Workflow Management for Grid Computing – A Grid Workflow Infrastructure

Dieter Cybok

msg systems ag, Munich, Germany
dieter_cybok@msg.de

Abstract: In this paper we propose a Grid Workflow Infrastructure, which serves as the base for specifying and executing collaborative interactive workflows within computational grids. The infrastructure is based on the Open Grid Services Architecture (OGSA) and leverages the concepts of the Business Process Execution Language for Web Services (BPEL4WS).
Using OGSA enables us to exploit advanced Grid features such as factories, lifecycle management and notifications. Leveraging BPEL4WS to a Grid enabled workflow language has the advantage that basic workflow functionalities, which are similar for Grid and Web Services, do not have to be developed again. The result is a state of the art Grid Workflow Infrastructure that was developed within a relatively short period. The main building blocks of the infrastructure are the specification of the Grid Workflow Execution Language (GWEL) notation and the implementation of a Grid workflow execution engine, using Globus Toolkit 3 (GT3) technology, for processing e-Science specific workflows specified in GWEL documents. The workflow engine itself is a high-level Grid Service, hence automatically Grid aware, that can be used within any GT3 environment.

1 Introduction

Many computing and data intensive scientific applications, such as climate modeling, astrophysics, high-energy physics, structural biology and chemistry, require the creation of a collaborative workflow infrastructure as part of their sophisticated problem solving processes. By using the Grid computing approach, such an infrastructure can be provided in order to enable the coordinated use of numerous distributed and heterogeneous resources.

Developing scientific applications for these large-scale collaborative Grid infrastructures is a difficult task. Although elementary Grid Services exist that enable scientific application developers to authenticate, access, manage, and discover remote resources, these services can currently not easily be composed to operate upon a specified workflow. Since many e-scientists lack the necessary low-level expertise to utilize the current generation of Grid toolkits, such as GT3, the development of a Grid Workflow Infrastructure is necessary.

The goal of our Grid Workflow Infrastructure was to develop a system that supports e-Science workflow and collaboration by integrating a workflow engine into any GT3 environment. Such a system could be beneficial for modeling and managing scientific processes of experimental investigation, evidence accumulation and result assimilation. Processes themselves can then be reused, shared, adapted and annotated with interpretations of quality, provenance and security.

Currently, workflow systems for Grid Services are evoking a high degree of interest. Therefore we describe three related projects. However, so far no product that makes fully use of the features of GT3 has been proposed.

2 Related Work

2.1 Grid Service Flow Language

One of the most well known projects that addresses workflow management in an OGSA environment [OGSA] is presented in [GSFL]. In the paper, technologies that address workflow for Web Services and leverage this technology for Grid Services are examined. The project mainly focuses on the definition of the Grid Services Flow Language (GSFL), but also includes a description of an implementation of a workflow engine for use in the now redundant OGSI [OGSI]. However, the project is based on GT version 2.4 and therefore does not consider any WS-Resource Framework [WSRF] features.

2.2 Grid Web Services and Application Factories

In [CCA], the authors propose a distributed software component architecture, XCAT, for Grid computing that is compatible with the Common Component Architecture specification. Furthermore they discuss how the work can be merged with the (then new) OGSI specification. Using this framework, e-scientists can compose an application from a set of distributed components. The result is a set of Web Services that collectively represent the executing application instance.

CCA components have two types of ports. One type of port, which is called a provides port, is identical to a Web Service port. The other type, called a uses port, is an external reference from one component to a provides port on another component that can be bound at runtime. XCAT can also make use of an XML-based messaging system that provides a simple way for components to publish or subscribe to messages.

2.3 Open Collaborative Grid Services Architecture

The Open Collaborative Grid Services Architecture (OCGSA) composes a set of common components that can be customized for individual applications [OCGSA]. The framework enables users to form ad-hoc collaborative groups by interacting over a set of predefined notification topics. It provides appropriate lifetime management for individual groups, offers an advanced discovery mechanism for service instances, and establishes sophisticated security mechanisms at different levels of the application.

Furthermore, the OCGSA framework extends the notion of Grid Services into the collaborative domain, by introducing the concept of a collaborative Grid Service. The collaborative Grid Service supports different levels of security policies, basic community chat, user presence management, and metadata information regarding the collaborators. OCGSA also supports the implementation of transient event archives in a secure infrastructure. The authors claim that the extensible OCGSA framework simplifies the development efforts of a large number of collaborative applications.

3 Workflow Management

Workflow is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal [WRM]. The goal of workflow management is to make sure that the proper activities are executed by the right service at the right time.

In figure 1, the reference model of the Workflow Management Coalition [WRM] shows the architecture of a workflow management system at the level of components.

