Java CoG Kit Workflow Concepts for Scientific Experiments

Gregor von Laszewski[1,21]
Argonne National Laboratory, Mathematics and Computer Science Division
Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60440
University of Chicago, Computation Institute,
Research Institutes Building #402, 5640 S. Ellis Ave., Chicago, IL 60637-1433

Contents

Java CoG Kit Workflow Concepts for Scientific Experiments

Gregor von Laszewski[1,2†]

Argonne National Laboratory, Mathematics and Computer Science Division

Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60440

University of Chicago, Computation Institute,

Research Institutes Building #402, 5640 S. Ellis Ave., Chicago, IL 60637-1433

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. 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 bridges the gap between Grid middleware and scientific applications. Our abstractions include specialized services, a Grid workflow engine and language, and Grid faces – graphical abstractions that can be employed in science portals and standalone applications.

1 Introduction

One of the important issues to address in Grid-based scientific and business computing is the coordination of tangible tasks through workflows. In recent years we have seen a shift to services oriented architectures in both disciplines. However, we still recognize the fact that a higher-level of abstraction is needed in order to hide the complexity of developing service-oriented middleware. Many application users in business or in the scientific community have simple needs that require easy abstractions and tools to utilize the complex Grid infrastructure. One of these needs is to make the use of Grids possible, even for scientists with little knowledge about Grids, through the availability of sophisticated workflow frameworks that can provide the necessary abstraction. It is a necessary step towards the vision of enabling

*Corresponding author: gregor@mcs.anl.gov

†Corresponding author: gregor@mcs.anl.gov

Grid computing so that it is seen as a utility for sophisticated workflow scenarios. We focus in this paper on workflow concepts that are important to enable scientific experiments.

The paper 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 an overview about the workflow activities conducted by the Java CoG Kit team members. Next we present in more detail our architecture that addresses some of the workflow management issues, and we provide an implementation that is part of the Java CoG Kit. The architecture includes an abstraction model, a workflow language based on XML and Gridant, and a Grid shell. We then present more details about the status of our implementation. We conclude our paper by identifying future research tasks.

2 Grid Workflow

Since many definitions of workflow exist in literature, it is important to identify the terminology that we use throughout this paper. Our definition of workflow is based on the one introduced in [37]:

> *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 definition is later summarized in [37] to the following definition of the term *workflow*:

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

A *workflow management system* (WoMS) executes such workflows according to [37]:

> *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.*

Hence, if we are *in the business of doing science or working on the Grid*, the same definitions apply to Grid workflows. We have to ask ourselves, what is different between business and scientific workflows? If we carefully examine the definitions given in [37], we discover many commonalities such as the need for a large number of resources in space and time. The real difference lies in (a) integrating Grid middleware into the workflow management system, and (b) focusing on the definition of workflow models that target cases utilizing the Grid infrastructure. We expect that the boundaries between business and scientific computing will soften even further in the future.

Formally, we define a *Grid workflow model* as a concept that helps the instantiation of a workflow model into an existing Grid infrastructure. It is 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
$\mathcal{W}_m$ = Workflow model.

Others refer to $\mathcal{W}_i$ also as concrete [15] or executable workflow [11] and to $\mathcal{W}_m$ as abstract workflow that is concretized as part of an instantiation.

One of the important issues in advanced Grid workflow is the concept of dynamically changing workflows and their associated states. We simply denote the change through the addition of a time stamp as part of our simple formalism and an adaptation function a at time T. Hence,

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

$$\downarrow a^T$$

$$\mathcal{W}_i^{T+1} = (\mathcal{G}_r^{T+1}, \mathcal{G}_s^{T+1}, \mathcal{Q}_u^{T+1}, \mathcal{W}_m^{T+1})$$

Naturally, the changes can either apply to the resources, the services, the quality expectations, and the abstract definition of the workflow model, or a combination of them. In order to specify such a transformation function, a series of previous workflow instantiations may be considered to determine a future instantiation. Hence,

$$\mathcal{W}_i^{T+1} \leftarrow a(\mathcal{W}_i^{T_{j_0}}, ..., \mathcal{W}_i^{T_{j_m}}),$$

where $m \in [0, t], j_0, ..., j_1 \leq m$, and $j_0 \cap ... \cap j_1 = 0$

2.1 Management Issues in Grid Workflow

To derive a sophisticated workflow management system, we have to analyze more closely the issues that lead to the practical reuse of such a system within a Grid environment. In Figure 1 we have categorized some of the management issues that are very important for enabling a powerful Grid workflow management system.

