Proposal for a bookchapter

Grid Workflow - An Integrated Approach

Gregor von Laszewski[1,2,*] and Mike Hategan[2,1]

[1]Argonne National Laboratory, Argonne National Laboratory,

9700 S. Cass Ave., Argonne, IL 60440

[2]University of Chicago, Computation Institute, Research Institutes Building #402,

5640 South Ellis Avenue, Chicago, IL 60637-1433

*Corresponding author: gregor@mcs.anl.gov

Abstract

Many scientific simulations and experiments require the coordination of numerous tasks posed by interdisciplinary research teams. Grids can provide access to the necessary high-end resources to conduct such tasks. The complex tasks and their interactions must be supported through convenient tools that protect the scientist from provisioning, and accessing such resources. To address this issue, we introduce a number of Grid abstractions that make the development of Grid middleware independent tools possible and allow for the integration of a number of commodity tools. Our vision is implemented through an integrated approach based on a layered architecture that attempts to bridge the gap between Grid middleware and scientific applications. Our abstractions include specialized services, a Grid workflow engine and language, and Grid faces which are graphical abstractions that can be employed in science portals and standalone applications.

1 Introduction

Grid computing has become a valuable asset in scientific and business communities for integrating distributed resources as part of virtual organizations. An important factor for the success of Grids is the development of sophisticated middleware based on standards. We have seen over the years a shift from API-oriented middleware to service oriented middleware. However, scientists that develop applications require an even higher level of abstraction in order to enable rapid prototyping and reuse of the Grid infrastructure in an easy fashion. Many scientific applications can benefit from expressing their tasks as part of a workflow. Due to the complexity of the Grid infrastructure and its evolving standards, it is even more important that much of this complexity be hidden from the end user. Sophisticated workflow frameworks can provide the necessary abstraction making the use of Grids even for scientists with little knowledge about Grids possible. Being able to express the scientific task of

discovery as structured workflows will have the added benefit that experiments can be described and documented in a concise way enabling for example support for experiment replication.

The chapter is organized as follows. First, we focus on the definition of workflow and identify some specific issues that are unique to Grid workflows and their workflow management systems. We present in more detail our architecture that addresses some of the workflow management issues and provide an implementation that is part of the Java CoG Kit. We list its unique features and demonstrate through simple language features that it enhances significantly the Grid experience. After we present more details about the status of our implementation, we conclude our chapter by identifying future research.

2 Workflow

Different definitions of workflow can be found in literature. However, the most widely accepted definition can be found in [21]. We quote

> *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. Whilst workflow may be manually organized, in practice most workflow is normally organized within the context of an IT system to provide computerized support for the procedural automation ...*

This leads to the following simple definition of a *workflow*:

> *The computerized facilitation or automation of a business process, in whole or part.*

In order to execute such a workflow we are in the need of a *workflow management system*. According to [21] it is defined as follows

> *A system that completely defines, manages and executes "workflows" through the execution whose order of execution is driven by a computer representation of the workflow logic.*

If we are in the business of doing science or working on the Grid, the same definitions apply to Grid workflows. The question arises is there anything different between business and science workflows? If we carefully examine the definitions given in [21], we discover a great deal of overlap. This includes for example, the need of a large number of resources in space and time. We find that the real difference is in (a) integrating Grid middleware into the workflow management system, and (b) focusing on the definition of workflow models that target use cases utilizing the Grid infrastructure.

Focusing on the Grid aspect, we can now define a Grid Workflow more formally. We refer to a *Grid workflow* as a concept that helps assisting the instantiation of a workflow model into an existing Grid infrastructure. In analogy

to the formalism common in computer science [12], it is simply defined as a set of Grid resources and services, a quality expectation defined by the user(s) and a workflow model acting on them. More formally,

$$\mathcal{W}_i = (\mathcal{G}_r, \mathcal{G}_s, \mathcal{Q}_u, \mathcal{W}_m)$$

where
$\mathcal{W}_i$ = Workflow instantiation,
$\mathcal{G}_r$ = Grid resources,
$\mathcal{G}_s$ = Grid services,
$\mathcal{Q}_u$ = Quality expectations from the user,
and a
$\mathcal{W}_m$ = Workflow model.

In the literature $\mathcal{W}_i$ is also sometimes referred to as concrete [13] or executable workflow [10], and $\mathcal{W}_m$ as abstract workflow that is concretized as part of an instantiation.

2.1 Issues in Grid Workflow

To focus on a more practical side of Grid workflows, we analyze which issues we must deal with to integrate Grids. To assist in this quest, we have listed a selected set of important issues in Figure 1that are related to Grid workflow management.

As part of the taxonomy depicted in Figure 1 we need to distinguish between the environment that a Grid workflow managing system targets and the lifecycle of a workflow. Within this chapter we will only target management issues that are typical for scientific and Grid workflow management issues. Hence, we need to deal with build, process, task, discovery, and project management issues that are related to the scientific discovery process. Additionally, we need to deal with concepts that are exposed as part of the Grid middleware and address issues such as virtualization, security, resource, data and information management. Each of these management issues poses further issues that are discussed in more detail in [20, 18].

The lifecycle of a Grid workflow consists of defining a workflow model based on a description methodology such as workflow languages or schemas. The definition of a workflow is done under extensive planning while considering services, resources, and quality assessments as discussed earlier. Checkpointing a workflow is of utmost importance as some workflows may run for month at a time. This poses special care on deployment strategies and adaptive software as the underlying services may change during runtime. Monitoring the workflow is important and must be made available through sophisticated interfaces to satisfy the novice and expert users. Lastly, it is crucial that the workflow adapts to ad-hoc that occur in the underlying Grid infrastructure or as part of the experiment process. Next we will discuss in more detail how workflows can support the scientific application management, as well as the integration of Grid application management issues.