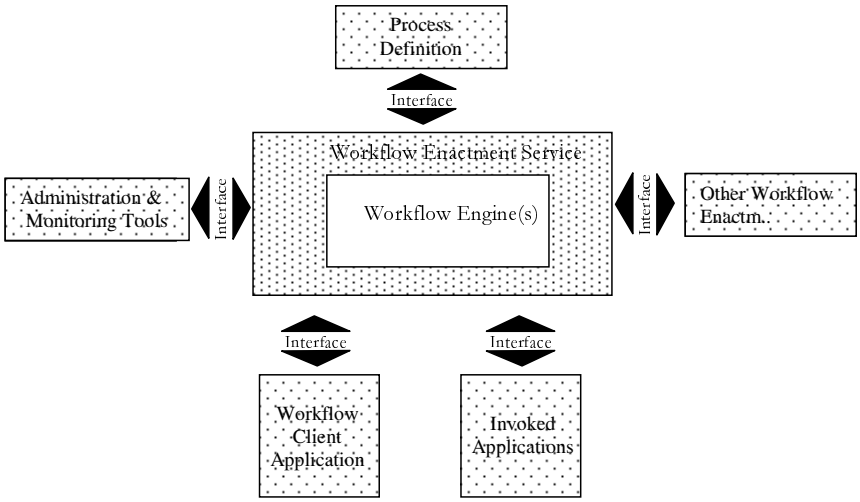

Figure 1: Workflow Reference Model

The workflow enactment service, which is the main component in the model, provides the run-time environment in which process instantiation and activation occurs. It includes one or more workflow engines. A workflow engine is software that provides the run time execution environment for a workflow instance including facilities to handle:

- interpretation of the process definition
- control of process instances, such as creation, activation, suspension and termination
- navigation between process activities, which may involve sequential or parallel operations, deadline scheduling, interpretation of workflow relevant data, etc
- sign-on and sign-off of specific participants
- identification of work items for user attention and an interface to support user interactions
- maintenance of workflow control data and workflow relevant data, passing workflow relevant data to/from applications or users
- an interface to invoke external applications and link any workflow relevant data
- supervisory actions for control, administration and audit purposes

3.1 Workflow Patterns

For defining the control-flow constructs of a workflow language some of the results from workflow research are useful. In [WorkflowPatterns] a set of the most common workflow patterns is proposed. For proposing GWEL, we expand on a small subset of these patterns to show the most relevant constructs.

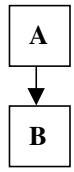

Figure 2

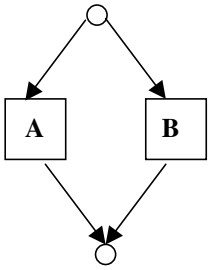

Figure 3

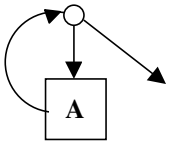

Figure 4

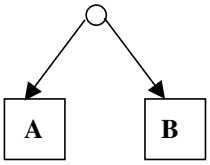

Figure 5

- Sequence (Figure 2): An activity in a workflow process is enabled after the completion of another activity in the same process.
- Parallel Split (Figure 3): A point in the process where a single thread of control splits into multiple threads of control which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order.
- Arbitrary Cycles (Figure 4): A point in a workflow process where one or more activities can be executed repeatedly. This pattern is also referred to as a loop.
- MultiChoice (Figure 5): A point in the process, where, based on a decision or control data, a number of branches (or just one) are chosen and executed as parallel threads. This pattern is also referred to as conditional routing.

Based on these patterns, more sophisticated constructs can be built. However, it is not the subject of this paper to elaborate further on the difference of workflow patterns between e-business and e-Science applications. It is assumed, that is currently sufficient for e-Science implementations to draw on the experience with e-Business workflow patterns. It remains to be investigated, whether e-Science specific workflow patterns will emerge in the future.

3.2 Web Services Workflow Management

To describe the concept of composing Grid Services, Web Services concepts can be reused. For Web Services composition, the terms orchestration and choreography are frequently used. Orchestrationdescribes how Web Services can interact with each other at the message level, including the business logic and execution order of the interactions. Choreographyexpresses the sequence of messages that involve multiple services. It represents the public message exchanges that occur between Web Services, rather than a specific business process that is executed by a single party. The main distinction between Web Services orchestration and choreography is twofold [WSOrchestration]. Orchestration refers to an executable business process that interacts with both internal and external Web Services. For orchestration, the process is always controlled from the perspective of one of the business parties. Choreography is more collaborative. Each service involved in the process describes the part it plays in the interaction.

The orchestration of Web Services poses challenges from both a technical and business perspective [BuildingWS]. On the technical side, the description of services, defined by the rules for sequencing operation invocations and for sending and receiving messages, is one of the most important issues. Furthermore, the problem of composing services into process-based interactions has to be solved. This includes the requirement that some composition bindings must happen at runtime in order to map the technology to business processes. From the business perspective, service integration is a workflow problem, which concerns not only the technical aspects but also introduces dependencies on aspects of business models.

The task of composing Web Services to create efficient Web processes also raises the following issues:

- Managing transactional integrity: A way to manage transactional integrity if something goes wrong has to be provided.
- Compensating transactions: The concept of undoing an action if a process or user cancels it is important.
- Managing exceptions: It has to be taken into account how the system will respond if there is an error or if the service invoked does not respond in a given time.
- Efficiency of a composed Web process: A composed process must be efficient in terms of service time and scalability. The ability to invoke services in an asynchronous manner is vital to achieving reliability, scalability, and adaptability. With asynchronous support, a business process can invoke Web Services concurrently rather than sequentially in order to enhance performance.

