Enabling Scientists through Workflow and Quality of Service

A S McGough[1], A Akram[1], D Colling[2], L Guo[1], C Kotsokalis[3], M Krznaric[1], P Kyberd[4], and J Martyniak[2]

[1] London e-Science Centre, Department of Computing, Imperial College London, London SW7 2BZ, UK
{aakram1,liguo,marko,asm}@doc.ic.ac.uk

[2] High Energy Physics Group, Department of Physics, Imperial College London London SW7 2BZ, UK
{d.colling,janusz.martyniak}@imperial.ac.uk

[3] Greek Research and Technology Network, Greece and National Technical University of Athens, Greece
ckotso@grnet.gr

[4] Brunel University, UK
Paul.Kyberd@brunel.ac.uk

Summary. There is a strong desire within scientific communities to Grid-enable their experiments. This is fueled by the advantages of having remote (collaborative) access to instruments, computational resources and storage. In order to make the scientists experience as rewarding as possible two technologies need to be adopted into the Grid paradigm. Those of workflow, to allow the whole scientific process to be automated, and Quality of Service (QoS) to ensure that this automation meets the scientists expectations. In this paper we present an end-to-end workflow pipeline which takes a users design and automates the processes of workflow design, resource selections and reservation through to enacting the workflow on the Grid. Thus removing much of the complexity inherent within this process.

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. The use of services such as computational resources and computer enabled instruments has been seen as an advantage to this process. Many scientists now seek to automate the process of bringing these tasks together through the use of workflows. Where a workflow is a set of tasks, along with a definition of how these tasks should interact.

The Grid [11] is an enabling paradigm for bringing these different services into a collaborative environment. The use of Web Services [20] is emerging as a the de-facto communication mechanism, with most Grid projects now supporting them to some degree. Worflow languages such as BPEL [18], WS-Choreography [7] and YAWL [21] 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. 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.

In this paper we show how to construct an end-to-end workflow pipeline, based on the ideas from McGough et al [15]. This is exemplified through the Grid architecture used within the GRIDCC [6] project. The workflow pipeline is illustrated in Figure 1. The scientist will develop the workflow within the Workflow Editor. Although a workflow is developed, the view presented to

the scientist is an abstraction away from the workflow language. 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 requirements on how the workflow should execute – a required level of *Quality of Service* (QoS). 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 2.

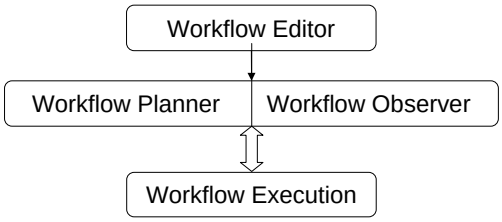

Fig. 1. An end-to-end workflow pipeline.

Once the workflow has been specified by the user the process of selecting the most appropriate set of resources needs to be performed. 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. A full breakdown of the Workflow Planner can be found in Section 3. The Workflow observer is designed to deal with cases where the workflow deviates from the plan defined within the workflow planner. The observer monitors the progress of the workflow and if the tasks deviate from the plan will cause the workflow to change in order to increase the chance of the workflow completing successfully. This is an area of active research.

1.1 The Grid architecture

Here we discuss the architecture of the Grid used within this paper. We are adopting the design used within the GRIDCC [6] project. Figure 2 shows the overall architecture of this 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 contains the Workflow Planner and the Workflow Observer along with the Workflow Engine, in the case of GRIDCC this is the ActiveBPEL [10] 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 resources and models derived from testbed experiments.

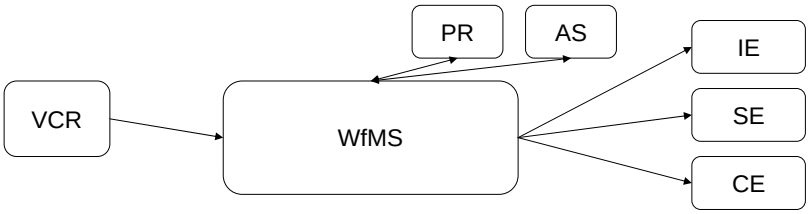

Fig. 2. The GRIDCC Architecture.

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 requested QoS constraints. The WfMS may choose to manipulate the workflow in order to achieve this level of QoS. This may include making reservations for tasks in the workflow and or changing the structure of the workflow [15]. The Workflow Engine is then responsible for submitting the tasks of the workflow to the different types of Grid resources with the Observer monitoring its progress.

