the workflow tasks, with the Observer monitoring progress.

In the following sub-sections, each of the components inside the workflow management system is explained in detail.

4.1. Workflow Editor

Producing BPEL documents and their associated Qos documents are both tedious and error prone. In order to facilitate the production of the input documents (BPEL and QoS) of our QoS aware WfMS, we provide use a workflow editor. Workflow editor provides a pallet grouping various QoS constraints. BPEL4WS specification doesnt define the quality issues related to overall workflow or individual Web services. QoS constraints can be coupled with different BPEL4WS activities particularly $< invoke >$ activity. QoS constraints for the workflow are specified in the separate file rather than embedding them in the BPEL4WS script. Our workflow editor differs from existing editors in the following aspects:

- Portal Based Editor: All open source and commercial workflow editors require installation and configuration of the editors before any use. Installation of workflow editor means access to local file system as admin, which may not be available. This JSR 168[19] compliant workflow editor will provide the editing tool on demand. Users can edit and save the workflow on the server and can access them from anywhere and whenever required. Use of JSR 168 complaint portal and portlet allows mixing the presentation layer of the editor with the back end business logic implemented in Java. Browser based clients have the inherent advantage as they do not need to be upgraded on the client side and provide a pervasive access mechanism.
- Drag and Drop: Our workflow editor provides a drag and drop facility to drag various BPEL4WS activities, Web services or operations from Web service registry, Quality of Service (QoS) constraints from QoS panel and variables from XML Schema registry into workflow designer pane. Dragging of different components on designer pane either updates the BPEL4WS script or create corresponding QoS instance. TheWorkflow Editor is based on the ActionScript 3.0[20] and MXML[21]; the core components of Macromedia Flash. ActionScript is a scripting language supporting various features only available in the desktop applications and MXML is the XML-based markup language for the presentation layer.

- Hiding Complexities: A generic BPEL4WS workflow designer demands advanced skill from workflow designers; i.e. thorough understating of Web Services architecture and different types and styles of Web Services, expertise in managing XML files from various namespaces, experience of any XML query specification such as XPath or XQuery; and familiarity with the BPEL4WS specification. Web Services and BPEL4WS specification have dependencies on other related specification e.g. WS-Addressing; which further complicates the designing process. The GRIDCC editor hides these complexities from the scientist and researchers by automating different tasks.

4.2. Performance Repository(PR)

Central to being able to perform any quality of service decisions, whether they be for hard or soft QoS, is information about the resources available. This information needs to be available. The place where this information is stored called the Performance Repository(PR). The PR will serve data of three generic types:

- Static information about how a particular service scales according to factors such as input, expected output and the task that it is running. This scaling information will be also be as function of the resource on which the service is running. For a web service the information may be broken down into different parts of the process of running the service (de-serialisation of the input message, time for the service to run, serialisation of the output message etc) and then stored in this form. This sort of information may be gathered by testing the service with well defined inputs and outputs on a well defined and isolated test-bed. This information may be gathered from the analysis of the performance of real running services.
- Dynamic information about the current state of the resources on the grid. For conventional Grids (such as the EGEE Grid) this will simply be a redirection from the existing IS however for IEs it may involve having listeners subscribed to the IMS of the IE. An important set of information will be on the current networking status between two sites.
- Eventually, the PR will also be able make predictions about future usage. Initially this will be through using knowledge of advanced reservations and ultimately using tools such as Network Weather Service.

  There are two customers for the PR's information, the individual user and the WfMS planner, and both use it in an essentially the same ways. When making a reservation the the PR is interrogated in order to acquire a list of the resources available that can satisfy the QoS demands. For soft QoS constraints the PR provides the input data to the models used to determine whether or not it is possible to manipulate the workflows in such a way as to realise the QoS constraints.

Web services are used as interfaces to enable both end users and WfMS's QoS clients to query, update the information stored in PR. Several available operations from PR are listed in table 1:

operations	**function descriptions**
uploadIE	upload data for particular instrument element
uploadSE	upload data for particular storage element
uploadCE	upload data for particular computing element
requestKnownIEService	request information of particular instrument element if the ID of it is known
requestKnownSEService	request information of particular storage element if the ID of it is known
requestKnownCEService	request information of particular computing element if the ID of it is known
requestIEService	request a list of available instrument elements if they satisfy the users' requirements
requestSEService	request a list of available storage elements if they satisfy the users' requirements
requestCEService	request a list of available computing elements if they satisfy the users' requirements

Table 1: Operations From Web Services of Performance Repository

4.3. Planner

There are three main components, namely *basic resolver, QoS reserver and constraint resolver* in planner as shown in Figure 2.