For deciding which language to use as a base for GWEL we have considered both, the Business Process Execution Language for Web Services and the Web Service Choreography Interface (WSCI) [WSCI]. Theoretically they are both candidates for a language on which GWEL will be based. In the following section we outline both of them.

3.3 Web Services Workflow Languages

Services composition languages usually build on top of WSDL. They provide and use WSDL services that consist of ports to provide operations. Each operation receives a message (one-way), receives and sends a message (request-response), sends and receives a message (solicit-response), or sends a message (notification). To compose services, a process model is necessary to specify the order in which the operations are executed. A Web services composition language provides the means to specify such a process model. An important difference between WSDL and a Web services composition language is revealed when considering the states of the services involved [Flow]. WSDL is stateless because the language is not aware of states in between operations. A Web services composition language records states for processes. By recording the state it is possible to enable long-lived transactions.

3.3.1 Business Process Execution Language for Web Services

BPEL4WS models the behaviour of Web services in a business process interaction. It was proposed by a consortium including members of IBM, BEA and Microsoft in July 2002.

BPEL4WS is essentially a layer on top of WSDL, with WSDL defining the specific operations allowed and BPEL4WS defining how the operations can be sequenced. A BPEL4WS document leverages WSDL in three ways [WSOrchestration]:

- Every BPEL4WS process is exposed as a Web service using WSDL. The WSDL describes the public entry and exit points for the process.
- WSDL data types are used within a BPEL4WS process to describe the information that passes between requests.
- WSDL might be used to reference external services required by the process.

Since BPEL4WS provides support for both executable and abstract business processes [BPEL] it can be used as an implementation language and to describe the interfaces of business processes.

Executable processes model the actual behaviour of a participant in a business interaction. They specify the execution order between a number of activities that constitute the process, the partners involved in the process, the messages exchanged between partners, and the fault and exception handling specifying the behaviour in cases of errors and exceptions [BPELAnalysis]. In executable processes, externally visible or (i.e. public) aspects of a business process are not separated from its internal workings. Business protocols use process descriptions that specify the message exchange behaviour of each of the parties involved in the protocol, without revealing their internal behaviour. The processes involved in a business protocol are called abstract processes and are not executable. However, in GWEL we just focus on the notion of executable processes.

BPEL4WS documents can be interpreted and executed by an orchestration engine, which is controlled by one of the participating parties. The engine coordinates the various activities in the process, and compensates the system when errors occur. Essentially, executable processes provide the orchestration support while the business protocols focus on the choreography of the services.

Web services workflows implemented as BPEL4WS processes have an instanced life cycle model. As shown in figure 6, a client of these services always interacts with a specific instance of the service (process).

3.3.2 Web Service Choreography Interface

WSCI was proposed by BEA Systems, Intalio, SAP AG, and Sun Microsystems and submitted to the W3C in June 2002 [WSCI]. Similar to BPEL4WS, WSCI is an XML-based language used to describe the flow of messages exchanged by a Web service participating in choreographed interactions with other services. WSCI enables the mapping of services as components for realizing processes. An important aspect of WSCI is that it only describes the observable behaviour between Web services. It does not address the definition

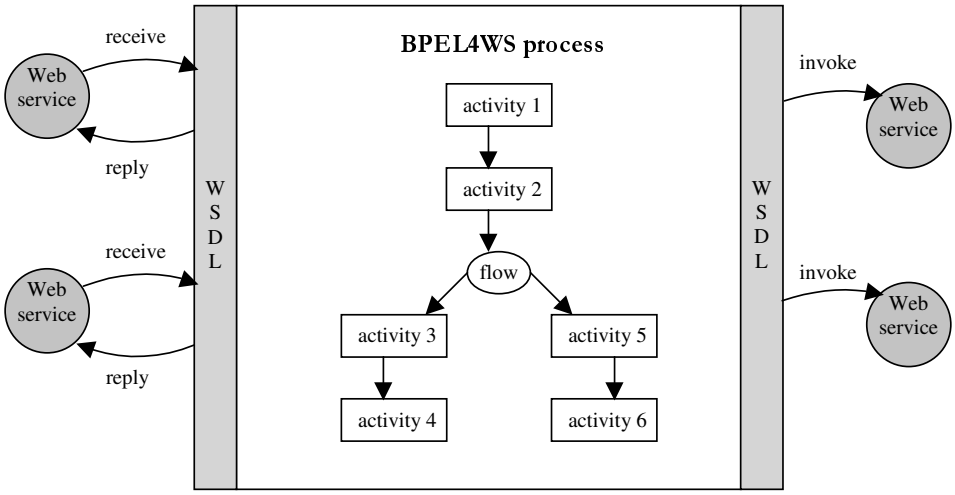

Figure 6: BPEL processmodel

of executable business processes as defined by BPEL4WS. Furthermore, a single WSCI document only describes one Web service's participation in a message exchange. As outlined in figure 7, a WSCI choreography comprises a WSCI document for each partner in the interaction [WSOrchestration]. In contrast to BPEL4WS, in WSCI there is no single controlling process managing the interaction.

