CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Brunel University Research Archive

GRIDCC: Real-time Workflow system

A.Stephen McGough, Asif Akram, Li Guo, Marko Krznaric,

Luke Dickens

London e-Science Centre

Imperial College London, London, UK

asm@doc.ic.ac.uk, aakram1@doc.ic.ac.uk,

liguo@doc.ic.ac.uk, marko@doc.ic.ac.uk, lwd03@doc.ic.ac.uk

David Colling,

Janusz Martyniak

High Energy Physics Group

Imperial College London, London, UK

d.colling@imperial.ac.uk, janusz.martyniak@imperial.ac.uk

Roger Powell, Paul Kyberd,Chenxi Huang

Brunel University

School of Engineering & Design

Uxbridge, UK

Roger.Powell@brunel.ac.uk, Paul.Kyberd@brunel.ac.uk

Constantinos Kotsokalis,
Panayiotis Tsanakas
Computing Systems Laboratory
National Technical University of Athens
Athens, Greece
ckotso@cslab.ece.grnet.gr, panag@cs.ntua.gr

July 29, 2007

Abstract

The Grid is a concept which allows the sharing of resources between distributed communities, allowing each to progress towards potentially different goals. As adoption of the Grid increases so are the activities that people wish to conduct through it. The GRIDCC project is a European Union funded project addressing the issues of integrating instruments into the Grid. This increases the requirement of workflows and Quality of Service upon these workflows as many of these instruments have real-time requirements. In this paper we present the workflow management service within the GRIDCC project which is tasked with optimising the workflows and ensuring that they meet the pre-defined QoS requirements specified upon them.

1 Introduction

The Grid[14] has emerged in recent years as paradigm for allowing users, potentially as members of distributed communities, to obtain access to large quantities of computing resources while providing a forum in which resource owners make their services available to a wide audience. The Grid has evolved into a Ser-

vice Orientated Architecture (SOA) [22] with Web Services [29] emerging as the de-facto communication mechanism. With the increase of users and communities within the Grid there is a drive to support different types of resources. Thus far most Grids have focused on making computational and, more recently, data resources available. However, for the Grid to gain mass adoption within the wider scientific community there is a need to integrate instruments into the Grid. In this paper we define an instrument to be any piece of equipment controlled through a computer interface such as telescopes, particle accelerators or power stations. In the rest of this paper we shall discuss scientific instruments, however, the principals may be mapped easily to other instrument types.

The desire for scientists to perform experiments through the Grid increases dramatically the need for workflows and real-time-support within the Grid. 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. Many scientists now seek to automate the process of bringing these tasks together through the use of workflows. Here a workflow is seen as a set of tasks along with a description of how these task must interact in order to achieve the final result.

We present here the Grid-Enabled Remote Instrumentation with Distributed Control and Computation (GRIDCC) project [8]. More specifically we present here the real-time-control of existing Grid resources and instruments elements from the project. The GRIDCC project is a 3 year project funded by the European Union which started in September 2004. There are 10 project partners from Greece, Italy, Israel and the United Kingdom. GRIDCC is extending the Grid to include access to, and control of, distributed instruments. Instruments

work in real-time and their successful operation often requires rapid interaction with conventional computing/storage resources and/or other instruments. The real-time and interactive nature of instrument control provides a critical requirement for the definition of acceptable Quality of Service (QoS) constraints for interactions between the different Grid components.

In this paper we present those parts of the GRIDCC architecture responsible for providing workflow and QoS support. In Section 2 we present related work. Section 3 presents the architecture for a real-time QoS aware workflow managment system along with the workflow pipeline. This is followed by a more detailed breakdown of the architecture: the Workflow Editor with QoS support – Section 4, QoS aware Workflow optimisation – Section 5, Reservation services – Section 7 and Performance Repository – Section 6. In Section 8 we evaluate the performance of our architecture. We conclude in section 9.

2 Related Work

Many Grid workflow systems and tools exist, such as: Askalon [13], DAGMan [26], GridFlow [6], Gridbus Workflow [34], ICENI [18, 21], Pegasus [10], and Triana [27]. Yu and Buyya [35] 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 [24]. In ICENI, the scheduling framework supports AR by using the meta-data of application components to-

gether with corresponding historical data stored for the purpose of performance analysis and enhancement [20, 32, 33]. This approach is also used in Pegasus, Gridbus Workflow 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 Business Process Execution Language for Web Services (BPEL4WS) [25], YAWL [30] and WS-Choreography [9] are powerful languages for developing workflows based on Web Services. However, they do not provide a mechanism for describing QoS requirements. Our approach to overcome this has been to develop a partner language to use with BPEL4WS. Instead of defining a new language for workflows with QoS requirements, or embedding QoS requirements within a language such as BPEL4WS, we use a standard BPEL4WS document along with a second document which points to elements within the BPEL4WS document and annotates this with QoS requirements. This allows us to take advantage of standard BPEL4WS tooling for execution and manipulation as well as provide QoS requirements. As we use XPath [1] notation to reference elements within the BPEL4WS document our approach can be easily adopted to other (workflow) languages based on XML.

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 con-

cerned about. The use of advanced reservations can help to reduce the amount of uncertainty within the Grid. However, in general, the selection of the best service to reserve is a complex process. We adopt the approach of a workflow pipeline presented by McGough et al. [19] to decompose this problem into more manageable steps.

Huang et al.[17] have proposed a just-in-time workflow optimisation service in which a workflow calls a proxy service rather than the actual service, which then determines the best resource to use. Our approach improves on this by allowing the end points in a BPEL4WS document to be updated on the fly thus "hiding" the cost of computing the best service in line with execution.

Thus far we have used simple workflow optimisation heuristics however we see that our approach can be easily adapted to use more optimal approaches such as stochastic optimisation [31] and overall workload optimisation [3].

3 Real-time QoS aware workflow managment system

The GRIDCC Workflow Management Service (WfMS) is defined as an end-to-end workflow pipeline. The workflow pipeline is illustrated in Figure 1. The scientist develops a workflow within the Workflow Editor, which presents an abstracted view, thus hiding much of the complexity. Rather than presenting a generic workflow editor the scientist is presented with an editor tailored more to their specific needs. Instruments are presented as instrument entities rather than generic Web Services. It is then the task of the Workflow Editor (software) to generate appropriate workflow language documents based on the scientists design. The user may also have a number of QoS requirements, these should be specified alongside the workflow description and submitted to the next stage.