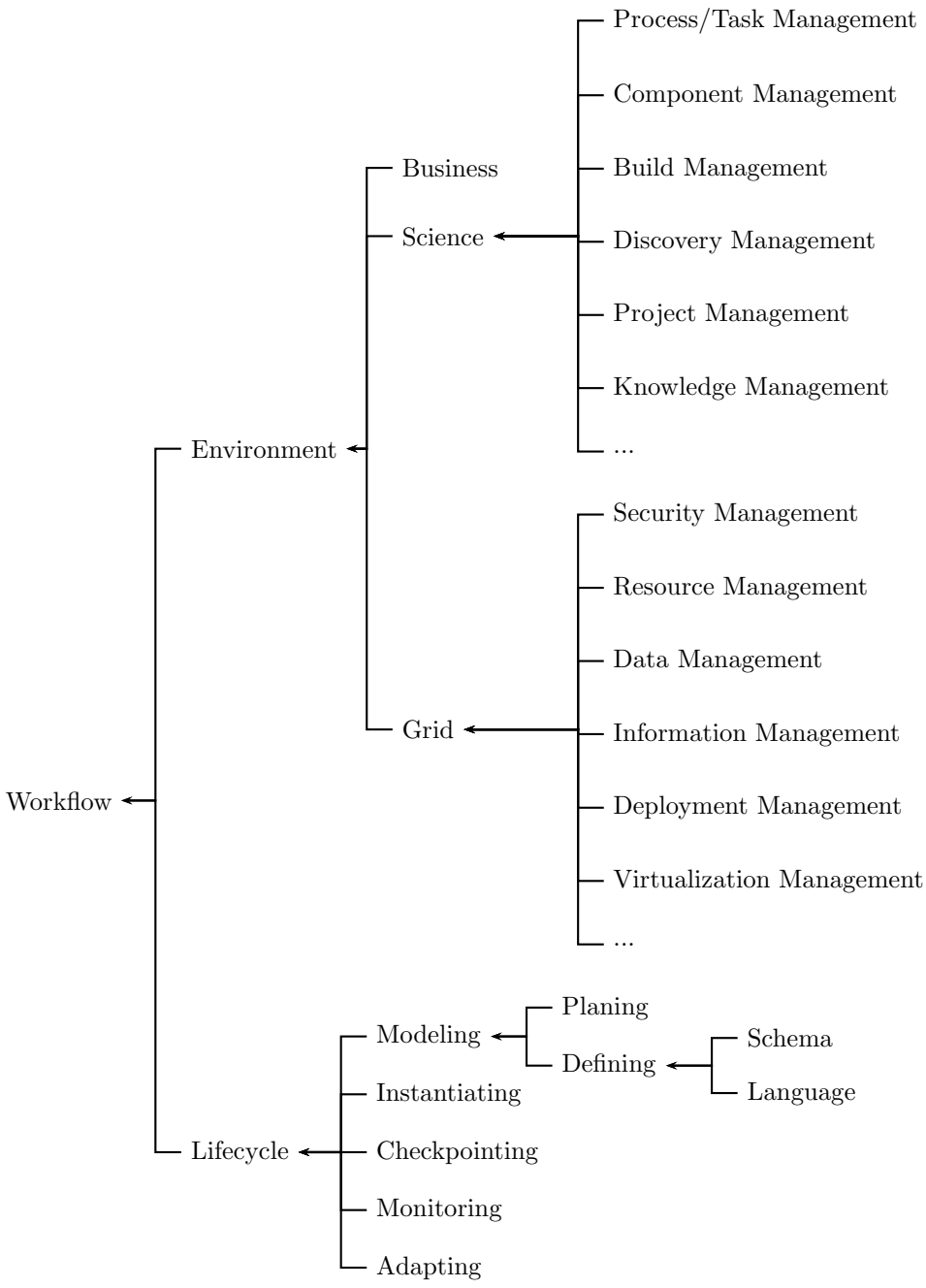


Figure 1: Workflows for Grids must address management issues posed by scientific and Grid applications

2.2 Science Application Management

In scientific applications we deal with management issues that support the scientific discovery process. Workflow abstractions build ideal tools to support the management of scientific applications. We have listed a set of management issues with increasing complexity and reuse of the previous management issues within.

Process and Task Management is an essential part of the scientific workflow. It allows the formulation of processes and tasks in order to define tangible items to be executed as part of a large and complex scientific discovery process.

Component Management involves the definition of components that implement strategies to execute processes and tasks and include them for example as part of a sharable repository within the scientific community in a community accepted component standard.

Build Management deals with bootstrapping and generation of software for later execution. Issues such as heterogeneous infrastructure and optimization of codes are of importance.

Discovery Management deals with issues related to the lifecycle of scientific experiment or task that may lead to scientific discovery.

Project Management deals with aspects that are outside of a single discovery or experiment process but deal with the management of a large number of them.

Knowledge Management comprises of gathering and dissemination of knowledge that is a direct result of a scientific activity. This may include the development of knowledge and data mining tools and associated portals [11] to integrate other members of the community.

2.3 Grid Application Management Issues

To employ the Grid appropriately as part of a workflow management system, it is important to address the following management issues.

Version and Deployment Management is important as we may deal with long running applications that are not effected by a change in the infrastructure that may happen during the workflow lifecycle.

Security Management deals with all aspects of enabling security such as single-sign-on, encryption, and data security. Special roles within a scientific discovery process may be assigned and the workflow may be assembled and monitored differently based on policies and privileges.

Resource Management deals with the integration of Grid resources and services that are part of a workflow instantiation.

Data Management deals with the data that may be generated during the instantiation of a workflow.

Information Management deals with information that will help the workflows to be dynamic with in respect to the available infrastructure and model.

Virtualization Management deals with the details of displaying and monitoring potentially disparate workflows as part of a large scale workflow management system.

2.4 Lifecycle Management

Each workflow management system must deal with the lifecycle of the workflow which includes the definition through workflow schemas or languages and the planning as part of the modeling effort. Workflows must be instantiated in order to properly assign and adjust resource requirements. Checkpointing and monitoring are of special importance to deal with and display fault situations.

3 Evolution of Grid Workflow

The development of Grid workflow systems is a natural progression as part of a combination of interwoven factors between available hardware, software, and scientific and business applications. Most recently we observe the emergence of Web and Grid technologies that together provide a path towards research activities in workflow. This path is marked by a number of phases.

Pre Web Phase. As part of this progress the desire to automate processes has lead to the development of early office automation software in the seventies to mid eighties leading to business of the mid eighties, and scientific workflow management systems of the late eighties. This view is shared with [22] in which a detailed historical perspective of this early development of workflow management systems is given and arranges the systems in chronological order.

Pre Grid Phase. Prior to the popularization of the term Grid, a variety of systems were used to define scientific workflows. Examples include HenCE[8] to define parallel programs visually, SciRUN [**?**] using a component based approach similar to that of AVS, a precurser to the Java CoG Kit that focuses on the aspects of Grid like workflow systems [17], and Webflow [**?**] to name only a view.

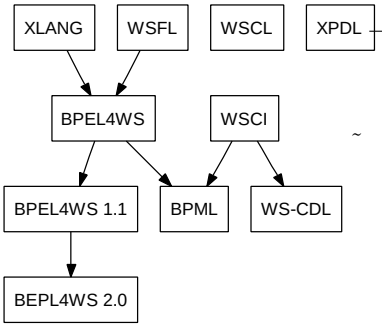
Early Grid Phase. Originally, only view Grid based systems dealt with workflows were available. The earliest of these systems was called GECCO and has now evolved to the Java CoG Kit [17]. Some time after that UNICORE [15] and Condor Dagman [6, 2] have been designed. Systems such as SciRUN have been augmented to include Grid scheduling facilities.

Grid Standrads Phases. Most recently the Grid community went through several phases to define Grid standards. The latest phase is still in progress and several standards have been submitted to OASIS. Hence the definition of Grid workflows based on the evolving standards was difficult. As part of this evolution, it was observed that higher level of abstractions can provide a convenient

interface to workflows for scientific calculations. Such a high level abstraction is defined for example by the Java CoG Kit not only on the workflow specification level, but also on the interface and API level. Also other systems such as Condor or Chimera have introduced language level abstractions that make it technically possible to adapt to the evolving standards as has been demonstrated through the last two years.

Web Services and Grid Phase. In the late 1990s with the maturing of Web Service standards a renewed interest in workflow arose to develop standards for interoperable web services including the concepts of coordination and choreography. In addition, the introduction of the Web Services Resource Framework and its associated standards has resulted in technologies developed by the Grid community that come closer to the Web services community. It is obvious that a large amount of overlap between both communities exists and that several technologies contributed by both groups enhance each other. Due to the fact of evolving standards not only in the Grid community, but also in the Web services community it is still difficult to develop technologies of lasting impact with wide acceptance that combine the three technologies: Grids, Web services and workflows. To illustrate the evolving standards let us focus on Figure 2 in which we have listed just a small number of relevant standards in relationship to web services and workflows. We observe that even in a matter of only three years a number of efforts have started, but that through merging the number has actually become smaller. Today, efforts such as BPEL4WS and WS-CDL have a strong momentum. Due to the evolution of the Grid standards efforts such as GSFL [19] that were designed prior to BEPL and WS-CDL have been halted. We expect that shortly other efforts in the Grid community will be initiated as is evident by the creation of workflow related research groups in the Global Grid Forum and the participation of companies in the GGF with vast interests to make Web services based workflow standards a success.