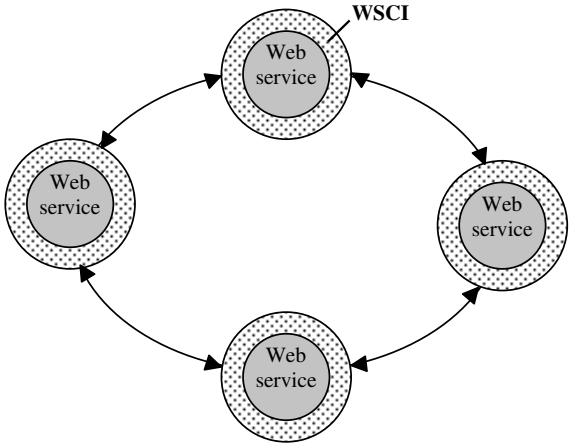

Figure 7: WSCI processmodel

4 A Grid Workflow Infrastructure

4.1 Overview

Based on the definition in section 3, Grid workflow can be defined as an organization of Grid Services into a well-defined flow of operations, and can be thought of as the composition of Grid Services over time to accomplish a specific goal. The composition of Grid Services in space defines how the services are logically connected at any point of time.

Currently, workflow systems for Grid Services are evoking a high degree of interest. However, it is an area that is currently not defined explicitly by the WS-Resource Framework specification. Furthermore, we are not aware of any tool that makes fully use of the features provided in GT3. With the implementation of our Grid Workflow Infrastructure we are contributing to fill this gap.

4.2 Grid Workflow Requirements

As discussed previously, there are many similarities between Grid-based workflows and workflows encountered in Web Services. So far, existing Web Service technologies have not addressed all requirements for workflows in the Grid. However, the newly emerged WS-Resource Framework specification provides a good base for functionality that is necessary for workflow management in Grid computing.

To describe the concept of composing Grid Services, the terms orchestration and choreography, which we defined in the Web Services context, can be reused. For the Grid Workflow Infrastructure, we are using the orchestration approach. The reasons for this decision are as follows:

- In the orchestration approach, a workflow process invokes a number of different services in a specific order because they have data and control dependencies between each other. This includes implementing logic that ties a set of services into an end-to-end process. That logic is then executed by a workflow engine that dispatches the right message to the right component and waits for the reception of the right message to activate the next service.
- In the e-Science context, the choreography approach provides a less accurate description of a workflow. It just documents the exchange of messages between a number of collaborating services. Choreography provides information about interfaces of services and how they plug to each other by defining public protocols that each service needs to be compliant with.

The benefit from using the orchestration approach is that in the implementation of the Grid Workflow Infrastructure, we just require one central service, the workflow engine that controls and executes the entire workflow process. Another advantage is that we can reuse the core concepts of BPEL4WS [BPEL], which is also based on the orchestration approach.

Using the choreography approach would have other disadvantages than just a weaker workflow description. In this scenario, each of the services involved would have to communicate with other services. The disadvantage of this approach is that for defining a

workflow, existing Grid Services could not be used. A new set of services had to be implemented that exchanges messages according to the choreography approach. Furthermore a central workflow engine would not be necessary.

Web Services technologies typically define their workflow in a way that all the data that is exchanged between the services involved, is transferred through the workflow engine. This is because they usually deal with a moderate level of data transmission across the Web Services. Since e-Science applications typically involve large amounts of data, the workflow engine could end up as a real bottleneck when applying the same approach. Therefore it is essential not to send the data itself but just references to it.

Life cycle management for Web Services is an issue that is now starting to be dealt with in the WS-Resource Framework. In Grid computing it is necessary to utilize resources efficiently and therefore not occupying them longer than necessary. Therefore instances of services taking part in the workflow should be created by the workflow engine from the specified factory and destroyed when they are not needed any more. In the Grid workflow infrastructure this is implemented with the help of notifications.

Certain Grid Services in the workflow will not be executing while others are. This may be due to the fact that the services, which need to execute earlier, run for weeks or that the service, which executes later needs data from the former to bootstrap itself. The Grid workflow infrastructure should therefore be able to handle this particular need.

5 Grid Workflow Execution Language

5.1 Syntax and Semantics of GWEL

GWEL is an XML based language that reuses ideas from BPEL4WS. Similar to the BPEL4WS process model, GWEL's process model represents a peer-to-peer interaction between services described in WSDL. Both, the Grid workflow and its partners are modeled as WSDL based Grid Services.

A GWEL workflow definition can be seen as a template for creating Grid Services instances, performing a set of operations on the instances and finally destroying them. The creation of instances is always implicit. Activities that receive messages can be annotated to indicate that the occurrence of that activity causes a new instance of the Grid workflow to be created.

5.2 Features

GWEL enables the specification of workflow descriptions for Grid Services in the GT3 environment. It is defined using an XML schema. The main features that are incorporated in the language are: Factory Links, Data Links, Variables, Fault Handlers, Life cycle and Control flow (also see figure 8).

- Factory Links: All the services that are part of the workflow have to be specified in the list of *factoryLinks*. The factories are identified throughout the GWEL document by a unique name, which is specified as part of the definition.