As part of the categorization depicted in Figure 1 we distinguish between the *environment* that a Grid workflow managing system targets and the *lifecycle* of a workflow. Although we target both business and scientific computing, we mostly focus our attention on managing scientific workflows. As part of this management issue we need to deal with the project,

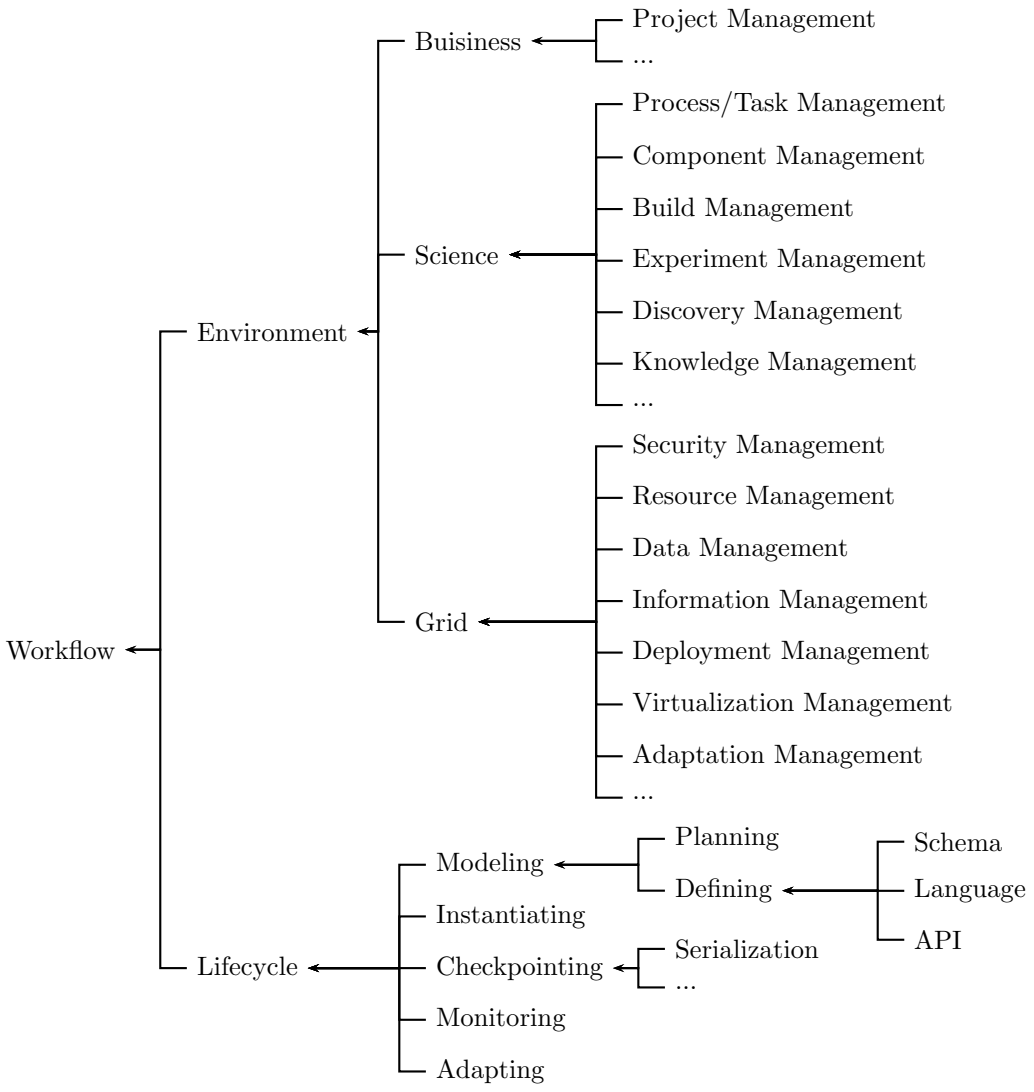

Figure 1: Workflows for Grids must address management issues posed by scientific and Grid applications.

process, task, build, discovery, and project management issues that are related to the scientific discovery process. Furthermore, we need to address management issues that are part of the Grid middleware and address issues such as virtualization, security, resource, data and information management. Each of these management issues poses further concerns that are discussed in more detail in [35, 26].