Further discussion of the Workflow Editor can be found in Section 4.

Once the workflow has been specified the process of selecting the most appropriate set of resources is performed. This is the task of the Workflow Planner. Within the QoS document the scientist may have specified constraints. These may be of three types:

- *Specified Resource*: The scientist has specified the resource that should be used. This may be due to the fact that the scientist has placed a sample onto a specific instrument at a given time. This informs the planner of fixed requirements on the workflow.

- *Specified Requirement*: The scientist may not know the best resource to use for a given task though knows that a reservation will be required. This could be for ensuring enough space is made available to store data. It is the role of the planner to convert this into a Specified resource.

- *Unspecified*: The resource is not specified nor is there a specific requirement on this resource. The planner may choose to convert this into a specified resource in order to achieve some overall QoS requirement.

Each QoS requirement may be strict (hard), in which case requirements should be considered mandatory – a hard constraint of 'this must take less than five minutes' must be adhered to or the whole workflow rejected. Alternatively the requirement may be soft (loose) in which case a value is given indicating the confidence required – a soft constraint of 'this should complete in less than five minutes in 80% of cases' can be accepted as long as the Planner believes that it can achieve this in 80% of cases. The Workflow Observer is designed to deal with cases where the workflow deviates from that planed. The Observer monitors the progress of the workflow and, if the tasks deviate from the plan, will trigger the Planner to modify the workflow in order to increase the chance

of the workflow completing successfully. This is an active area of research. A full breakdown of the Workflow Planner/Observer can be found in Section 5.

3.1 The Grid architecture

Figure 3 shows the overall architecture of the real-time-system used within the GRIDCC project. In the architecture the Virtual Control Room (VCR) is the user interface to the Grid. The Workflow Editor is intergrated within the VCR. The Workflow Management Service (WfMS) contains the Workflow Planner and Observer along with the Workflow Engine, in the case of GRIDCC this is the ActiveBPEL [4] 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) – a 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 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 [19]. The Workflow Engine is invoked and is responsible for the ordered execution of the workflow tasks, with the Observer monitoring progress.

Figure 2 illustrates in more detail the internal structure of the WfMS. A BPEL4WS document along with an associated QoS document is received. This can first be validated before being passed through various stages (outlined further in Section 5) before the BPEL4WS document is submitted to the

BPEL4WS engine and the revised QoS (and BPEL4WS) is submitted to the Observer. In order to allow the BPEL4WS engine to communicate with the existing Grid resources inline convertors and facades are used to manipulate the Web Services calls.

4 Workflow Editor with QoS support

In recent years, Web Services have been established as a popular 'connection technology' for implementing a SOA. The interface required for Web Services is described in a WSDL file (Web Service Description Language [12]). Integration of two or more services into a more complex service can be achieved through 'Service Orchestration'. Workflows are managed by a workflow engine, which orchestrates the interactions between services.

The GRIDCC project is aiming to provide a Web based workflow editor based on BPEL4WS [25] 1.1 specification. BPEL4WS is generally regarded as the de-facto standard for composing and orchestrating workflows in the business environment and is now achieving wide adoption within the scientific community. The goal of the BPEL4WS specification is to provide a notation for specifying business process behavior based on Web Services. BPEL4WS builds on top of the Web Service stack and is defined entirely in XML, compatible with other emerging technologies including WS-Reliable Messaging [23], WS-Addressing [11], WS-Coordination [16] and WS-Transactions [28] (BPEL4WS is not limited to only these technologies).

4.1 Business Process Execution Language

BPEL4WS provides a rich vocabulary for the description of business processes supporting two different types of process declaration. An 'Abstract Process' specifies the messages exchanged between participating services without expos-

ing the internal details of the process. An 'Executable Process' extends abstract process to allow specification of the exact details of a process workflow.

The BPEL4WS 1.1 specification defines various building blocks, known as 'activities'. These activities can be used to specify an executable process' business logic to any level of complexity. Activities can be grouped into: *basic*, such as procedural steps (e.g. invoke, receive and assign), *structured*, which control the flow of the workflow (sequence, switch and foreach) and *special* such as terminate, validate and fault handlers. Within the BPEL4WS workflow process itself, different activities pass information among themselves using global data variables.

Although the BPEL4WS specification is tailored more to the requirements of business processes, there are properties which make BPEL4WS useful in the scientific environment. These properties include Modular Design, Exception and Compensation Handling. Furthermore, scientific services are often subject to change, especially with regard to the data types and service-endpoint locations. Data flexibility is supported using 'generic' or 'loose' Web Service data types. Finally, BPEL4WS specification allows for workflow adaptivity, which is mainly facilitated by the 'empty' activity providing a placeholder for future extensions.

However, the BPEL4WS specification does not provide any direct mechanism to support management and monitoring of a workflow nor does it address Web Service security issues, such as passing credentials through a BPEL4WS engine. The WfMS uses "deanonymisation" technique in which users identity is passed in the message header and stored in the WfMS. The outgoing message is intercepted to attach user credentials and perform a secure call to an external service.

Generating a workflow document is therefore a process of bringing activities together to achieve the end-goal. Similarly to a low-level programming language

this usually is a complex and tedious process. A number of BPEL4WS editors exist, such as the Active BPEL Editor and the Oracle BPEL4WS Process Manager. However, these are developed for computer scientists who wish to develop workflows and have an extensive knowledge of BPEL, and WSDL.