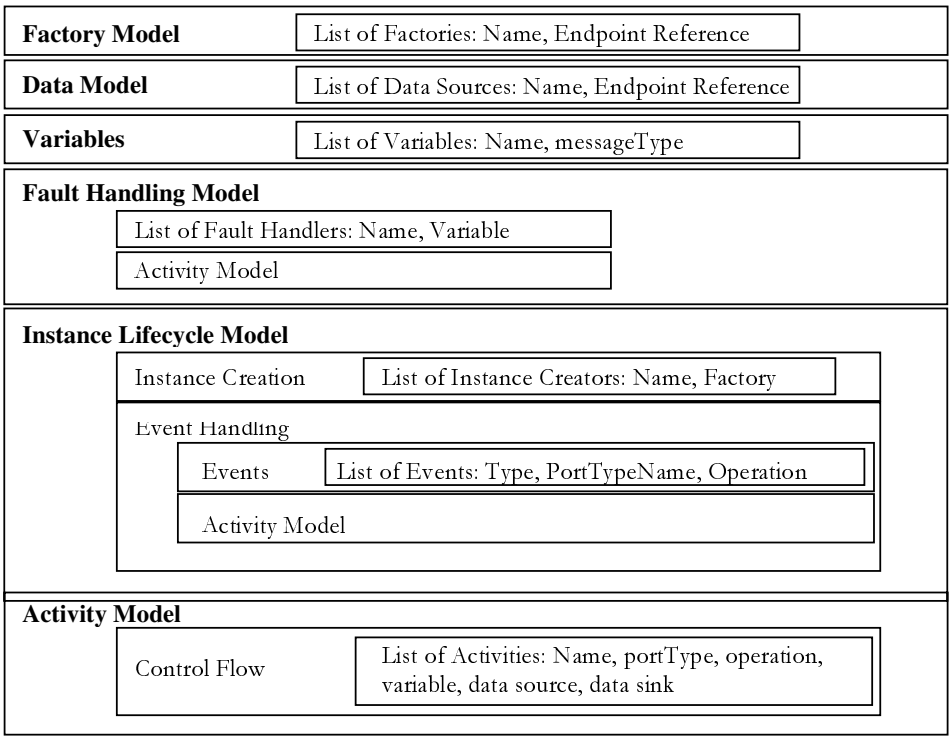

Figure 8: GWEL Elements

- Data Links: The *dataLinks* element has a similar structure to the *factoryLinks* element. Its purpose is to specify any data sources and data storage locations that will be used throughout the workflow.
- Variables: A variableidentifies the specific data exchanged in a message flow, which typically maps to a WSDL messageType. When a service is invoked, the appropriate variable is populated so that subsequent requests can access the data. Variables can be used to manage the persistence of data across Grid Services requests.
- Fault Handlers: GWEL introduces a mechanism for dealing with exceptions and faults. The aim of fault handling is to undo the unsuccessful work when a fault has occurred. The *faultHandlers* element specifies the faults that should the caught and the activities that should be executed upon a specific fault.
- Life Cycle: The *lifecycle* element deals with creating and destroying instances of factories. It can contain several *createInstance* sub-elements and one eventHandlers sub-element. In order to specify the creation of an instance, the new instance's name and the name of the factory have to be defined in the *createInstance* element. This element requires the name of the new instance and the factory name, form that the instance is created, for its attribute values.

Service destruction is managed through notifications. To make this possible, all services involved in the workflow have to be implemented as notification sources and send a notification to the workflow engine service, which is a notification sink. Upon receiving a notification, the workflow engine can react according to what activity is specified in the *onNotification* element. This element has the name of the sending instance, the port type and the operation as its defining attributes. Additionally an activity such as destruction or creation of an instance is defined. The most useful activity in there is probably a destruction of an instance, which is defined by the *destroyInstance* element.
The destruction element takes the instance name of the instance to be destroyed and its managing factory name as attribute values. Theoretically a whole new workflow could be defined as the activity element. If necessary, new instances could be created that execute certain operations and so on.
Instead or additionally to the *onNotification* element an *onAlarm* element can be specified. The purpose of this element is to set a certain time. After this time has expired, the activity inside the element has to be performed by the workflow engine. This activity is similar to the *onNotification* element. One of the purposes of this element is to handle worst-case scenarios of the workflow. For example, a workflow starts to execute and uses all the required resources but then stalls for some reason. Without human intervention, this scenario could last for a long time. Therefore one can set a time limit in the *onAlarm* element and after the alarm went off, destroy all instances and free resources.

- Control Flow: The *controlflow* element in the GWEL document deals not just with the control flowbut also with the data flow of a workflow, which is dictated by the *controlflow*. Similar to BPEL4WS, a GWEL control flow is a flow-chart like expression of an algorithm. Each activity in the process is classified as primitive activity or structured activity.

6 Grid Workflow Execution Engine

For the prototypical implementation of the workflow engine we have chosen Java, since that allowed the most rapid development. The architecture of the workflow execution engine is based on the workflow reference model [WRM], which was introduced in section 3.

A workflow file parser is responsible for recursively parsing GWEL documents and then dispatching the elements to the appropriate instruction classes. The workflow file parser uses JDOM to parse the document. JDOM is an open source library for Java-optimized XML data manipulations [JDOM]. Although it is similar to the W3C's DOM, it is an alternative document object model that was not built on DOM or modeled after DOM. The advantage of using JDOM is that we did not have to implement our own parser.