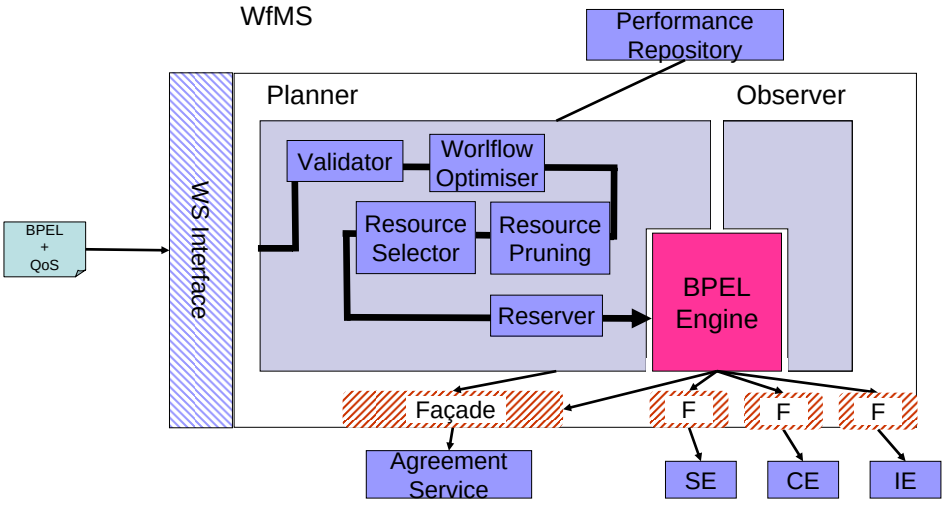

Fig. 3. The architecture of the WfMS.

Figure 3 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 and converted from XML into an object representation. The workflow and QoS document can then pass through various stages (outlined further in Section 3) before the BPEL4WS document is submitted to the BPEL4WS engine and the revised QoS submitted to the Observer. In order to allow the BPEL4WS engine to communicate with the existing Grid resources inline facades are used to manipulate the Web Services calls.

2 Workflow editing

Service Oriented Architecture (SOA) [16] encourages software components development for on-demand consumption. SOA principles also influence the business logic of services by encouraging good design, i.e. by promoting modular and dispersed components that can be separately developed and maintained. 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 [9]). Services exposed as Web services (such as those wrapping and exposing legacy codes) can be integrated into complex services that may span multiple domains and organizations.

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 by acting as a broker or 'middle-man'. The GRIDCC project is aiming to provide a Web based workflow editor based on Business Process Execution Language for Web Services (BPEL4WS) [18]. BPEL4WS is generally regarded as the de-facto standard for composing and orchestrating workflows. 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 [17], WS-Addressing [8], WS-Coordination [13] and WS-Transactions [19] (BPEL4WS however, is not strictly limited to only these technologies).

2.1 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 exposing the internal details of the process with the objective to specify only the externally observable aspects of process behaviour without revealing their internal implementation. An 'Executable Process' extends abstract process to allow specification of the exact details of a process workflow.

The BPEL4WS specification defines various building blocks, known as 'activities'. These activities can be used to specify an executable process's business logic to any level of complexity. The BPEL4WS 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* like 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.

On the other hand, reusability of primitive and structured activities in BPEL4WS is limited. For example, it is not possible to re-execute an activity that is defined earlier in the script by referring to it later. In this scenario, the activity needs to be re-defined where necessary. Moreover, it is not possible to inject, say, Java code directly into a workflow script. One would need to wrap up a Java code as a Web service with the associated performance impact. Alternatively, some technology like Web Services Invocation Framework (WSIF) [1] may be used to invoke local Java objects using the SOAP [14] protocol. This makes extending a BPEL4WS for, say, QoS more complex.

Furthermore, the BPEL4WS specification does not provide any direct mechanism to support management and monitoring of a workflow script. Also, BPEL4WS specification does not address Web service security issues, such as passing of credentials through a BPEL4WS engine. There WfMS uses "deanonymisation" technique in which users identity is passed in the message header. The the outgoing message is intercepted to pick user credential based on her identity and to perform a secure call to an external service.

Generating a workflow document is therefore a process of bringing these 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 Active BPEL Editor and Oracle BPEL4WS Process Manager. However, these are developed for computer scientists who wish to develop workflows and an extensive knowledge of BPEL, WSDL, etc is required.

2.2 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, and a rich set of keyboard shortcuts, and provision to save partial workflows and integrate them as building blocks for complicated and sophisticated workflows. The purpose of 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. 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 editors before any use. Installation of workflow editor means access to local file system as admin, which may not be available. This JSR 168 [4] 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: GRIDCC 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. The Workflow Editor is based on the ActionScript 3.0 [12] and MXML [5]; 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 in following ways:

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 of the partner Web Services are called through `<invoke>` activity. The `<invoke>` activity specify 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 specify `<input>` and `<output>` variables to store the values of outgoing and incoming messages. If these variables are not existing 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 Designer creates structured activities as the container and other activities can be dropped in the container to visually assist the modular designing of the workflow.

Web Service Registry: Editor provides very basic Web services registry to arrange Web services for later use. When a user add a new Web Service to the registry, designer creates corresponding `<partnerLinkTypes>`, `<partnerLinks>` and `<partnerLink>` elements for later use in the `<invoke>`, `<reply>` or `<receive>` activities. Web services registry hides inner details and complexities of BPEL4WS script and WSDL extensions (required by BPEL4WS specification) from the user and designer can concentrate more on the business logic rather than wrapping the partner services in various elements. 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 parse 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 and 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: GRIDCC editor provides a pallet grouping various QoS constraints. BPEL4WS specification doesn't 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.

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 easy 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.