Grid and Web Upperware. Although it will be necessary to develop standards and methodologies to integrate current and future Grid middleware into a workflow strategy, it is important to recognize that the development of middleware reaches an increased level of sophistication through the development of advanced concepts and services. Such concepts have actually been demonstrated already by the previously mentioned systems such as the Java CoG Kit, Condor, and others while hiding as much of the underlying Grid middleware as possible in services and solutions that we term Grid Upperware. Systems such as the Java CoG Kit show that it is possible to develop upperware that can utilize different implementations of the Grid middleware as part of the workflow. This is of especial importance as we need to consider that the underlying middleware can be changing during the course of a long running workflow. More to this issue is discussed in a later section.



Specification	Introduced	Deprecated by
XLANG	Jun. 2001	BPEL4WS
WSCL	Mar. 2002	
WSCI	Aug. 2002	WS-CDL
WSFL	Aug. 2002	BPEL
XPDL	Nov. 2002	
BPML	Nov. 2002	
BPEL4WS 1.0	Jul. 2002	BPEL4WS 1.1
BPEL4WS 1.1	May 2003	
BEPL4WS 2.0 (draft)	Feb. 2005	
WS-CDL (draft)	May 2005	

Figure 2: Evolution of the standard in relationship to selected key web related technologies

4 Overview of Selected Grid Workflow Management Systems

A number of research groups have worked on the creation of workflow systems for Grid and non Grid environments. We will enumerate a subset of these systems in order to highlight features that have been found useful by the community. The systems we look at are Condor DAGMan, Globus, Pegasus and Chimera, Unicore, Triana. In more detail we will look at the Java CoG Kit workflow as it projects an integrated approach to Grid workflow fulfilling requirements of many users.

4.1 Condor DAGMan

DAGMan (Directed Acyclic Graph Manager) [2] is a recent addition to the Condor [3, 14] software. It is a meta-scheduler that extends the condor scheduler by allowing dependencies between condor jobs. The representation of dependencies is specified as part of a direct acyclic graph, where the nodes represent programs and the edges dependencies between the programs. The DAG is described in a simple text file in which each node represents a condor job that takes a number of input files and produces a number of output files. Besides specifying dependencies, DAGMan can support elementary error recovery and reporting. In each node a user can define a pre and a post script that is to be run prior and posterior to the condor job execution. Other frameworks use simply a separate graph node for this separation. DAGMan and Condor are implemented in C. A Grid Services interface based on the GT4 standard is under development. Condor is not open source and extensions to this framework are therefore not possible by the general user community. The advantage of the system is the ease of specification of DAGs as ASCII files, the integration with the well known Condor software. The disadvantages are that no XML specification

language is available, the system is closed source, and that it is more difficult to integrate your own customized schedulers and Grid mapping frameworks. No Graphical user interface is provided. Condor is well suited to interface to Grid backends.

4.2 Globus Toolkit

Globus is a Grid middleware toolkit that has evolved significantly during its presence within the community. It started with an API layer to support Metacomputing and has now been totally redesigned to project a Service oriented model for Grid computing. Concepts such as the Grid security infrastructure that allow delegation of jobs to be part within a production Grid or the introduction of state full services in contrast to stateless Web services are innovations that are being standardized by the Global Grid Forum in conjunction with efforts at IETF and OASIS. The Globus toolkit does not support workflows as pioneered by the Java CoG Kit within the Grid domain. However the newest Globus toolkit (version 4) uses a file stag-in and stage-out pattern and is now integrated part of the job execution service (called GRAM) distributed with the toolkit. Hence it provides high level features similar to that of a single Condor DAGMan node. In addition, it provides simple reliability features of the stage-in and stage-out process. Due to this similarity it is relatively straight forward for DAGMan to integrate Globus Toolkit Grid services as part of the backend infrastructure and the other way round. The advantage of the Globus Toolkit is that it projects the quasi standard of Grid middleware and that a large number of projects base their development on the services included within the Globus Toolkit. The disadvantage is that the Globus Toolkit has undergone three phases of development with technically very different service implementations. However, based on the newest development and the promotion of the protocols through standard bodies the system will become less prone to significant changes. In addition, Toolkits such as the Java CoG Kit have proven an interesting approach to provide a set of consistent service descriptions even though the underlying services and their protocols change significantly.

4.3 Pegasus and Chimera

Chimera [5] and Pegasus [4, 13] are systems that work hand in hand to define a workflow model and to instantiate it within a Grid environment. The workflow model is centered on the concept of virtual data which specifies data as part of a number of transforms. Hence, the input to chimera can be expressed as partial workflow descriptions that specify input files on which logical transformations are applied, leading to output files. Instead of referring to an output file the workflow designer can refer to the partial workflow that creates this output file. This is a concept is especial useful if the operations to obtain the modified data are cheaper than maintaining a new copy of the data. The specifications of the transformations are defined with the help of a virtual data language. Chimera is associated with tools and methods to define the virtual data workflow models and Pegasus is responsible for instantiating the workflows within a Grid environment. Pegasus supports the mapping of partial workflows into the Grid

environment to allow late binding of the resources in order to adapt to the changing Grid environment during the workflow instantiation and execution. The advantage is that transformations may be cheaper than the storing of the result of transformations. The disadvantage is that the system to express such transformations can become quite complex and that intermediary data may not be available at a time it may be needed by other components.

4.4 Unicore

UNICORE (Uniform Interface to Computing Resources) [15, 16] is providing a seamless, secure, and intuitive access to distributed resources. It deals with job management including creation, submission, and monitoring, data management, application support, and single sign-on. It integrates resources as part of a meta computing environment. A graphical user interface is available through which the user can specify with the programs to be executed on compute resources. As part of the UNICORE abstract job model DAG-based workflows can be specified In addition it contains conditionals and repetitive execution of job groups or tasks as part of the workflow model. The advantage of the system is that it is easy to define workflows to generate programs that work on remote machines. The disadvantage is that it is that the integration with the newest Grid standards is in progress.

4.5 Triana

Triana presents itself as a problem solving environment integrating a visual interface with data analysis tools. Triana workflows are based on the notion that every piece of work is part of a specific experiment and its execution needs to be coordinated as part of a workflow. Experiments are generated by a scientist, which is supported by a Triana workflow management system to handle the experiments as efficiently and effectively as possible. As other workflow systems, Triana is based on a layered architecture that separates the visualization and representation from the instantiation of the workflow. Triana workflows are WSFL like representation of task graphs task graph consist of three types of elements: tasks, control links and data links, where a task represents an operation, a control link defines the sequence of tasks in the model, and a data link describes the flow of data between tasks. Triana is a arge software and contains a sophisticated visualization framework for workflow graphs. The advantage of Triana is the visualization framework, which makes it possible to maintain moderately sized workflows in graphical fashion. The disadvantage is that the use may not as targeted to the Grid as other efforts such as Condor or Java CoG Kit.

4.6 SCIRUN

SCIRUN is a scientific workbench that focuses on the presentation of components and their integration as part of a graphical user interface similar to AVS. It allows users to construct, manage, and debug scientific simulations in a variety of scientific disciplines. In SCIRUN components are defined that can be connected with each other through the description of input and output relation-

ships of the components. The assembly of such component can be viewed as a workflow since SCIRUN allows for parallel and conditional execution of tasks. SCIRUN has components integrated that allow running of the computational intense tasks on the Grid. Besides integration in Grids distributed computing frameworks such as CORBA are also supported. It includes a sophisticated GUI workflow modeler as well as controls to modify a workflow while it is running. The advantage of SCIRUN is its component model and the ability to generate sophisticated visualization of scientific data. The disadvantage is that the user obtains the power of SCIRUN through its user interface.