6.1 Life Cycle Instructions

The implementation of the life cycle instructions in the workflow document is one of the most important parts of the engine. The purpose is to handle the creation of instances directly and destruction indirectly. This is achieved through a creation and an event handler

mechanism. Creation of an instance is done through importing the stubs for the instance that has to be created. For our prototype development we assumed that the workflow engine has access to the stubs that are required for creating an instance. In real world applications this is usually no the case. However, the GWSDL document of a Grid Service is usually publicly available. Therefore, in a future version of the workflow engine, there would be a mechanism that downloads the required GWSDL file, generates the stubs from it and stores them in a local repository until they become necessary.

Event handlers are permanently listening for messages (onNotification, onAlarm) to arrive. This is achieved through the subscription method, which is available through GT3. Once an event handler gets a message it carries out what is specified in the GWEL document.

6.2 Control Flow Instructions

The task of the control flow implementation is to coordinate the control flow and the data flow of the workflow. This is achieved by dispatching the control flow instructions to the appropriate class for sequential or parallel execution. This mechanism works recursively. The top most structured activity determines which class handles the execution. Once an activity is acquired by the parser, Java reflection is used to execute the appropriate invocation. The benefit of using reflection is that it supports dynamic retrieval of information about classes and methods by name, and allows for their manipulation form within the executing program.

7 Case Study

7.1 Overview

In order to show the benefits and limitations of the Grid Workflow Infrastructure, we demonstrate an e-science workflow example that is specified as a GWEL document and executed by the workflow engine.

This example represents the simplest form of a workflow, a sequence of two activities. However this is sufficient to show the key points of the infrastructure. The e-science workflow we are presenting deals with the process of obtaining Bayesian Networks from data by using the unweighted L1 metric spanning tree algorithm.

A Bayesian Network expresses the dependencies and independencies of a set of variables [BN]. Those joined by arcs are dependent, those not are at least conditionally independent. The idea of a spanning tree is to start with a set of nodes and then find a set of undirected arcs between them that forms a tree.

For two variables A and B

- P(A&B) = P(A) P(B) if they are independent
- P(A&B) = P(A) P(B|A) if they have some dependency

P(B|A) may be smaller or larger than P(B) and if P(B) = P(B|A) then A and B are independent. This leads to the unweighted L1 metric dependency measure:
Dep(A,B) = |P(A&B) - P(A)P(B)|

Two nodes are joined if Dep(A,B) is large.

To produce Bayesian Networks form large data sets, we implement two independent services (L1 service and Sort service) that are orchestrated through the workflow engine according to a GWEL document. In the example we demonstrate how a Bayesian Network is computed, that can predict the probability of a patient suffering form Hepatitis C when certain characteristics are shown.

To achieve this, the L1 service reads in data, computes the unweighted L1 metric of each pair of variables and stores the intermediate result in a file. The sort service then reads in the intermediate result, orders the arcs depending on their unweighted L1 metric measure and then stores the final result in another file. From this result the Bayesian Network can easily be visualized.

7.2 GWEL Document

In the following paragraphs we show the main steps of how the workflow for this example can be specified as a GWEL document.

After having defined the top- level attributes, we specify the factory links and data links as follows:

```
<factoryLinks>
  <factoryLink name="L1">
  <handle name="http://146.169.50.157:8080/ogsa/services/
            workflow/core/l1/L1FactoryService"/>
  </factoryLink>
  <factoryLink name="Sort">
      <handle name="http://146.169.50.157:8080/ogsa/services/
                workflow/core/sort/SortFactoryService"/>
  </factoryLink>
</factoryLinks>
```

The handles of the factory links point to the URL where the factory services have been deployed.

```
<dataLinks>
   <dataLink name="raw_data_input_file">
                <handle name="../database/HepatitisC.txt"/>
   </dataLink>
   <dataLink name="l1_result_file">
                <handle name="../database/RESULT_HepatitisC_L1.txt"/>
   </dataLink>
   <dataLink name="sorted_data_file">
                <handle name="../database/SORTED_HepatitisC_L1.txt"/>
   </dataLink>
</dataLinks>
```

The next step is to define the lifecycle model. In the workflow we need two instances, one from each of the factories mentioned above. Furthermore we would like to destroy the instances after they have finished execution and sent a notification to the workflow engine.

```
<lifecycle>
   <!-- creating service instances form the factories -->
   <createInstance instance_name="l1_instance">
                                          <factoryLink name="L1"/>
   </createInstance>
   <createInstance instance_name="sort_instance">
                                          <factoryLink name="Sort"/>
   </createInstance>

   <!-- event handling-->
   <eventHandlers>
      <onNotification instance_name="l1_instance"
                                  portType="NotificationL1PortType"
                                  operation="computeL1">
          <destroyInstance instance_name="l1_instance">
              <factoryLink name="L1"/>
          </destroyInstance>
      </onNotification>
      <onNotification instance_name="sort_instance"
                                  portType="NotificationSortPortType"
                                 operation="sort">
          <destroyInstance instance_name="sort_instance">
              <factoryLink name="Sort"/>
          </destroyInstance>
      </onNotification>
   </eventHandlers>
</lifecycle>
```