These components are chained by the QoS documents that are passed through. These QoS guarantees need to be makable both directly by the user through the client interface(workflow editor) and as part of workflows that are be-

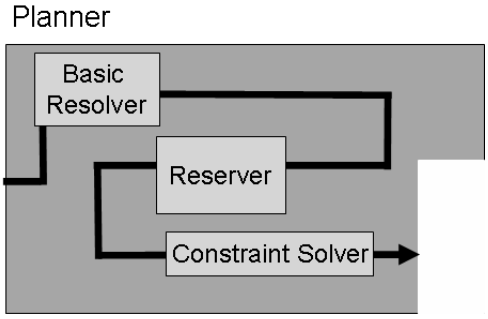

Figure 2: Components of QoS Aware Workflow Management System

ing manipulated by some components (basic resolver, reserver) of the planner. A user submits a BPEL document, which is likely to contain a number of service invocations-referred to as a task. Without loss of generality this document may contain one or more tasks. A separate document is used to describe the QoS requirements placed onto this BPEL document. A QoS requirement document is composed of sets of QoS constraints, which are linked to separate BPEL activities (both basic activities and structure activities) as shown in Figure 3. QoS requirements falls into

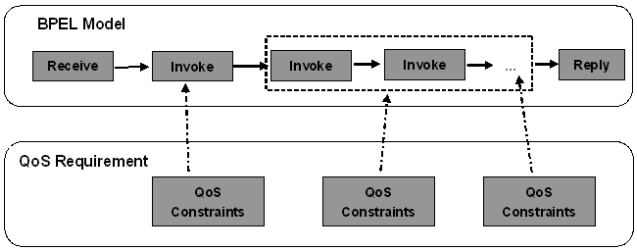

Figure 3: Connecting QoS Document and BPEL Model

four categories according to the range of activities in BPEL models that it specifies:

- **Global requirement** is a QoS document that specifies single global QoS requirements. A simple example is given as follows (for simplicity, all the unnecessary technical details are omitted):

$$
\begin{aligned}
&<?xml\ version = "1.0"\ encoding = "UTF-8"?>\\
&<QoSRequirements>\\
&\quad<QoSConstraint>\\
&\quad\quad<XpathReference>\\
&\quad\quad\quad/process\\
&\quad\quad</XpathReference>\\
&\quad\quad<Resources>\\
&\quad\quad\quad<CPUSpeed>2048000</CPUSpeed>\\
&\quad\quad</Resources>\\
&\quad\quad<MaxDurationTime>100</MaxDurationTime>\\
&\quad\quad<Reliability>100</Reliability>\\
&\quad</QoSConstraint>\\
&</QoSRequirements>
\end{aligned}
$$

In the above example, there is a single *XpathReference* pointing to the entire process of the BPEL document. Thus everything within this process must match these QoS requirements. In this case the overall time should be less than 100 seconds, all CPUs should be 2Ghz and all resources should be fully reliable.

- **Single invoke activity requirement** is a QoS document that specifies requirement on a particular BPEL invoke activity. Only that invoke activity needs to satisfy the QoS requirement specified as shown in the following example:

$$
\begin{aligned}
&<?xml\ version = "1.0"\ encoding = "UTF-8"?>\\
&<QoSRequirements>\\
&\quad<QoSConstraint>\\
&\quad\quad<XpathReference>\\
&\quad\quad\quad/process/sequence[1]/invoke[1]\\
&\quad\quad</XpathReference>\\
&\quad\quad...\\
&\quad</QoSConstraint>\\
&</QoSRequirements>
\end{aligned}
$$

In this case there is a single *XpathReference* pointing to a single invoke element of the BPEL document. Thus everything within this invoke must match these QoS requirements.

- **Multiple invoke activities requirement** is a QoS document that specifies requirement on a set of invoke activities. In a single QoS constraint, several *XpathReferences* pointing to different invoke activities in a BPEL model are defined.

$$
\begin{aligned}
&<?xml\ version = "1.0"\ encoding = "UTF-8"?>\\
&<QoSRequirements>\\
&\quad<QoSConstraint>\\
&\quad\quad<XpathReference>\\
&\quad\quad\quad/process/sequence[1]/invoke[1]\\
&\quad\quad</XpathReference>\\
&\quad\quad<XpathReference>\\
&\quad\quad\quad/process/sequence[2]/invoke[2]\\
&\quad\quad</XpathReference>\\
&\quad\quad...\\
&\quad</QoSConstraint>\\
&</QoSRequirements>
\end{aligned}
$$

- **Separate QoS requirements** specifies several QoS constraints elements which might be independent on each other in a QoS document.