5 Java CoG Kit Workflow

To support a high level workflow vision as introduced in Section 2.1 we have chosen to adapt an incremental approach based on feedback from our users. Through our interactions we have identified that many application developers desire to program the Grid in familiar higher level frameworks that allow rapid prototyping. The Java CoG Kit provides such rapid prototyping abilities as part of the Java and XML frameworks. It integrates a variety of commodity tools, protocols, approaches, methodologies, while accessing the Grid through Grid toolkits. The Java CoG Kit has evolved from a project that exposes much of the Globus Toolkit functionality through Java to a framework that contains a significant feature enhancement to the Globus Toolkit architecture. Based on its features, it is used by, and distributed in part with the Globus Toolkit version 3 and version 4. Often, users of CoG Kit technologies do not know that they are using them.

5.1 Features

In order to support our vision of integration workflows management tools into the Java CoG Kit, we have identified a number of higher level abstractions including Grid tasks, transfers, jobs, queues, hierarchical graphs, schedulers, and workflows, and control flows, which make the development of Grid programs easier. However, in contrast to other Grid efforts we have provided a mechanism in our workflow management framework that allows the integration of a variety of Grid and commodity middleware in an easy-to-comprehend framework based on the concepts of protocol independent abstractions, providers, and bindings. These are discussed below.

Providers. We have introduced the concept of Grid providers that allow different Grid middleware to be used as a part of an instantiation of the Grid abstractions. Hence the programmer does not have to worry about the particularities of the Grid middleware. Through dynamic class loading we have the ability to do late binding against an existing production Grid. This includes the implementation of the Grid (task) abstractions, version binding against existing Grid Toolkits, and resource binding.

Abstractions. We have identified a number of useful abstractions that help in the development of elementary Grid applications. These abstractions include job executions, file transfers, workflow abstractions, job queues

and can be used by higher level abstractions for rapid prototyping. As the Java CoG Kit is extensible users can include their own abstractions and enhance the functionality of the Java CoG Kit.

Bindings. Through these concepts the Java CoG Kit protects your development investments by protecting you from changes to the Grid middleware.

Based on these elementary concepts, we designed a layered architecture that allows the gradual enhancement of workflow capabilities within our application (see Figure 3).

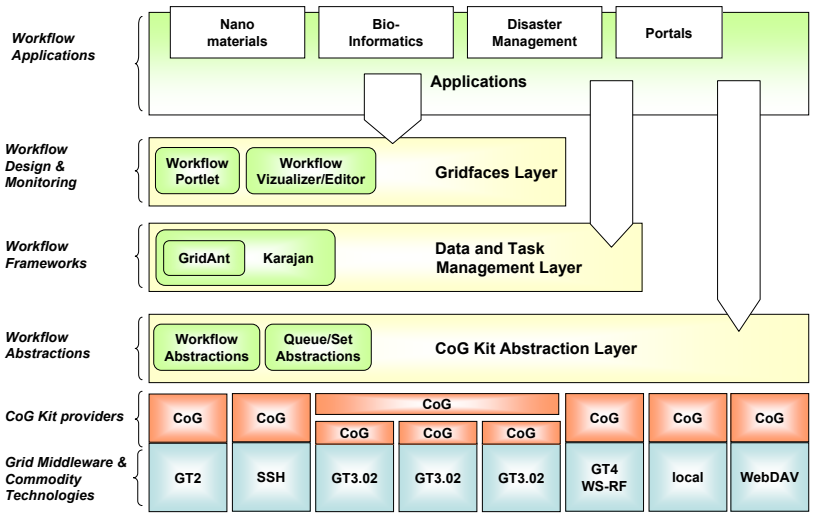


Figure 3: The layered approach of the Java CoG Kit provides mechanisms for incrementally enhancing workflow management components.

On the bottom of the architecture we have the typical Grid middleware. Above it lies our Java CoG Kit abstraction layer that focuses on job submission file transfer and authentication. With the help of CoG providers, we can now access a number of different Grid middleware. One may ask, why bother, why not stick to one Grid middleware? Practical experience shows that the vision of one middleware can only be achieved if everyone agrees to the same protocols and standards. However, this has in the past been an issue of content, as (a) either the standard evolves (b) the middleware evolves (c) or the production Grids had incompatible software versions installed. Due to our advanced architecture, we can address the issues by developing appropriate providers. As

we will see in the later part of the chapter, this will allow us even to enable switching between different Grid middleware in a running workflow.

The simplest form of workflow abstractions that we support are embedded in the definition of a task. This may include a file transfer, a job submission, an authentication or any other task that has to be done. Our tasks are defined to have a status that can be queried. Based on this elementary definition, we define task queues and sets.

On the next higher level, we define APIs, tools, and services that help in the coordination of such tasks. It is handled by a workflow engine that we have derived from GridAnt. However, the scalability of GridAnt was limited. Hence, we designed a complete new workflow engine with many more advanced language features. This workflow engine is also called *Java CoG Kit workflow Karajan engine*.

At the next level, we define Gridfaces that are visual abstractions shared amongst stand-alone applications or portals. With the help of Gridfaces, it will become easy to develop visuals for either portals or stand-alone applications.

The value of the Java CoG Kit workflow solution lies in its simplicity and its ability to be integrated in a solution that allows us to expose workflow to a variety of users. As indicated in Figure 4, we are prototyping a system that provides an API based on abstractions and the integration of services. Command line interfaces, web portals, and a Grid Desktop that expose the workflow functionality in a convenient user interface are also under development. The integration of these tools is possible through an integrated but modular architecture as depicted in Figure 5. Our workflow system contains at its heart the workflow engine that is augmented by a variety of tools, our workflow specification languages, and programing frameworks to reuse workflows and the engine. It is important to note that the workflow engine is itself an abstraction and could for example be replaced by a specialized version suitable and customized for a particular community.

In the rest of the chapter, we illustrate a number of important features of the Java CoG Kit workflow to highlight some of its capabilities.

5.2 Language

The Java CoG Kit workflow engine is based on a language that is derived from ant and GridAnt. We have integrated flow control constructs such as condition, loops, as well as, sequential, parallel blocks. Figure 6 shows a subset of the workflow patterns that are supported within the Java CoG Kit workflow engine. It is important to note that we also have the ability to define the pattern of hierarchical tasks as indicated in Figure 7.

Specialized grid enabled tasks for job submission file transfer and authentication are available that can be augmented with appropriate providers.