Finally, we define the sequential control and data flow. The workflow engine invokes first the L1 instance which takes in the raw data and outputs the computed data file. Then the sort instance is invoked and gets the handle of the data file passed.

```
<controlflow>
    <sequence name="sequence1">

        <!-- interacting with the l1 instance -->
           <invoke instance_name="l1_instance"
                               portType="l1_port"
                     operation="computeL1"
                               variable="l1input"
                     dataInFrom="raw_data_input_file"
                     dataOutTo= "l1_result_file"/>

        <!-- interacting with the sort instance -->
           <invoke instance_name="sort_instance"
                               portType="sort_port"
                     operation="sort"
                               variable="sortinput"
                    dataInFrom="l1_result_file"
                     dataOutTo="sorted_data_file"/>
    </sequence>
</controlflow>
```

7.3 Workflow Execution

For executing the workflow, the GWEL document has to be specified first, then the client can be started. The client communicates with the workflow engine factory and creates an

instance of it. After that the workflow file is passed to the instance for execution. During execution the workflow engine communicates with the services involved and goes through the following main steps:

- Client invokes the workflow factory
- Workflow engine creates an instance
- Client submits GWEL document to the instance
- Workflow engine instance parses the document and creates a L1 service and a Sort service instance
- Workflow engine subscribes to both instances to receive notifications
- Workflow engine invokes L1 instance
- L1 instance inputs data from the data source, computes it and stores it
- L1 instance sends a notification to the workflow engine
- Sort instance inputs data, computes it and stores the final results
- Sort instance sends a notification to the workflow engine
- When the workflow engine receives the notifications, the instances are destroyed

This process is visualized in an activity diagram in figure 9.

8 Conclusions

In this paper we provide an investigation of problems that are involved in Grid workflow management and provide a solution to them, namely the Grid Workflow Infrastructure. This includes some background research on workflow management in general and a detailed investigation of Grid workflow requirements. Furthermore we describe how the proposed infrastructure matches these requirements. The result is the definition of an open infrastructure for Grid workflow management that is coherent with the standards of the Globus Alliance, the W3C and the Workflow Management Coalition. Finally we discuss an implementation of this infrastructure. The two main building blocks are the specification of the GWEL notation and the implementation of a workflow engine for the GT3 environment. With the help of a case study we demonstrate the feasibility of the claims of the Grid Workflow Infrastructure.

9 Future Work

During the course of our project, many new future research possibilities emerged. In the following paragraphs we outline the most relevant ones.

Currently we reuse concepts and elements of BPEL4WS in GWEL. However, it might be beneficial if GWEL would be a superset of BPEL4WS. That means on the one hand it incorporates all the elements of BPEL4WS and on the other hand it also incorporates the required Grid specific mechanisms. The advantage is that if BPEL4WS changes, GWEL would not have to be modified in order to be compliant with the latest version.

The most important area of improvement for the workflow engine is the introduction of a mechanism for finding Grid Services, which are specified in the GWEL document,