$$
\begin{aligned}
&<?xml\ version = "1.0"\ encoding = "UTF-8"?>\\
&<QoSRequirements>\\
&\quad<QoSConstraint>\\
&\quad\quad<XpathReference>\\
&\quad\quad\quad/process/sequence[1]/invoke[1]\\
&\quad\quad</XpathReference>\\
&\quad\quad<Resources>\\
&\quad\quad\quad<CPUSpeed>2048000</CPUSpeed>\\
&\quad\quad</Resources>\\
&\quad</QoSConstraint>\\
&\quad<QoSConstraint>\\
&\quad\quad<XpathReference>\\
&\quad\quad\quad/process/sequence[2]/invoke[2]\\
&\quad\quad</XpathReference>\\
&\quad\quad<MaxDurationTime>100</MaxDurationTime>\\
&\quad\quad<Reliability>100</Reliability>\\
&\quad</QoSConstraint>\\
&</QoSRequirements>
\end{aligned}
$$

4.3.1. Basic Resolver(BR) The first functional component of planner is basic resolver. Given a QoS document which states that a resource needs reserving, though the selection of the resource has not been determined, it will query the Performance Repository to select a resource to make a reservation on. No inspection of the BPEL document is done at this stage. The QoS element requesting a reservation without a named resource is then changed into an element requesting a reservation with a named resource. This can then be passed onto the QoS Reserver for making the actual reservations. Supposing that there is an incoming QoS document in which user specifies the description of required resources as follows:

```
<?xml version = "1.0" encoding = "UTF − 8"? >
< QoSRequirements >
    < QoSConstraint >
        < XpathReference >
          /process/sequence[1]/invoke[1]
        < /XpathReference >
        < ReservationRequired >
            < StartTime > ... < /StartTime >
            < EndTime > ... < /EndTime >
            < Resources >
              < CPUSpeed > 2048000 < /CPUSpeed >
            < /Resources >
        < /ReservationRequired >
    < /QoSConstraint >
< /QoSRequirements >
```

The basic resolver first sends the resource descriptions (< $CPUSpeed >$ 2048000 $< /CPUSpeed >$ for this example) to the performance repository. PR then performs resource search based on its information and returns a list of possible resource endpoints that satisfy the resource requirements. The QoS document is finally rewritten by BR before it is passed to QoS reserver:

```
<?xml version = "1.0" encoding = "UTF − 8"? >
< QoSRequirements >
    < QoSConstraint >
        < XpathReference >
          /process/sequence[1]/invoke[1]
        < /XpathReference >
        < ReservationRequired >
            < StartTime > ... < /StartTime >
            < EndTime > ... < /EndTime >
            < EndPoints >
                < EndPoint > cpu2.doc.ic.ac.uk < /EndPoint >
                < EndPoint > cpu3.doc.ic.ac.uk < /EndPoint >
            < /EndPoints >
        < /ReservationRequired >
    < /QoSConstraint >
< /QoSRequirements >
```

4.3.2. QoS Reserver The QoS Reserver inspects incoming QoS documents looking for requests for making reservations with known resources. The Agreement Service is contacted in order to make these reservations. The QoS document is then updated to indicate that the reservation has been made and records the unique token used to access the reservation. All requests for reservations are processed here. In the current implementation if reservations cant be satisfied then the whole document will be thrown back to the user to select new reservation times. Once we have components capable of selecting timings for reservations internally the workflow and QoS will be returned to this component.

4.3.3. Constraint Resolver This is the component that provides scheduling functionality into the planner. This implementation is based on a constraints equation method. The workflow along with the QoS requirements is converted into a set of constraint equations. Information from the Performance Repository is used to solve these constraint equations. Different utility functions can be adopted according to different users' requirements. How the utility functions are defined totally relies on specific application domain thus we are not trying to discuss them in any detail in this paper.

4.4. Observer

The Observer receives a completed copy of the QoS document, and the BPEL document, at the same time that the Workflow Engine receives the BPEL document. This QoS document contains timing information as to how the workflow is expected to execute. The Observer is then able to monitor the progress of the executing workflow, through status calls, in order to ensure that the workflow executes as desired. If the workflow deviates from the expected plan, in either direction, the observer is able to invoke the planner to re-compute the workflow in order to achieve the desired QoS requirements. Currently we are supporting the dynamic changing of endpoints within the BPEL documents. All endpoints within the BPEL document are defined to be assigned from variables which are assignable through calls to the running workflow, via calls to WSDL methods. If a workflow endpoint is determined to no longer be appropriate a call can be made to the appropriate method to change the endpoint of the service. This allows for changes to the endpoints without the need to compute these at the time of the call. We are investigating other such approaches to dynamically alter the BPEL workflow during execution.