4.2 Aim of GRIDCC Editor

The flexibility and richness of features in the BPEL4WS specification, brings unavoidable complexities. These complexities hinder the use of standard workflow by scientist and researchers in academic domain. Considering the user requirements we can identify two categories: (1) users executing existing workflow for complex scientific procedures in which they may not be experts; and (2) expert users engineering new or existing processes as workflows. For the first type of users Web Applications are appropriate since executing existing workflows means uploading the workflow, and, supporting parameters and constraints to the server for use by the workflow engine. The second type of user needs rich client software with advanced functionality. This includes comfort features, such as a polished user interface, drag-and-drop, a rich set of keyboard shortcuts, and provision to save/load partial workflows. These partial workflows can then be integrated, as building blocks, to form parts of more complicated and sophisticated workflows. The purpose of the GRIDCC editor is to provide a hybrid solution and address limitations of the BPEL4WS specifications which are normally ignored by existing open source and commercial editors. The GRIDCC 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 editor before use. Installation of a workflow editor means access to local file system (normally) as admin, which may not be available. This JSR 168 [5] compliant workflow editor will provide

the editing tool on demand without the need to install. Users can edit and save the workflow on the server, this can be performed either form their own Web Browser or any other (subject to security constraints) at any time in the future. This provides a powerful framework for the sharing of workflows. The use of a JSR 168 compliant portal and portlet interface allows for a mixing of the presentation layer od the editor (running within a Web Browser) with the back-end workflow editor logic which is implemented in Java and runs on the server side. This allows for a simple Graphical User Interface (GUI) written for a Web Browser to delegate all of the complicated logic, for validating a workflow, to be handled by the server side. This logic is often referred to as the business logic. 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: The GRIDCC editor provides a drag and drop facility which allows the user 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 elements. The Workflow Editor is based on ActionScript 3.0 [15] and MXML [7]; the core components of Macromedia Flash. ActionScript is a scripting language supporting various features, such as drag and drop, in the Web Browser. These features are normally only available through the use of a desktop (installed) application. MXML is the XML-based markup language for the presentation layer.

Hiding Complexities: A generic BPEL4WS workflow editor 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 in following way:

1. A workflow process requires global variables for its proper functioning. Whenever a new process is created a global corresponding `<variables>` element is created at specific location as required by BPEL4WS specification. The `<variables>` element is a registry of different XML specific data types and variables.

2. A workflow orchestrates various Web Services wrapped in `<partnerLinkType>`, `<partnerLinks>` and `<partnerLink>` elements. Each `<partnerLinkType>` element wraps a `<portType>` of the Web Service, which itself wraps various operations. The GRIDCC editor creates the `<partnerLinkType>`, `<partnerLinks>` and `<partnerLink>` when any Web Service is added in the Web Service registry.

3. Operations on the partner Web Services are called through `<invoke>` activity. The `<invoke>` activity specifies the `<partnerLink>`, `<portType>`, `<operation>`, `<input>` and `<output>` elements defined in WSDL or BPEL4WS script. An operation of the Web Service can be dropped directly on a Workflow process from a Web Service registry and the editor itself creates the corresponding `<invoke>` activity by relating various required elements.

4. An `<invoke>` element must specify `<input>` and `<output>` variables to store the values of outgoing and incoming messages. If these variables do not exist then the GRIDCC editor creates the `<variable>` required for successful invocation of the operation in the `<variables>` element (bullet

point 1).

5. The GRIDCC editor also adds "exception handling" activities with `<empty>` activities as a template for flexibility and extensibility. The designer of the workflow can replace these `<empty>` activities with actual business logic.

Ease of use: The GRIDCC editor is easy to use due to its built in error handling. The Workflow Editor validates the actions of the designer and makes sure different workflow activities are arranged according to pre-defined specifications. Different activities are arranged in the logical groups and different activities are enabled only when they can be used. The BPEL4WS Designer maps structured activities of the BPEL4WS language e.g. Sequence, Process, Flow as Graphical User Interface containers such as a Panel. This mapping was natural choice as structured activities of BPEL4WS can wrap simple activities like invoke, receive or reply and similarly GUI containers can contain simple widgets like buttons, text boxes, labels. The main advantage of this mapping is the enhanced user interface as when the container is dragged or re-positioned visually then all of its inner components are re-positioned relatively, which can be likened to re-arranging of a sequence activity along with all its sub activities within the BPEL4WS process.

Web Service Registry: The Editor provides very basic Web Services registry to arrange Web Services for later use. When a user adds a new Web Service to the registry the editor creates corresponding `<partnerLinkTypes>`, `<partnerLinks>` and `<partnerLink>` elements for later use in the `<invoke>`, `<reply>` or `<receive>` activities. The Web Services registry hides the inner details and complexities of BPEL4WS script and WSDL extensions (required by the BPEL4WS specification) from the user and the designer can concentrate more on the business logic rather than wrapping the partner services in various elements. The Web Service registry can also be used as pool of semantically

equivalent but geographically dispersed Web Services.

XML Schema Registry: When any Web Service is added in the Web Services registry, the editor parses the Web Service and extract data types declared in the `<wsdl:type>` and `<wsdl:message>` elements and build the registry of various data types. XML data in the registry is namespace aware, however, the user doesn't need to address the namespace issues and conflicts. At design time the user only drags the required data type and editor creates the respective variable with correct structure in the BPEL4WS script.

QoS Constraint: The GRIDCC editor provides a pallet grouping of various QoS constraints. The BPEL4WS specification does not define the quality issues related to overall workflow nor individual Web Services. QoS constraints can be coupled with different BPEL4WS activities particularly `<invoke>` activities. QoS constraints for the workflow are specified in the separate file rather than embedding them in the BPEL4WS script.

Workflow Monitoring: Different workflow engines describe the state of a workflow in XML but not all. If a workflow state is available in XML format then it can be exposed and can be queried by the client application. It is also possible to retrieve a workflow snapshot at run time in XML; which can be transformed using XSLT into user acceptable formats. However, it must be stated that extracting an XML snapshot of the current state of the executing workflow requires some understanding of the underlying workflow engine. Consequently, it is easier to create a workflow monitoring Web Service plugged in the workflow script that can be used by workflow clients to monitor or track the execution of a workflow in a portable way. Development of a custom monitoring service can be especially important when status of a workflow and different activities is crucial to decide the execution path of workflow. If workflow monitoring can be integrated with existing management capabilities, it will help to give a more

complete picture of workflow and control on the execution progress at any given moment.

5 QoS aware workflow optimisation

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 is paramount given the inherent shared nature of the Grid. This is compounded through the limited capability of certain high demand services. Furthermore, QoS is essential to deal with the requirements for real time interactions, such as guaranteeing the streaming and storage of data from an instrument. In this environment an aggressive best-effort usage of services will usually satisfy clients on the first come first serve basis regardless of the QoS requirements. This effect can be alleviated through the client specifying loose (soft) and strict (hard) QoS requirements.