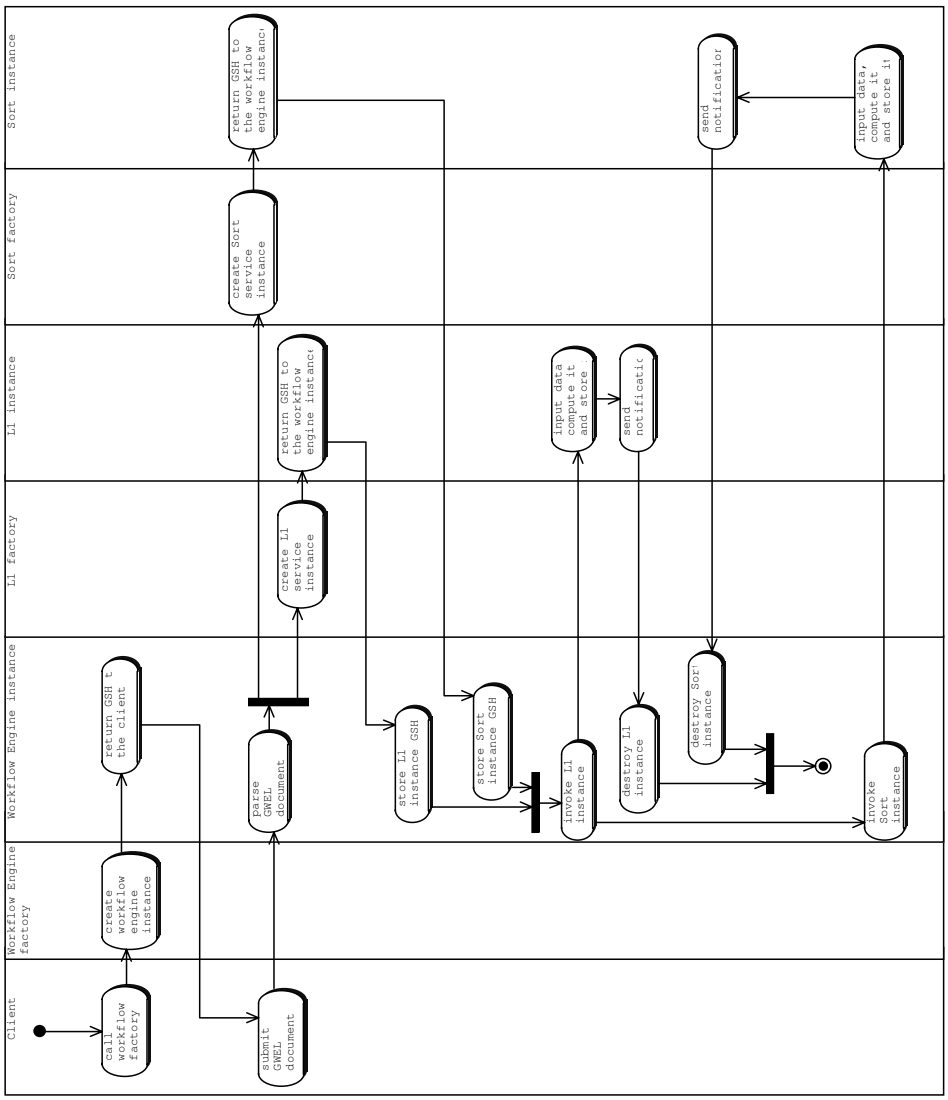

Figure 9: Activity diagramm

dynamically. This could include an instantiation service that can find factories on the Grid, downloads them to the local system and instantiates them. In order to implement this mechanism, issues like authentication, identification and rights to access certain resources would have to be looked at.

Another add-on for the workflow engine would be a graphical workflow editor. This editor could be implemented as a client for the workflow engine Grid Service. Its purpose is to create, modify and visualize GWEL documents.

So far we are not specifically considering performance issues in the Grid Workflow Infrastructure. In an improved version it might be worth to investigate the usefulness of performance data for more efficient workflows. Therefore it might be interesting to look at the findings of the GrADS project and maybe include some of its features in the Grid Workflow Infrastructure. The GrADS project is developing a software architecture designed to support adaptation and performance monitoring for reliably high performance in Grids [GRADS].

References

[BN] Duncan Gillies, Building networks from data, Intelligent Data and Probabilistic Inference lecture notes, http://www.doc.ic.ac.uk/%7Edfg/ProbabilisticInference/Bayesian.html, November 2002

[BPEL] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, S. Weerawarana, Business Process Execution Language for Web Services, Version 1.1, http://www-106.ibm.com/developerworks/library/ws-bpel/, May 2003

[BPELAnalysis] P. Wohed, W. van der Aalst., M. Dumas, A. ter Hofstede, Pattern Based Analysis of BPEL4WS, Technical Report FIT-TR-2002-04, QUT, Eindhoven University of Technology, 2002

[BuildingWS] S. Graham, S. Simeonov, T. Boublez, D. Davis, G. Daniels, Y. Nakamura, R. Neyama, Building Web Services with Java, Sams Publishing, 2001

[CCA] Madhusudhan Govindaraju, Sriram Krishnan, Kenneth Chiu, Aleksander Slominski, Dennis Gannon, and Randall Bramley, Merging the CCA Component Model with the OGSI Framework, Proceedings of CCGrid2003, 3rd International Symposium on Cluster Computing and the Grid, www.extreme.indiana.edu/xcat/publications/xcatogsi.pdf, May 2003.

[Flow] W. van der Aalst, Don't go with the flow: Web services composition standards exposed, IEEE Intelligent Systems, Jannuary/February 2003

[GRADS] Grid Application Development Software Project, http://hipersoft.cs.rice.edu/grads/gradsoft.htm

[GSFL] Sriram Krishnan, Patrick Wagstrom and Gregor von Laszewski, GSFL : A Workflow Framework for Grid Services, http://www-unix.globus.org/cog/papers/gsfl-paper.pdf, July 2002

[JDOM] Jason Hunter, JDOM and XML Parsing, Oracle DEVELOPER, http://otn.oracle.com/oramag/oracle/02-sep/o52jdom.html, September 2002

[OCGSA] Kaizar Amin, Sandeep Nijsure, Gregor von Laszewski, Open Collaborative Grid Service Architecture (OCGSA), http://www1.bcs.org.uk/DocsRepository/03700/3789/amin.htm, December 2002

[OGSA] Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, http://www.gridforum.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf, June 2002

[WorkflowPatterns] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, A. Barros, Workflow patterns, Technical report FIT-TR-2002-2, Queensland University of Technology, to appear in Distributed and Parallel Databases, Kluwer, 2002

[WRM] D. Hollingsworth, Workflow Management Coalition - The Workflow Reference Model, http://www.wfmc.org/standards/docs/tc003v11.pdf, 1995

[WSCI] S. Askary, A. Arkin, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, S. Zimek, Web Service Choreography Interface (WSCI) 1.0, www.w3.org/TR/wsci/, August 2002

[WSOrchestration] C. Peltz, Web services orchestration - a review of emerging technologies, tools, and standards, http://devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestration.pdf, 2003

[WSRF] The WS-Resource Framework, http://www-fp.globus.org/wsrf/default.asp, January 2004