5.3 Workflow Gridfaces

An important aspect of a workflow system to appeal to the scientific user community is to provide elementary capabilities for visualizing the workflow and its status. We have developed a prototype visualization tool that can render a

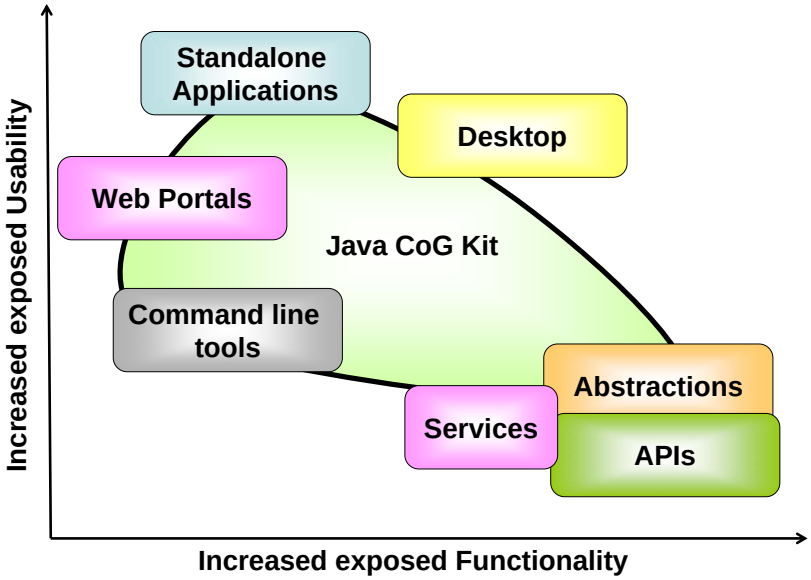


Figure 4: The Java CoG Kit integrates several mechanisms that together build a powerful workflow management system for Grids.

graph based on the task model (Figure 8). To be able to render large graphs we have introduced two capabilities. First, we are developing a capability to render graph nodes that can themselves be graphs. Secondly, we enable a graph overview window and allow zooming into portion of that graph. As each node can be annotated with a label it is possible to provide convenient help descriptions for the user through an annotation. The workflow display component is also contains functionality to control and monitor the workflow. Once the user presses the start button, he will be able to monitor the progress of the graph. It is possible to set breakpoints in the graph that is, the workflow gets executed only till right before the task that is to be executed. Hence, it is possible to support debugging as well as tasks in the graph that require interactive steering and can not be automated. In addition to our graph vizualizer, we have developed a workflow debugger, that allows us to step through the workflow and observe the variables associated with it (9). This debugger has also the ability to utilize breakpoints so the workflow designer can focus on the tasks that may need further inspection.

6 Workflow Language

In this section, we present a subset of simple examples that illustrate the syntax of our XML based workflow language. It is easy to derive other more convenient representations as discussed in Section 7. However, we have decided to present

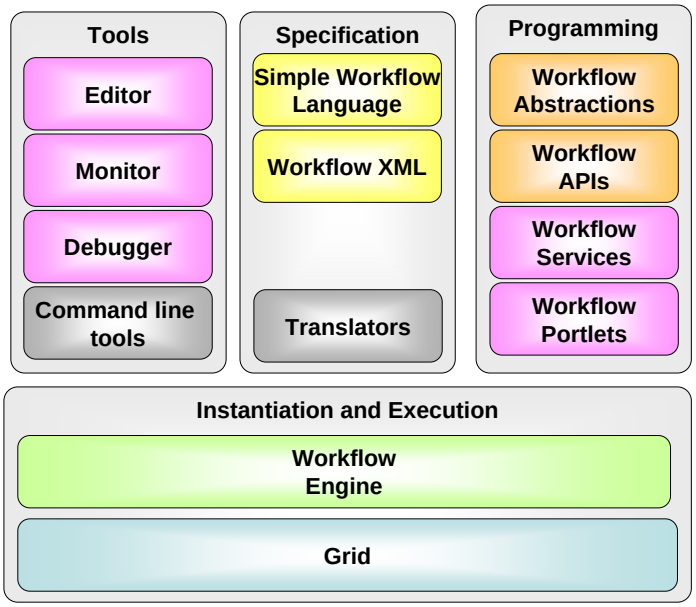


Figure 5: Components of the Java CoG Kit workflow framework

our features in XML to demonstrate the similarity to the originating GridAnt system. To keep the examples short we use "... to indicate a block of statements that is not shown but are essential to the illustration of a language feature.

6.1 Modularization

Here, we describe the functions that allow us to modularize our workflows.

Project. We have introduced the concept of named projects as introduced in GridAnt. This allows us to refer to different workflow projects by name.

```
<project name="cog"> ... </project>
```

Element. One of the most important features of our workflow language is the element tag that allows the definition of new elements into the language. Elements have a name and a number of parameters. Once defined, the element can be reused under its name in the program code. Assume we define the following element.

```
<element name="printParameters"
         arguments="input,output">
   <echo message="Input = {input}"/>
   <echo message="Output = {input}"/>
</element>
```

Once it is defined, it can be invoked in the following way.

```
<printParameters input="in.dat" output="output.dat">
```

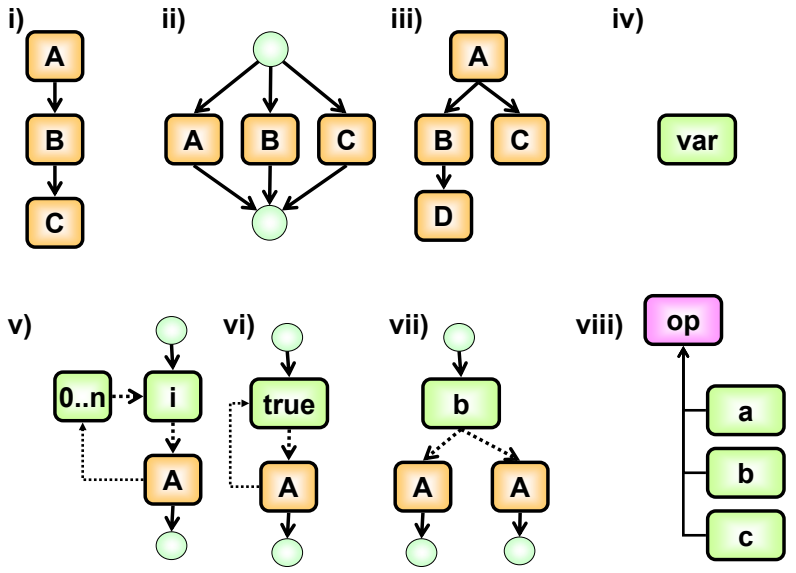


Figure 6: Selected workflow patterns that are supported by the Java CoG Kit. i) sequence ii) parallelism iii) fork iv) variables v) for loop vi) choice viii) operators

Namespaces. Through the use of namespaces the definition of elements can be augmented automatically with a prefix. This can be of special importance if multiple workflow developers have provided elements with the same name. Therefore, it allows an easy way of distinction. Assume we define the following namespace

```
<namespace prefix="gregor"> ... </namespace>
```

Then, each element that is defined in the block is preceded by the prefix "gregor:".

Include. To structure element definitions and other Java CoG Kit workflow scripts it is possible to organize them in separate files similar to the C include statement. Such files can be included in a workflow as follows:

```
<include name="cogkit.xml">
```

Descriptions and Annotations. Each element can be augmented with an annotation and description which is useful to provide metadata for the element. The description should usually include a more detailed textual explanation of what this element is for. The annotation is typically used in visualization tools and may include a label for the element that is supposed to be displayed by such tools. Annotations and descriptions can be changed during the course of a workflow instantiation.

```
<element name="element" annotation="label"
   description="Demonstrating annotations"/>
```

6.2 Variables and Operators

Variables and operators can be used to make the workflow dynamic.

Variables. We have introduced an untyped variable. Similar to python, the value is interpreted in its context.

```
<set name="index" value="0"/>
```

This variable can be referenced in the workflow as "{index}"

Lists, Ranges, Maps. We have also introduced additional data structures that make the programming of advanced workflows easier. These features include lists, ranges, and maps or hash tables. We illustrate here only an example for a range as it will come in handy for our definitions of loops. For more information on these data types we refer to the manual [9].

```
<set name="range">
  <range from="1" to="10"/>
</set>
```

Operators. We have defined a number of standard operators that help in conditional statements. Such operators include math and boolean operators such as sum, product, equal, not, and many more. An example on how a simple math operator can be used is given below. It calculates the sum of 1,2, and 3.

```
<math:sum>
   <argument value="1"/>
   <argument value="2"/>
   <argument value="3"/>
</math:sum>
```

6.3 Workflow Structural Elements

Many workflow systems are just based on the definition of direct acyclic graphs. They do have properties such as lack of loops which makes it possible to conduct easy program verification. However in practice such systems fall short as many Grid workflows require the generation of many repeated tasks as part of for example parameter studies. Hence our workflow engine not only supports hierarchical graphs, but also loops, conditions and even recursion.