3 Workflow with Quality of Service Requirements

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 highly demanded 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 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 of keeping information about resource availability over time, of ensuring that a 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 2, or as part of 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 an activity. A separate document is used to describe the QoS requirements placed onto this 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.

As such a simple QoS language has been defined which uses XPath[2] 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.

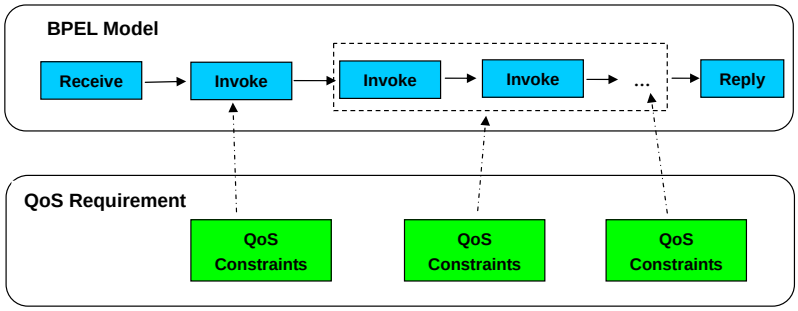

Fig. 4. Connecting QoS Document and BPEL4WS Model.

QoS requirements fall into three categories according to the range of activities 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

```
<?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>
```

Fig. 5. A global requirement.

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 6

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

Fig. 6. A single invoke activity requirement.

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 7 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.

```
<?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>
```

Fig. 7. Multiple invoke activity requirements.

3.1 QoS Components

The Planner is responsible for ensuring QoS adherence. This is achieved through the use of a number of stages – see Figure 8. 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.

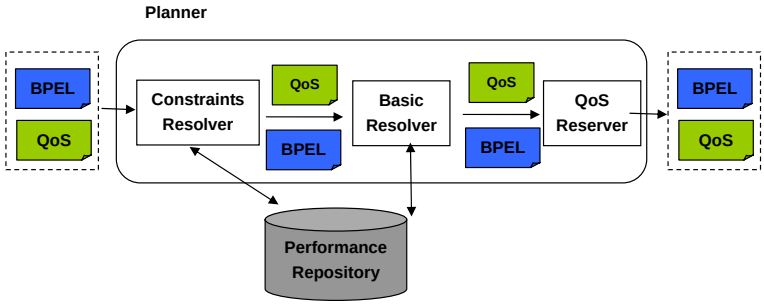

Fig. 8. QoS Components

Performance Repository (PR) is central to being able to perform any quality of service decisions. The Performance Repository will serve data of two types: modeling and dynamic. Model information provides information about how the service performs under different conditions. This may include such information as the reliability of an instrument or the amount of time it takes to bring the instrument on-line. Dynamic information, provided by the Information Service in many Grids, indicates the current status of the resources. Such as load, network bandwith or if the service is currently in use.

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[3]. There are two customers for the PR's information, the individual user and the WfMS planner. 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.

Constraint Resolver is the first component to provide 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.

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.

4 Conclusion and future work

In this paper we have presented an end-to-end workflow pipeline which takes a users design and implements it within the Grid. The workflow is augmented with a QoS document defining the users expectations for the execution. This is developed through a Workflow Editor which abstracts the user away from the technical complexities of dealing with WSDL and BPEL. 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.

We are working to extend the workflow developer and WfMS. We are working with the application scientists from the GRIDCC project to make the workflow developer more tailored to the community. We are also developing the WfMS to allow the Observer to modify the running workflow within the BPEL4WS engine. This will allow us to change the flow of the workflow and the resources used.

References

1. Web service Invocation Framework (WSIF). `http://ws.apache.org/wsif/`.
2. Xml path language (xpath) version 1.0. Technical report, 1999.
3. http://nws.cs.ucsb.edu/ewiki/. Technical report, 2005.
4. Alejandro Abdelnur and Stefan Hepper. Porlet specification (jsr-168). `http://jcp.org/aboutJava/communityprocess/review/jsr168/`.
5. Christophe Coenraets. An overview of MXML: The Flex markup language. `http://www.adobe.com/devnet/flex/articles/paradigm.html`, March 2004.
6. 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.
7. David Burdett and Nickolas Kavantzas. `http://www.w3.org/TR/ws-chor-model/`.

8. 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.
9. 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.
10. Active endpoints. ActiveBPEL Engine (2.0). `http://www.activebpel.org`.
11. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, July 1998.
12. Gary Grossman and Emmy Huang. ActionScript 3.0 overview. `http://www.adobe.com/devnet/actionscript/articles/actionscript3\_overview.html`, June 2006.
13. 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.
14. Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau and Henrik Frystyk Nielsen. Soap version 1.2. `http://www.w3.org/TR/soap12-part1/`.
15. 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.
16. Duane Nickull and Francis McCabe. Soa reference model. `http://www.oasis-open.org/committees/tc\_home.php?wg\_abbrev=soa-rm`.
17. 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.

18. 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.
19. 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.
20. W3C. Web Service. `http://www.w3.org/TR/ws-arch/`.
21. 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.