5. Conclusion and Future Work

In this paper we have presented how the function of quality of services are used within the GRIDCC project to allow for integration of existing Grid technology with that of instruments. Workflows are defined through an editor which allows the augmentation of QoS requirements, defining the users expectations for the execution. The WfMS provides a mechanism for building QoS on top of an existing commodity based BPEL4WS engine. Thus allowing us to provide a level of QoS through resource selection from apriori information along with the use of advanced reservation. The workflow editor and Observer are areas of current development within the project. We are working with application scientists from the GRIDCC project to abstract the ed-

itor from the BPEL / QoS languages and make them more accessible to the scientists. We are investigating other techniques which will allow us to dynamically change the execution of the BPEL workflow once deployed to the engine.

References

[1] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, July 1998.

[2] D. Nickull and F. McCabe. Soa reference model. http://www.oasis-open.org/committees/ tchome.php?wgsoa-rm.

[3] W3C. Web Service. http://www.w3.org/TR/ws-arch/.

[4] T Andrews and F Curbera and H Dholakia and Y Goland and J Klein and FLeymann and K Liu and D Roller and D Smith and S Thatte and ITrickovic and SWeerawarana. Business Process Execution Language forWeb services version 1.1, (BPEL4WS). http://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, May 2003.

[5] David Burdett and Nickolas Kavantzas. http://www.w3.org/TR/ws-chor-model/.

[6] W.M.P. van der Aalst and L. Aldred and M. Dumas and A.H.M. ter Hofstede. *Design and Implementation of the YAWL system.* In Proceedings of The 16th International Conference on Advanced Information Systems Engineering (CAiSE 04), Riga, Latvia, june 2004. Springer Verlag.

[7] D.J. Colling, L.W. Dickens, T. Ferrari, Y. Hassoun, C.A. Kotsokalis, M. Krznaric, J. Martyniak, A.S. McGough, and E. Ronchieri. *Adding Instruments and Workflow Support to Existing Grid Architectures.* volume 3993 of Lecture Notes in Computer Science, pages 956C963, Reading, UK, April 2006.

[8] 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(2-4):143C169, 2005.

[9] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. *Condor-A Distributed Job Scheduler.* Beowulf Cluster Computing with Linux. The MIT Press, MA, USA, 2002.

[10] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. *GridFlow: Workflow Management for Grid Computing.* In Proceedings of 3rd International Symposium on Cluster Computing and the Grid (CCGrid), Tokyo, Japan. IEEE CS Press, Los Alamitos, 12C15 May 2003.

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

[12] A. Mayer, S. McGough, N. Furmento,W. Lee, S. Newhouse, and J. Darlington. *ICENI Dataflow and Workflow: Composition and Scheduling in Space and Time.* In UK e-Science All Hands Meeting, Nottingham, UK, pages 894C900. IOP Publishing Ltd, Bristol, UK, Sep. 2003.

[13] S. McGough, L. Young, A. Afzal, S. Newhouse, and J. Darlington. *Workflow Enactment in ICENI.* In UK e-Science All Hands Meeting, Nottingham, UK, pages 894C900. IOP Publishing Ltd, Bristol, UK, Sep. 2004.

[14] E. Deelman, J. Blythe, Y. G. C. Kesselman, G. M. andK. Vahi, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. *Mapping Abstract Complex Workflows onto Grid Environments.* Journal of Grid Computing, 1(1):9C23, 2003.

[15] I. Taylor, M. Shields, and I. Wang. *Resource Management for the Triana Peer-to-Peer Services.* In J. Nabrzyski, J. M. Schopf, and J. Weglarz, editors, Grid Resource Management - State of the Art and Future Trends, pages 451462. Kluwer Academic Publishers, 2004.

[16] J. Yu and R. Buyya. *A taxonomy of workflow management systems for grid computing.* GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, March 10, 2005.

[17] S. Andreozzi and T. Ferrari and S. Monforte and E. Ronchieri. *Agreement-based Workload and Resource Management.* In Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing, Melbourne, Australia, Dec. 2005. IEEE Computer Society.

[18] Xml path language (xpath) version 1.0. Technical report, 1999.

[19] Alejandro Abdelnur and Stefan Hepper. Porlet specification (jsr-168). http://jcp.org/aboutJava/ communityprocess/review/jsr168/.

[20] Gary Grossman and Emmy Huang. ActionScript 3.0 overview. http://www.adobe.com/devnet/ actionscript/articles/actionscript3 overview.html, June 2006.

[21] Christophe Coenraets. An overview of MXML: The Flex markup language. http://www.adobe.com/ devnet/flex/articles/paradigm.html, Mar. 2004.