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. 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).

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. For example, the reservation of a given portion of a resource such as RAM, disk space, network bandwidth or instrument. The reservation service provider is responsible for keeping infor-

mation about resource availability over time, for ensuring that no resource is over-comitted and for supporting resource-specific locking mechanisms.

QoS provisioning of our work relies on both strict and loose QoS guarantees. These QoS requirements can be made either by a user through the client interface as discussed in Section 4, or as part of the workflow manipulation process. 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 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.

A user submits a BPEL4WS document, which is likely to contain a number of service invocations — referred to as a task. A separate document is used to describe the QoS requirements placed on the BPEL4WS document. Although a number of languages exist for describing QoS requirements, none exist which is suitable for describing QoS requirements with a workflow description. Rather than adding the QoS requirements within the BPEL4WS document it was decided that it would be cleaner to have a second document containing the QoS requirements. This would allow us to use existing BPEL4WS tooling to develop and execute workflows. The use of a second document which references the workflow document was also seen as an advantage as it would make migration to other workflow languages easier.

As such a simple QoS language has been defined which uses XPath[1] references into the BPEL4WS document to tag activities, both basic and structured activities as shown in Figure 4. These XPath tags are then annotated with the QoS requirement such as time to execute, disc space required, memory required.

QoS requirements fall into three categories according to the range of activ-

ities in BPEL4WS models that it specifies: global, single invoke and multiple invoke.

A **global requirement** is a QoS element that specifies single global QoS requirements. A simple example is given in Figure 5 (for simplicity, all the unnecessary technical details are omitted).

In the above example, there is a single *XPathReference* pointing to the entire process of the BPEL4WS document. Thus everything within this process must match these QoS requirements. In this case the overall time should be less than 100 seconds, all CPU's should be 2Ghz and all resources should be fully reliable.

Single invoke activity requirement is a QoS element that specifies requirement on a particular BPEL4WS invoke activity. Only that invoke activity needs to satisfy the QoS requirement specified as shown in Figure 7.

In this case there is a single *XPathReference* pointing to a single invoke element of the BPEL4WS document. Thus everything within this invoke must match these QoS requirements.

Multiple invoke activities requirement is a QoS element that specifies requirement over a set of invoke activities – i.e. all must satisfy (jointly) the QoS requirement. In a single QoS constraint, several *XPathReferences* pointing to different invoke activities in a BPEL4WS model are defined. See Figure 8 Where the time for both invoke activities must be less than 100 seconds.

It should be noted that multiple QoS elements may exist within the same QoS document and that these elements may have overlapping effects on the BPEL4WS document. For example there may be a requirement for the entire workflow to complete within an hour while several of the individual tasks may need to be completed in less than five minutes each.

5.1 QoS Components

The Planner is responsible for ensuring QoS adherence. This is achieved through the use of a number of stages – see Figure 6. Namely *constraint resolver, basic resolver, performance repository and QoS reserver*. These components are chained by the QoS and BPEL4WS documents that are passed through. We explain each of them in detail in the following sub-sections.

Performance Repository (PR) is central to being able to perform any quality of service decisions. Information is held within the PR as to how different entities within the Grid behave. This may be information about how reliable a service is or how long it is expected to take to execute. Information is retrieved from the PR in order to determine if a workflow can be executed within the requested QoS requirements and if so which are the most appropriate resources to select.

Constraint Resolver (CR) is one example of how the Planner may choose the best set of service to use. In general this will come as the first component within 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 in the form of linear equations which can be solved by using Mixed Integer Linear Programming. Information from the Performance Repository is used to help solve these constraint equations.

Basic Resolver (BR) takes 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 BPEL4WS 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.

QoS Reserver inspects incoming QoS documents looking for requests for making reservations with known resources. The Agreement Service is then 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 can't 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.

5.2 Observer

The Observer receives a completed copy of the QoS document, and the BPEL4WS document, at the same time that the Workflow Engine receives the BPEL4WS 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 to the workflow engine, 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. As each workflow engine implements status calls in a different way not only is this approach difficult to implement but also unclear as to the performance hit that will be incurred. The ActiveBPEL editor seems to just expose its internal state and as such calls seem to have little overhead. However, this call should only be made as needed and is an area of open research.

Currently we are supporting the dynamic changing of endpoints within the BPEL4WS documents. All endpoints within the BPEL4WS 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 BPEL4WS workflow during execution.

6 Performance Repository

The Performance Repository (PR) is the central source of information about how services within the Grid behave. This information includes: reliability, availability, accessability and expected sojorn time. The PR provides two main functions, those of gathering and analysing information about other services and that of providing responses to queries about these services.

The PR is implemented as a database and contains two main types of data: modeling and collected. Model information provides mathematically modeled descriptions on how a service operates. This is produced through off-line analysis of the service. Alternatively a service may present information at regular intervals to the PR. This collected information can be data-mined in order to obtain estimates for the service. We seek to extend the PR such that it may derive models from collected information as well as use seasonality to help predict more accurately future state. The intention is to provide a service comparable with the Network Weather Service[2] for services within the Grid. There are two customers for the PR's information, the individual user and the WfMS planner.

Requests for information from the PR may not be handled by the PR in isolation. For example a query of "What 3Ghz Pentium computers are more than 80% reliable?" would first make use of the standard resource selection service to determine which resources are 3Ghz Pentium processors and then

check this candidate set against their reliability. However, this only holds for static information. Queries about current workload and reservations held is not within the remit of the PR.

7 Reservation Services

The reservation service, termed "Agreement Service" (AS) in the GRIDCC architecture, is implementing data types and operations similar to those defined in early drafts of Open Grid Forum's WS-Agreement specification. Essentially, the AS is a proxy to implement a signaling protocol for establishing Service Level Agreements, which in our context refers to contracts for reservations of storage space and instrument elements. By "signaling" we refer to the fact that the reservation information (for instance start/end time) is not visible to the protocol, but rather carried in the message payload – the *Service Description Terms*, in AS terminology. The fact that this protocol and the respective proxy service is resource-agnostic (also because the resource endpoint is expected to be readily provided by the agreement consumer) allows for significant flexibility with regard to the type of resource used. A schematic representation of the different components in such a system is illustrated in Figure 9. The AS is standing between the agreement consumer and the *Reservation and Allocation Service Provider* (RASP). The agreement consumer is the entity requesting the reservation, and may or may not be the same entity with the reservation consumer. The RASP is a gateway to the resource, for reservation purposes. What the AS does, is that it receives an *agreement offer* complying to a specific template, parses the terms, interprets them if required (typically through corresponding external processors) and then contacts the RASP to request the reservation on behalf of the agreement consumer.