The lifecycle of a Grid workflow deals with defining a workflow model based on a description methodology, such as workflow languages or schemas. Extensive planning leads to a workflow model while considering services, resources, and quality assessments as discussed earlier. To guarantee long running workflows checkpointing is a necessity.. Monitoring the workflow must deal with issues such as fast response vs. performance and scalability. User interfaces should support the convenient display of large workflows typical in scientific communities. Further, the workflow should adapt to ad-hoc changes that occur in the underlying Grid infrastructure or as part of the experiment process. Next we discuss in more detail how workflows can support the scientific application management, as well as the integration of Grid application management issues.

2.2 Workflows for Scientific Experiments

The management issues related to scientific experiments are manifold and increase in complexity the more general they are. We depict in Figure 1 a subset of management issues that need especial attention for scientific experiment management.

Processes and Tasks need to be managed in order to define tangible items to be executed as part of the complex scientific discovery process. Processes and tasks are the elementary parts of workflows for experiment management.

Components need to be managed to allow reuse within the community as part of the complex execution of processes and tasks.

Build processes need to be managed to simplify the bootstrapping and generation of software for later execution.

Experiments need to be managed to deal with individual discoveries conducted during single or multiple experiments. Parameter studies as part of the experiment management are common.

Discovery processes need to be managed as part of the evaluation processes of the experiments. It is important to derive a strategy that evaluates the experiments and encourages the discovery of new results.

Knowledge extracted from the discovery process need to be managed and communicated to other scientists so the results can be shared.

2.3 Grids for Scientific Experiments

To employ the Grid appropriately as part of a workflow management system, one must address the following management issues.

Deployment processes need to be managed as we may deal with long-running applications that should not be affected by a change in the infrastructure during the workflow lifecycle.

Security processes need to be managed while dealing 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.

Resources need to be managed as part of the sophisticated Grid infrastructure that are reused by workflow instantiations.

Data management will help to move the necessary inputs for the workflow processes to the appropriate resources.

Information that describes the state of the Grid and the workflow needs to be managed in order to adapt appropriately to the environment conditions.

Virtualization management addresses the concept of virtualization in Grids including management of members of the virtual organization.

Adaptation management deals with the changing environment as part of the workflow instantiation. Workflow models may be modified depending on the state of the environment in which the workflow is operating. An example is the selection of a particular workflow dependent on user requirements or changes in the infrastructure.

2.4 Lifecycle Management

A workflow management system deals 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 faulty situations.

3 Java CoG Kit Concepts

Next we describe the concepts to support workflows in the Java CoG Kit as part of a sophisticated workflow management system and framework. One of the goals of this system is to support the experiment management in scientific applications. Hence, the workflow management system must interface between the scientific experiment, the applications that are executed as part of the experiment, the collaborations necessary to plan and organize such experiments, and the experiment infrastructure – which includes that not only the Grid infrastructure, but also commodity infrastructures and laboratory infrastructures related to the experiment itself. We have depicted the role of the workflow management system in Figure 2.