Condition. The workflow language contains an if construct including elsif blocks. The syntax is given by an if tag in which a conditional is included. The statments to be executed are enclosed in then ore else tags.

```
<if><condition> ...  </condition>
    <then> ... </then>
    <elsif><condition> ...  </condition>
       <then> ... </then>
    </elsif>
    <else> ... </else>
</if>
```

Choice. A choice will execute child elements in succession until one completes without error. A choice has a transactional behavior and buffers its return values. This allows us to more easily parallelize a choice as demonstrated in a forthcoming section. Please note that in the sequential block all elements are executed in sequential order.

```
<choice>
  <sequential>
    <print>You won't see this as
           we create an error. </print>
    <generateError message="Error"/>
  </sequential>
  <sequential>
    <print>You will see this</print>
  </sequential>
</choice>
```

Loops. The definition of loops is straight forward and may include ranges and conditions.

```
<for name="i" in="{range}">
  ...
</for>
```

6.4 Recursion

In some cases it is useful to be able to define a recursive element returning a value. We show this feature in Figure 10. However, we would like to point out that with a simplification of the syntax as discussed in Section 7 this code becomes significantly simpler to understand. The example we have selected is to define a recursive element that calculates a factorial. It is important to note that within the definition of the element we can call the element tag to conduct the recursion. The return value is specified in the *number* tag.

6.5 Parallelism

We have introduced elements to express sequential and parallel blocks of workflow statements. In a sequential block, all elements are executed in sequential order while in a parallel block the statements are supposed to be executed in paralllel.

```
<sequential> ... </sequential>
<parallel>   ... </parallel>
```

Scope Variables are scoped based on their position within sequential and parallel blocks. The following example demonstrates that the value of the variable is dependent on its scope.

```
<parallel>
  <sequential>
    <set name="a" value="2"/>
    <echo>a is {a}</echo> <!-- 2 -->
  </sequential>
  <sequential>
    <set name="a" value="3"/>
```

```
    <echo>a is {a}</echo> <!-- 3 -->
  </sequential>
</parallel>

<echo>a is {a}</echo> <!-- 1, no ambiguity -->
```

Parallel Loops. In case the order of execution does not matter, the loop can be augmented with the argument mode that can than be set to parallel [1].

```
<for mode="parallel" name="i" in="{range}">
  ...
</for>
```

Parallel Choice. We have also implemented the feature of a parallel choice that is specified through the mode argument[2]. The parallel choice can be of importance when the execution of its individual element blocks will take a long time to finish. Starting the executions in parallel and taking the one that finishes first can increase the performance in exchange for possibly unused cycles. The following code will start the first and the second block in parallel. It will return this output generated by the second block and print "Second" as its execution time is shorter.

```
<Choice mode="parallel">
  <sequential>
    <wait delay="100"/>
    <print>First</print>
  </sequential>
  <sequential>
    <wait delay="50"/>
    <print>Second</print>
   </sequential>
</parallelChoice>
```

Grid Tasks. We have introduced a number of elementary Grid tasks that we have identified to be most useful to many in the Grid community. These tasks are abstractions above the Grid middleware and allow easy adaptation towards different versions and protocols of the underlying Grid middleware.

At this time we distinguish the tasks grid:execute that executes a program on local or remote grid resources, grid:transfer that transfers files between resources, and grid:authentication that authenticates to the grid.

In each case it is possible to define a provider that determines that protocol and mechanism that is used to execute the appropriate Grid task. We have developed providers for GT2, GT3, and GT4, as well as condor. It is possible to include your own providers into the Java CoG Kit through our extensive use of Java interfaces as part of our software engineering effort. The beauty of our approach is that the developers do not have to deal with the internal working of the Grid services. The only thing they need to know is which version

[1] At present we have not implemented the mode but instead we use the element tag parallelFor

[2] At present we have not implemented the mode but instead we use the element tag parallelChoice

of grid middleware they run on their Grids. The rest is provided by the Java CoG Kit. This enables a significant simplification in the use of Grids as we not only abstract above the Grid middleware but also enhance it by providing a sophisticated workflow framework.

The following example illustrates the simplicity of our approach. The example starts authenticates to a grid, copies a file to a remote machine, and compiles that program on the remote machine.

```
<project name="RemoteCompilation">
  <include file="cogkit.xml"/>
  <task:authenticate provider="GSI"/>
  <set name="host"
       value="hot.mcs.anl.gov"/>
  <set name="path"
       value="/usr/local/j2sdk1.4.2_05"/>
  <task:transfer desthost="{host}"
                 provider="gridftp"
                 srcfile="Test.java" />
  <task:Execute host="{host}" provider="gt2"
    executable="{path}/bin/javac"
    arguments="Test.java"/>
</project>
```

6.6 Error Handling

Since many things can go wrong while computing on a Grid, dynamic error handling is an important asset to any Grid workflow. We include the ability to catch errors and react on them through the definition of an *onError* element. The error element can be augmented with a match argument that specifies a regular expression that when true allows to invoke the program block enclosed in the onError element. Our example shows how to pop up a graphical user interface for entering the credentials in case they have expired or they can not be found.

```
<onError match="(._Expired credentials detected._)|
(._Proxy file._not found._)">
  <echo>Invalid credentials detected.</echo>
  <executeJava
mainClass="org.globus.cog.karajan.
           util.ProxyInitWrapper"/>
  <echo>Restarting failed element</echo>
  <executeElement element="{element}"/>
</onError>
```

Checkpointing. We have added the ability to checkpoint our workflows by assuming the availability of checkpointing within an element. If an element not able to checkpoint, the workflow is rolled back to the position where the last successful checkpoint is set. We also can add breakpoints into the workflow in order to have more features for debugging or to allow interactive manipulation of a running workflow.

6.7 Java and Python Language Support

We have already implemented the feature of executing arbitrary Java programs as part of our workflow as demonstrated in the next image. This is achieved by calling the underlying Java interpreter from the JRE. However, we have not distributed this code yet with the Java CoG Kit as we need to verify licensing restrictions.

```
<java>
  System.out.println("Hello World");
</java>
```

In addition to being able to execute arbitrary Java Programs in a block of statements, we are able to call Java methods within the workflow through the executeJava Element. The example given starts up a Graphical user interface for creating a Grid proxy.

```
<executeJava
mainClass="org.globus.cog.karajan.
           util.ProxyInitWrapper"/>
```

Similar to Java tag, we intend to also support python programs with the help of Jython.

```
<python> ... </python>
```

6.8 Performance Augmentations

We have implemented some simple mechanisms to support performance measurements and to increase the performance through element caching.

Timers. In some cases it is important that the time taken for of particular workflow operations is measured. This may be useful to help in determining if a particular calculation is overdue in order to invoke a termination or a rescheduling. Hence, we have introduced named times that can be queried.

```
<timer name="timer1"> ... </timer>
```

we can time a block and with

```
{timer1}
```

we can refer to its value as it is exposed as a variable to the workflow.