In a distributed system like the Grid, multiple Agreement Services might

co-exist, for different administrative domains or geographical regions. For this reason, monitoring the resource state and keeping the lock information on the AS itself would be wrong. Therefore, the AS is only an intermediary for abstracting reservation requests for different resources; It is up to the resource itself to honor the contract and ensure the availability of the reserved resources.

7.1 Advance reservations for overall performance enhancement

Advance reservations can affect the overall performance of a workflow in two different ways, depending on whether it is computing facilities, network resources, storage space, or instruments being reserved. Here, we do not discuss the case of reserving data in digital libraries or other repositories.

The performance change is directly related to the shared (or not, thereof) nature of resources to be reserved. If a resource can be used in a shared manner, as is the case for instance with computing facilities and network links, advance reservation can boost workflow execution times when demanding processing or transfer of information is involved. In the former case, for instance, the user locks a slice of the computing facilities for herself, so the processing tasks submitted will finish as soon as they are expected to. If, however, there is no reservation in place, the user will either have to compromise for slower performance, to wait until the resources are available to her, or to re-route the task thus introducing another discovery-matchmaking-submission cycle. The case is similar for network resources.

When it comes to non-sharable resources, the workflow performance may suffer from the need to wait until the resources are made available from others, overall. For instance, if a workflow must use a specific instrument to retrieve data for the region it is installed in, there is no option other than to wait for

the instrument to become available for use. Should this requirement be known well in advance, the reservation of the resource would allow its immediate use at a pre-specified time, thus removing any queuing delays.

In both cases, the usefulness of reserving resources in advance is directly related to the specific application (workflow) taken into consideration. An application for the batch-processing of historic data can typically accept delays introduced from shared use of resources. For real-time applications which demand immediate and efficient access to resources, the ability to reserve resources in advance may be of critical importance, as they may have to block and suffer significant delays should they not have reserved them when needed.

8 Evaluation and Performance

In this section we provide an evaluation of the real-time services used within the GRIDCC project. Due to the nature of this work being based on bringing workflow support to Grid middleware and for the support of Instruments within the Grid it is difficult to give a quantitative analysis of this work. It is difficult to evaluate the benefit of providing workflow support within the Grid. Instead we focus here on the quantitative features of the system such as the effect on performance of using reservations, the effect of using soft guarantees and the overheads incurred from using this approach.

Figure 10 outlines the test architecture and the locations of where timings are taken from.

8.1 Reservation Jobs

In the current implementation, reserving storage is always dependent on the resource being used, with regard to the completion time of the operation. The only thing that changes from the AS perspective is the amount of space to be

reserved, which is passed on to the storage resource. As such, efficiency of the AS for storage reservations is always the same. It is interesting, however, to see how the AS scales with instrument reservations, where the message payload increases linearly as the number of partitions to be reserved increase (Figure 11). We conducted a number of experiments over a local network, so that networking delays do not affect our measurements. The actors involved in the experiments were a client to the AS, running on the same workstation as the AS. The remote instrument is simulated by a service running on a low-end computer on the same Local Area Network, connected at 100 Mbps. We run a series of 10 experiments with varying number of instrument partitions to be reserved and came up with their averages. As shown in Figures 12 and 13, the AS scales linearly as the message payload size increases.

This is not the case when reserving instruments by quoting their type or functional and non-functional characteristics. In this case, the message size changes insignificantly, and it is the RASP and resource implementation that affects performance for the biggest part. Similarly, when reserving CPUs, the AS performance depends largely on whether reservations take place by naming the exact resources to be reserved, or by grouping them according to their properties and reserving them in this way. Computing facilities reservation has not been implemented in our work yet, and is currently work in progress.

8.2 Soft Guarantee Jobs

– In an unloaded Grid if we submit a job with soft QoS requirements we can use this for selecting resources based on the soft QoS requirement and show more jobs are executed within the QoS requirements than when no QoS requirements are present.

Figure 14 shows the results of QoS aware workflow based on the test of ten

grid resources. In Figure 14, x-axis stands for QoS time(actual runtime), y-axis stands for proportion of successfully completed jobs. 30 Grid resources are used for setting up the experiment including IE, SE and CE resources. Workflows that use these resources with/without QoS are tested. The experiment result indicates that when our QoS component is applied in the WfMS, the resources can be better explored. At one extreme end, if there is no QoS component in WfMS, only very limited number (around 20%) of resources are used by end users, which is straight forward, since these resources have to be selected in advance manually. In contrast, at the other extreme end, the discovery of grid resources stops increase no matter how complex QoS requirements is designed. This is understandable, sine some of the resources (especially for the new ones) might never be selected by QoS. To improve this, particular utility functions have to be adopted in.

9 Conclusion and future work

In this paper we have presented the real-time services used within the GRIDCC project to allow for integration of existing Grid technology with that of instruments. These include the end-to-end workflow pipeline which takes a users design and implements it within the Grid, reservation services and performance repository. 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 editor away from the BPEL4WS / QoS languages and make them more accessible to the scientist. We are investigating other techniques which will allow us to dynamically change the execution of the BPEL4WS workflow once deployed to the engine. Thus allowing for real time adaption of the workflow in light of the changing state of the Grid.

Time	Name	Description
$t_7 - t_0$	User Time	Execution time as seen by the user
$t_{1b} - t_{1a}$	Performance Query	Time to query the Performance Repository
$t_{6b} - t_{6c}$	Agreement Query	Time to query the Agreement Service
$t_{1c} - t_1$	Planner time	Time to plan the execution
$t_{6a} - t_{1c}$	User Workflow	Time taken to execute workflow as seen by the Planner
$t_{5a} - t_{2a}$	BPEL Time	Time the BPEL engine takes to execute workflow
$t_5 - t_2$	User Service time	Time (as seen by the BPEL engine) to call a service
$t_4 - t_3$	Service time	Time to execute a service

Table 1: Meaning of timings in Architecture

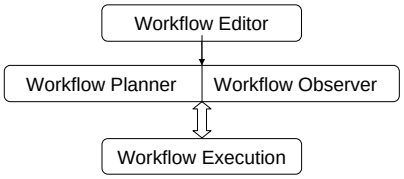

Figure 1: An end-to-end workflow pipeline.

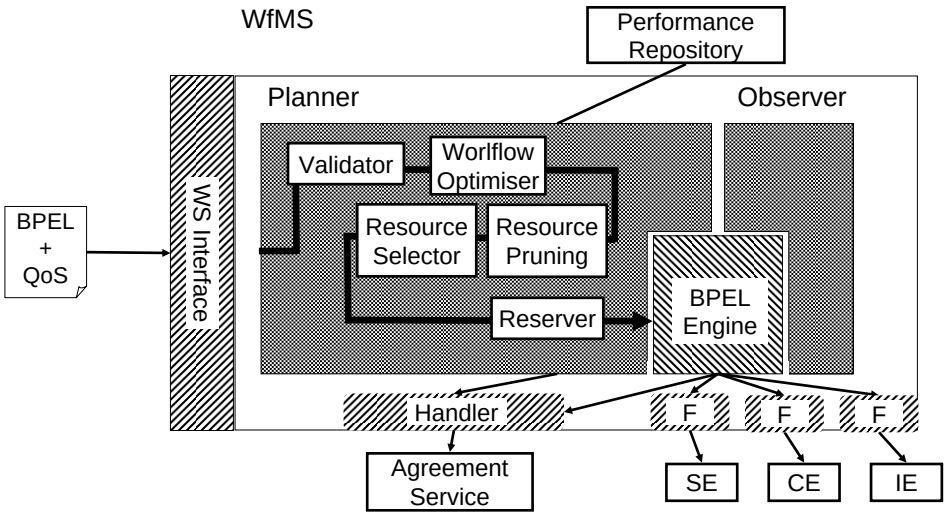

Figure 2: The architecture of the WfMS.

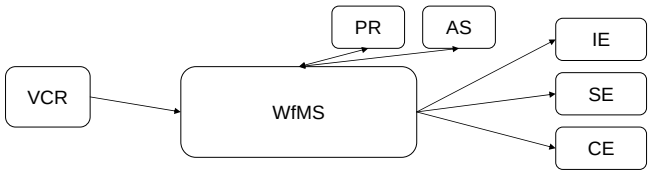

Figure 3: The GRIDCC Architecture.

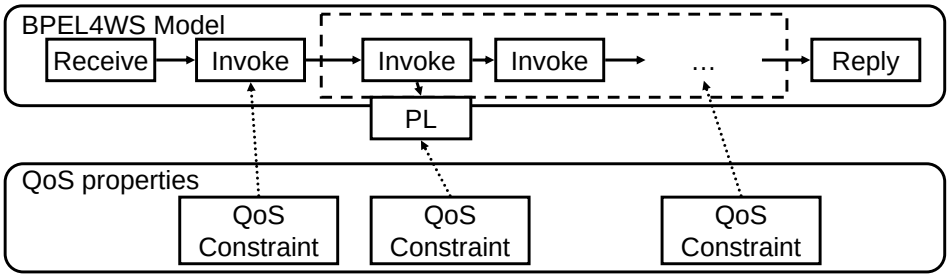

Figure 4: Connecting QoS Document and BPEL4WSModel.

```
<?xml,version="1.0",encoding="UTF-8"?> <QoSRequirements>
    <QoSConstraint>
        <XPathReference>process</XPathReference>
        <Resources>
            <CPUSpeed>2000000</CPUSpeed>
        </Resources>
        <MaxDurationTime>100</MaxDurationTime>
        <Reliability>100</Reliability>
    </QoSConstraint>
</QoSRequirements>
```

Figure 5: A global requirement.

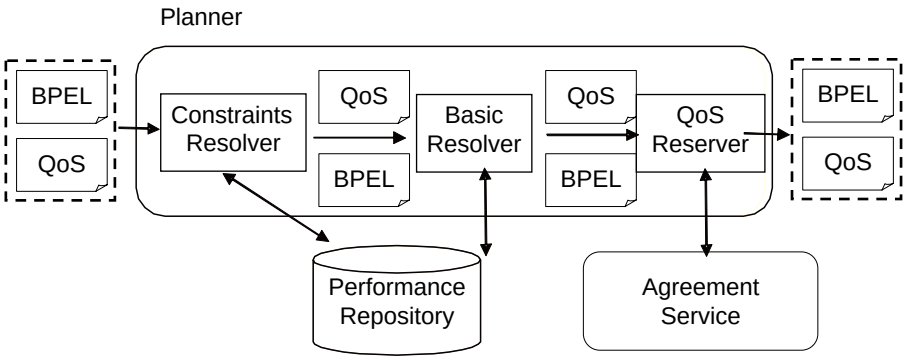

Figure 6: QoS Components

```
<?xml\,version="1.0"\,encoding="UTF-8"?> <QoSRequirements>
    <QoSConstraint>
        <XPathReference>
          /process/sequence[1]/invoke[1]
        </XPathReference>
        ...
    </QoSConstraint>
</QoSRequirements>
```

Figure 7: A single invoke activity requirement.

```
<?xml\,version="1.0"\,encoding="UTF-8"?> <QoSRequirements>
    <QoSConstraint>
        <XPathReference>
          /process/sequence[1]/invoke[1]
        </XPathReference>
        <XPathReference>
          /process/sequence[2]/invoke[2]
        </XPathReference>
        <MaxDurationTime>100</MaxDurationTime>
    </QoSConstraint>
</QoSRequirements>
```

Figure 8: Multiple invoke activity requirements.