Element Result Caching. We have implemented the concept of element result caching into our framework. In case we have an element that returns a result, this result is cached and can be reused at a later time without recalculating the element. This feature can be switched off by using the parameter caching="false". This feature is most useful in case the results calculated occur repeatedly in the workflow. The calculation of the factorial is a good example for the use of element caching. Here the method signature can be used to store the result and reuse it at a later time. This leads to significant performance improvements.

7 Syntax Translators

We have designed our language based on XML so it is possible to verify the inputs easily and to allow others to develop source-to-source code compilers or translators that provide a more convenient form of specifying Grid workflows. We also have designed a simplistic language that can be translated into our XML specification and be executed by our workflow engine. Figures 10 and 11 contrast how simple an alternative syntax can be. In our simplistic language we simply replace the beginning and end tags with operators that use beginning and closing braces $op(...)$ instead of $< op > ... < /op >$. The * indicates the multiplication and the ? the if condition. It is feasable that other source-to-source translators could be developed thus enabling other frameworks to make use of our workflow engine. We envision that the community could develop partial translators for subsets of Java, Python, or BEPL2WS to name just a few.

8 Performance

We have conducted simple performance experiments and found that the performance of the Java CoG Kit workflow engine is quite reasonable. On an Intel Pentium 4 Mobile CPU with 1.60GHz and 512MB of memory, running Linux 2.6.11.5, we have achieved the following performance numbers. We have been able to execute on average 14233 workflow elements per second. Internally, we have handled on average 28490 events per second, with an average event time of 29.5 μs. Hence, it will exceed by far the speed that is provided for executing typical Grid tasks and will not cause a bottleneck.

9 Evaluation and Integration

In [1] we have demonstrated the usefulness of the workflow for nanomaterials applications. Based on our earlier experience with other scientific applications, we are now in a position to start implementing solutions that address the workflow management issues as depicted in Figure 1. We will proceed from top down to illustrate how our workflow system can help. It is clear that our specific definition of tasks within the Grid workflow management and the introduction of definable tasks as part of workflow elements we can support the desire of scientists to express processes or tasks. Because we introduced a module concept and scoping, we can design components that can be stored in a component repository that we are developing at present. As the Java CoG Kit workflow is derived from GridAnt, it supports the build process and can assist in the build management of software by integrating compilers on remote machines as demonstrated in some of our examples within this chapter. Due to the ability to integrate Java methods into the program and design new elements with return values, we can also support access to Grid information services. Project and knowledge management can also be supported, which is currently under development as part of a chemistry application [19]. The integration of Grids in our workflow system is of elementary importance. We have shown that we interface with Grids on several levels and even deal with different versions. Along with

the features presented in the Java CoG Kit and the Globus toolkit, our workflow system has the potential to integrate many management issues that are listed in Figure 1. We hope to leverage from other efforts to also integrate support for virtualization and information management. However, these efforts are still ongoing research projects and are undergoing significant changes [7]. Hence we have not yet addressed them as yet.

10 Conclusion

This chapter provides the first detailed write up about the Java CoG Kit workflow system. We have analyzed some essential concepts that are part of a Grid workflow system. Based on this analysis and our experience with the Java CoG Kit that won the best poster award at the last Supercomputing 2004 conference we came to the conclusion that we needed to enhance our efforts in developing an easy interface to the Grid that includes the ability to utilize workflows. In contrast to some other efforts, our workflow system is open source and extensible. It allows the integration of commodity tools and frameworks through our language bindings. We have made sure to project a future oriented architecture that allows the gradual enhancement while addressing explicit problems with today's and tomorrows Grid middleware. Examples of such issues are presented in the chapter and include the change of the deployed Grid middleware and versions as part of a workflow instantiation of long running workflows. Our abstractions have proven effective in changes against protocols and APIs of well known Grid middleware. Other features that we support are the dynamic adaptation of workflows at runtime which is based on our module concept. Our integrated approach not only includes the availability of a sophisticated language but also the development of more sophisticated tools that make the manipulation and handling of the workflows more easy. Our future goal is to develop additional tools that can either be exposed as stand-alone applications or as part of a portal through portlets developed together with the Open Grid Computing environments project [11].

11 Availability

The Java CoG Kit workflow can be downloaded from http://www.cogkit.org.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38. DARPA, DOE, and NSF support Globus Project research and development. The Java CoG Kit is supported by DOE SciDAC and NSF Alliance. We like to thank Deepti Kodeboyina for her help in improving the presentation of this chapter[19].

References

[1] Kaizar Amin, Mihael Hategan, Gregor von Laszewski, Nestor J. Zaluzec, Shawn Hampton, and Al Rossi. GridAnt: A Client-Controllable Grid Workflow System. In

37th Hawai'i International Conference on System Science, Island of Hawaii, Big Island, 5-8 January 2004. Available from: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--gridant-hics.pdf.

[2] DAGMan (Directed Acyclic Graph Manager). Web Page. Available from: http://www.cs.wisc.edu/condor/dagman/.

[3] Condor Version 6.4.7 Manual, 2003. Available from: http://www.cs.wisc.edu/condor/manual/v6.4/2_11DAGMan_Applications.html.

[4] Ewa Deelman, James Blythe, Yolanda Gil, and Carl Kesselman. Pegasus: Planning for Execution in Grids, 2002. Available from: http://www.isi.edu/~deelman/Pegasus/pegasus%20overview.pdf.

[5] Ewa Deelman, James Blythe, Yolanda Gil, and Carl Kesselman. *Grid Resource Management*, chapter Workflow Management in GriPhyN. Kluwer, 2003. Available from: http://www.isi.edu/~deelman/Pegasus/grm_chapter.pdf.

[6] J. Frey, T. Tannenbaum, I. Foster, et al. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*, 2001.

[7] The Globus Project. Web Page. Available from: http://www.globus.org.

[8] HeNCE (Heterogeneous Network Computing Environment). Available from: http://www.netlib.org/hence/.

[9] Java Commodity Grid (CoG) Kit. Web Page. Available from: http://www.cogkit.org.

[10] Kepler. Web Page. Available from: http://kepler.ecoinformatics.org/.

[11] Open Grid Computing Environments. Web Page. Available from: http://www.ogce.org.

[12] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization*. Dover Publications Inc., 1999.

[13] Pegasus. Web Page. Available from: http://pegasus.isi.edu/.

[14] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the Grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002. Available from: http://media.wiley.com/product_data/excerpt/90/04708531/0470853190.pdf.

[15] Unicore. Web Page. Available from: http://www.unicore.de/.

[16] Unicore plus final report. Joint Project Report for the BMBF Project UNICORE Plus, 2002. Available from: http://www.unicore.org/documents/UNICOREPlus-Final-Report.pdf.

[17] Gregor von Laszewski. A Loosely Coupled Metacomputer: Cooperating Job Submissions Across Multiple Supercomputing Sites. *Concurrency, Experience, and Practice*, 11(5):933–948, December 1999. The initial version of this paper was available in 1996. Available from: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--CooperatingJobs.pdf.

[18] Gregor von Laszewski and Kaizar Amin. *Grid Middleware*, chapter Middleware for Commnications, pages 109–130. Wiley, 2004. Available from: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--grid-middleware.pdf.

[19] Gregor von Laszewski, Branko Ruscic, Kaizar Amin, Patrick Wagstrom, Sriram Krishnan, and Sandeep Nijsure. A Framework for Building Scientific Knowledge Grids Applied to Thermochemical Tables. *The International Journal of High Performance Computing Applications*, 17(4):431–447, Winter 2003. Available from: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--knowledge-grid.pdf.