Figure 9: Agreement Service interactions

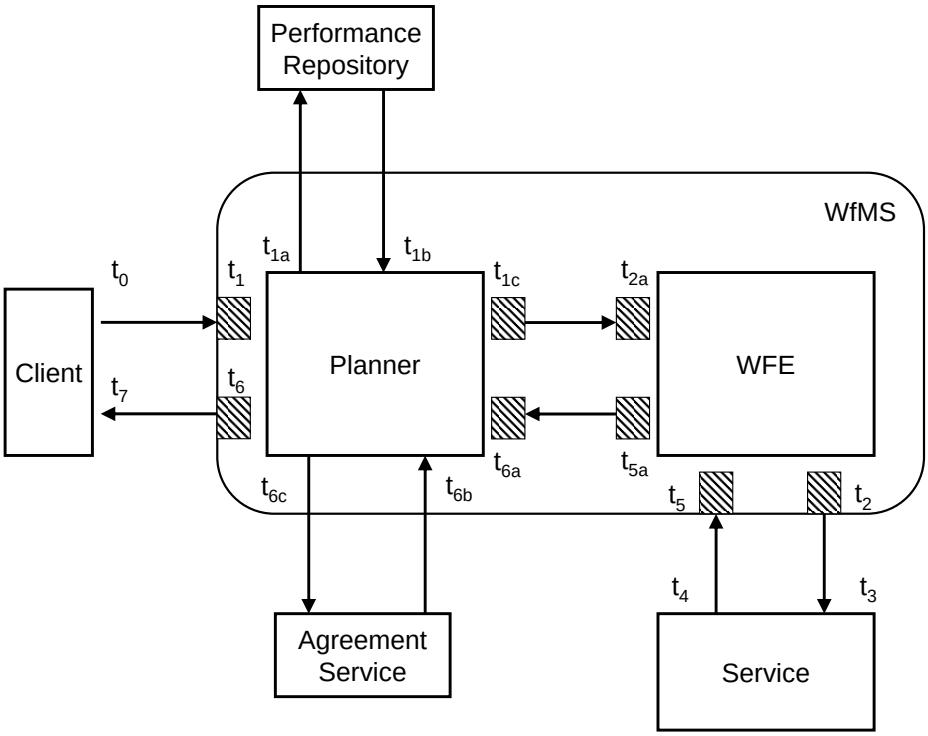

Figure 10: Locations of timings within architecture

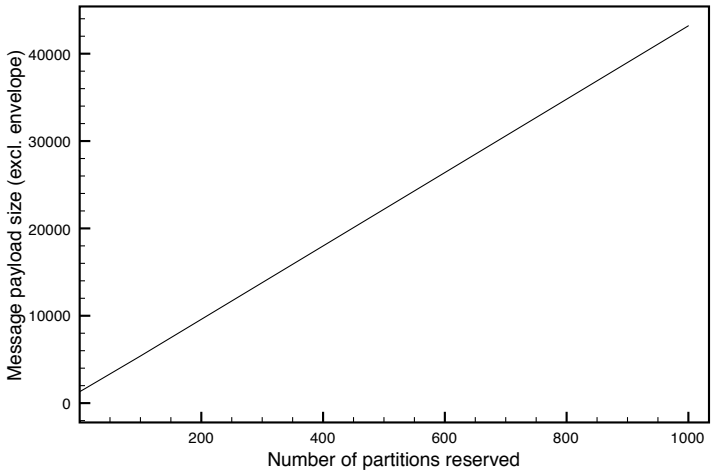

Figure 11: Instrument reservation invocation messages size, in Bytes

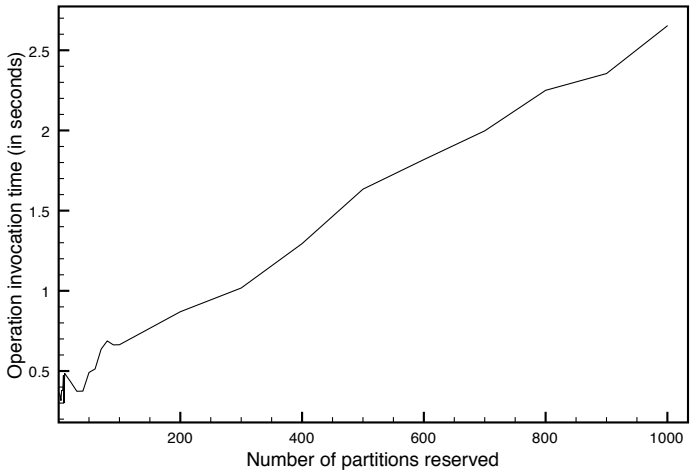

Figure 12: Instrument reservation completion time (linear scale)

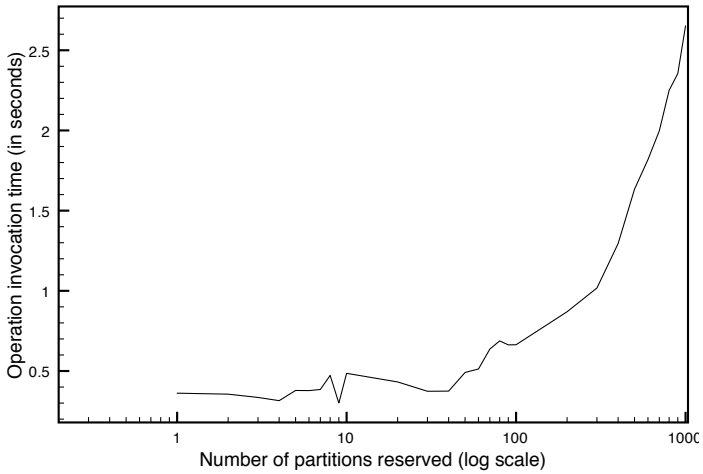

Figure 13: Instrument reservation completion time (logarithmic scale)

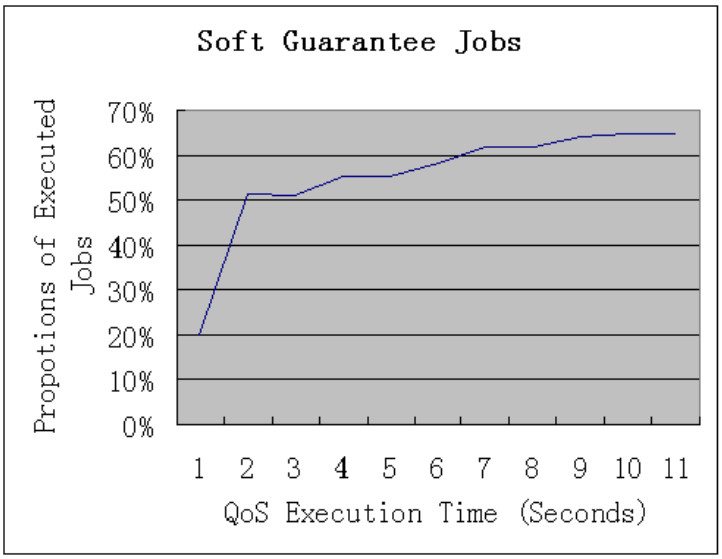

Figure 14: Soft QoS Guaranteed Jobs

References

[1] Xml path language (xpath) version 1.0. `http://www.w3.org/TR/xpath`, 1999.