[20] Gregor von Laszewski and Patrick Wagstrom. *Tools and Environments for Parallel and Distributed Computing*, chapter Gestalt of the Grid, pages 149–187. Parallel and Distributed Computing. Wiley, 2004. Available from: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--gestalt.pdf.

[21] The Workflow Reference Model. The Workflow Management Coalition, January 1995. Available from: http://www.wfmc.org/standards/docs/tc003v11.pdf.

[22] Michael zur Muehlen. *Workflow-based Process Controlling. Foundation, Design, and Implementation of Workflow-driven Process Information Systems*, volume 6 of *Advances in Information Systems and Management Science*. Logos, Berlin, 2004. Available from: http://www.workflow-research.de/Publications/Book/index.html.

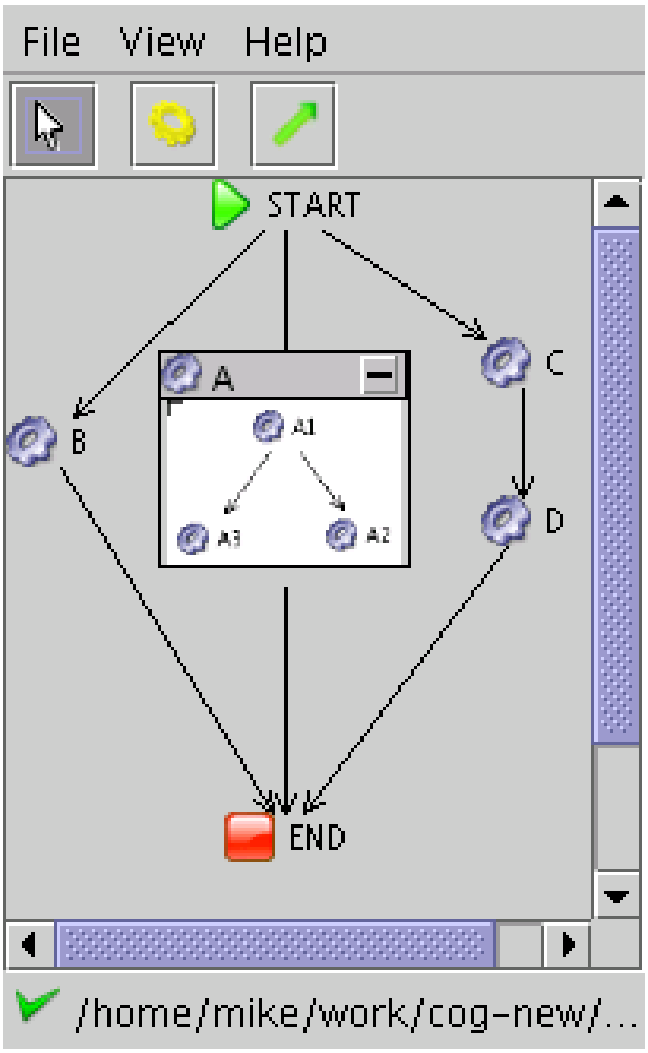


Figure 7: The Java CoG Kit is planed to handle hierarchical graphs.

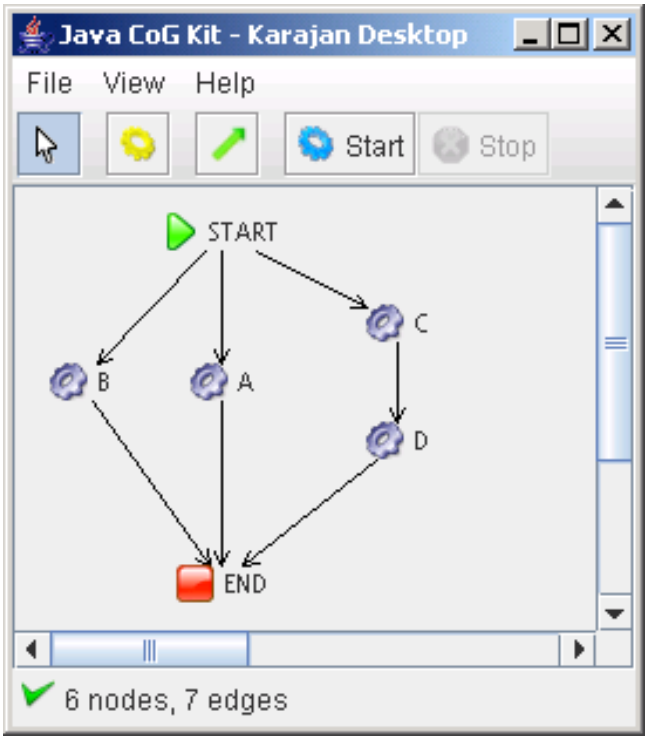


Figure 8: Parallel and sequential constructs simplify the definition of graphs.

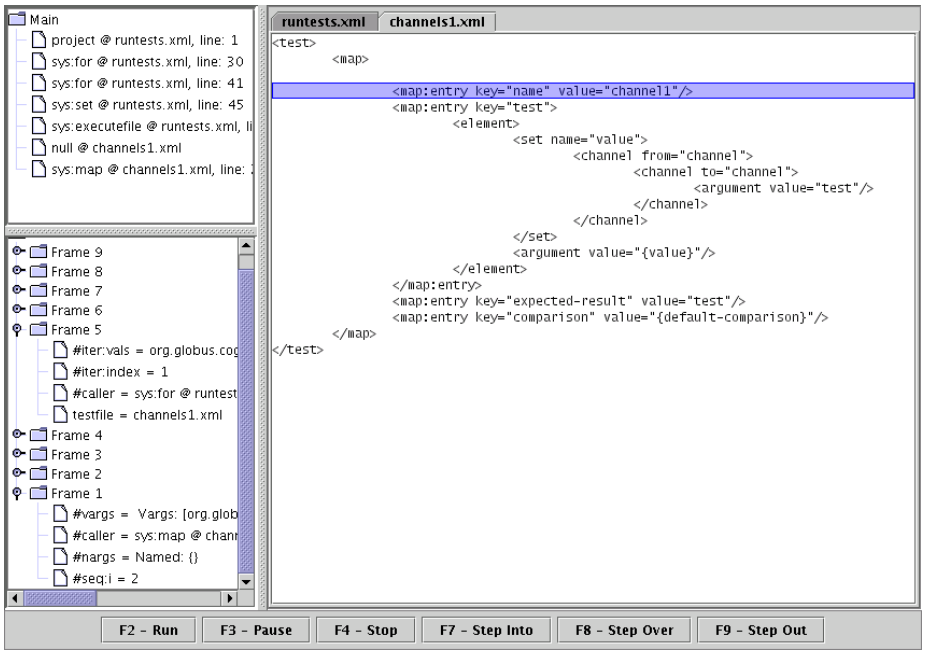


Figure 9: The Java CoG Kit has a simple sequential debugger that allows the stepwise debugging of the workflows.

```
<element name="factorial" arguments="n">
  <if>
    <condition>
      <math:equals value1="{n}" value2="1"/>
    </condition>
    <then>
      <number>1</number>
    </then>
    <else>
      <math:product>
        <variable>n</variable>
        <factorial>
          <math:sum>
            <variable>n</variable>
            <number>-1</number>
          </math:sum>
        </factorial>
      </math:product>
    </else>
  </if>
</element>
```

Figure 10: This factorial specification demonstrates the ability to use recursion.

```
element(factorial, [n],
  if(
    ?(<=(n, 1))
    then(1)
    else(
      *(n, factorial(-(n, 1)))
    )
  )
)
```

Figure 11: The simplified Java CoG Kit workflow language allows the specification of more complex code in a much more simple fashion.