[2] Network weather service. `http://nws.cs.ucsb.edu/ewiki/`, 2005.

[3] A Afzal and J Darlington and AS McGough. Stochastic Workflow Scheduling with QoS Guarantees in Grid Computing Environments. pages 185–194, Changsha, China, October 2006.

[4] Active Endpoints. ActiveBPEL Engine (2.0). `http://www.activebpel.org`.

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

[6] 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, 12–15 May 2003.

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

[8] 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 956–963, Reading, UK, April 2006.

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

[10] E. Deelman, J. Blythe, Y. Gil C. Kesselman, G. Mehta andK. Vahi, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing*, 1(1):9–23, 2003.

[11] Don Box and Erik Christensen and Francisco Curbera and Donald Ferguson and Jeffrey Frey and Marc Hadley and Chris Kaler and David Langworthy and Frank Leymann and Brad Lovering and Steve Lucco and Steve Millet and Nirmal Mukhi and Mark Nottingham and David Orchard and John Shewchuk and Eugne Sindambiwe and Tony Storey and Sanjiva Weerawarana and Steve Winkler. Web services Addressing (WS-Addressing), August 2004.

[12] E Christensen and F Curbera and G Meredith and S Weerawarana. Web services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, March 2001.

[13] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto Jr, and H. L. Truong. ASKALON: a tool set for cluster and Grid computing. *Concurrency and Computation: Practice and Experience*, 17(2-4):143–169, 2005.

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

[15] Gary Grossman and Emmy Huang. ActionScript 3.0 overview. `http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html`, June 2006.

[16] L F Cabrera and G Copeland and M Feingold and R W Freund and H T Freund and J Johnson and S Joyce and C Kaler and J Klein and D

Langworthy and M Little and A Nadalin and E Newcomer and D Orchard and I Robinson and J Shewchuk and Tony Storey. Web Services Coordination (WS-Coordination) 1.0. ftp://www6.software.ibm.com/software/developer/library/WS-Coordination.pdf, August 2005.

[17] L Huang and DW Walker and Y Huang and OF Rana. Dynamic Web Service Selection for Workflow Optimisation. Proceedings of the UK e-Science All Hands Meeting, September 2005.

[18] 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 894–900. IOP Publishing Ltd, Bristol, UK, Sep. 2003.

[19] A.Stephen McGough, Jeremy Cohen, John Darlington, Eleftheria Katsiri, William Lee, Sofia Panagiotidi, and Yash Patel. An End-to-end Workflow Pipeline for Large-scale Grid Computing. *Journal of Grid Computing*, pages 1–23, February 2006.

[20] S. McGough, L. Young, A. Afzal, S. Newhouse, and J. Darlington. Performance Architecture within ICENI. In *UK e-Science All Hands Meeting, Nottingham, UK*, pages 906–911. IOP Publishing Ltd, Bristol, UK, Sep. 2004.

[21] 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 894–900. IOP Publishing Ltd, Bristol, UK, Sep. 2004.

[22] Duane Nickull and Francis McCabe. SOA Reference Model. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm.

[23] Ruslan Bilorusets and Don Box and Luis Felipe Cabrera and Doug Davis and Donald Ferguson and Christopher Ferris and Tom Freund and Mary Ann Hondo and John Ibbotson and Lei Jin and Chris Kaler and David Langworthy and Amelia Lewis and Rodney Limprecht and Steve Lucco and Don Mullen and Anthony Nadalin and Mark Nottingham and David Orchard and Jamie Roots and Shivajee Samdarshi and John Shewchuk. Web Services Reliable Messaging Protocol (WS-ReliableMessaging). `ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200502.pdf`, February 2005.

[24] 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, December 2005. IEEE Computer Society.

[25] T Andrews and F Curbera and H Dholakia and Y Goland and J Klein and F Leymann and K Liu and D Roller and D Smith and S Thatte and I Trickovic and S Weerawarana. Business Process Execution Language for Web services version 1.1, (BPEL4WS). `http://www6.software.ibm.com/software/developer/library/ws-bpel.pdf`, May 2003.

[26] 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.

[27] I. Taylor, M. Shields, and I. Wang. Resource Management for the Triana Peer-to-Peer Services. In J. Nabrzyski, J. M. Schopf, and J. Węglarz, editors, *Grid Resource Management - State of the Art and Future Trends*, pages 451–462. Kluwer Academic Publishers, 2004.

[28] W Cox and F Cabrera and G Copeland and T Freund and J Klein and T Storey and S Thatte. Web Services Transaction (WS-Transaction) 1.0. `http://dev2dev.bea.com/pub/a/2004/01/ws-transaction.html`, January 2004.

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

[30] 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.

[31] Y Patel and AS McGough and J Darlington. QoS Support for Workflows in a Volatile Grid. In *Procedings of the 7th IEEE/ACM International Conference on Grid Computing*, Barcelona, Spain, September 2006.

[32] L. Young and J. Darlington. Scheduling Componentized Applications on a Computational Grid. MPhil/PhD Transfer Report, Imperial College London, University of London, 2004.

[33] L. Young, S. McGough, S. Newhouse, and J. Darlington. Scheduling Architecture and Algorithms within the ICENI Grid Middleware. In *UK e-Science All Hands Meeting, Nottingham, UK*, pages 5–12. IOP Publishing Ltd, Bristol, UK, Sep. 2003.

[34] 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.

[35] 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.