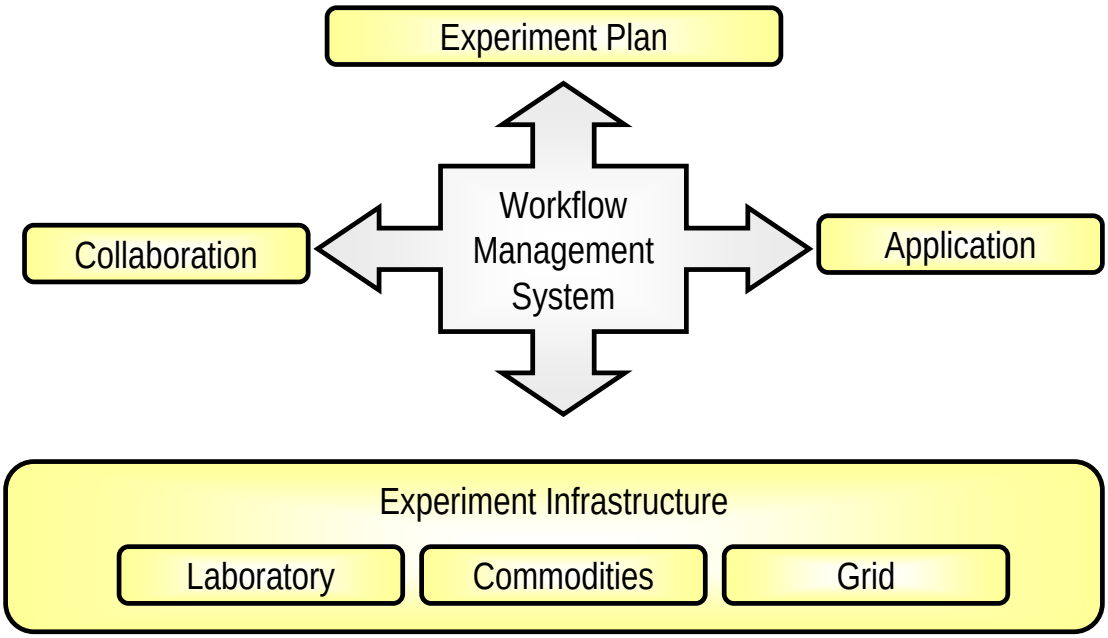

Figure 2: The layered approach of the Java CoG Kit provides mechanisms for incrementally enhancing workflow management components.

3.1 Historical Perspective

Our view in regard to experiments and workflows has been shaped by our long collaborations with experimental scientists. Activities that center on the concept of workflow have a long tradition within the members of the Java CoG Kit team. We list in Table 1 a selected number of activities and their association in time. Based on these collaborations, accompanied by our Grid experience [25], we have developed several concepts as part of the Java CoG Kit [30] that help us in the development of sophisticated workflow systems. These concepts include abstractions for queuing systems and workflow graphs with simple dependencies as found in [23]; event-based notifications and monitoring as found in [23, 29, 28, 34]; a simple execution pattern [29], now termed *cog pattern*; hierarchical graphs [29]; structured control flow with loops and conditions [31]; visual interfaces to augment the workflow execution [23]. One additional important concept is an adaptive workflow framework, as described formally in Section 2. As part of the work described in [23] we have defined a dynamically adapting scheduling algorithm that chooses optimized algorithms based on performances

measurements to identify a set of resources that fulfill a given high-level function such as a matrix multiplication. The task of conducting the matrix multiplication is specified at the time the workflow is specified. However, its instantiation and the appropriate selection of which algorithm to chose are conducted during runtime at a later time.

Table 1: Timeline of the Java CoG Kit workflow activities

Date	Refernces	Activity
1994-1995	[23]	Interactive parallel computing environment for meta-computers
1995-1996	[22, 24]	A workflow metacomputing system in Java
Aug. 1996		*Start of the Globus Project*
1997		A fault tollerant high throughput broker with application notification was demonstrated at SC'1997
1997	[29]	A graphical user interface to a workflow system fore scheduling jobs on the Grid, demonstrated at SC'1997 (GECCO)
2000	[29]	The term Java CoG Kit is introduced
2001	[4]	Adaptation to the new Globus Toolkit, reimplementation selected features of previous systems.
2002	[36, 33]	Grid Service Flow Language
2002	[8]	The jglobus module is distributed with GT3
2003	[27, 1]	GridAnt, a workflow system based on ant
2003		Karajan, a much enhanced version of Gridant (interfaces with GT2, GT3, SSH through providers)
2004	[4]	GridShell, a simple shell for sequential flows (interfaces with GT2, GT3, SSH through providers)
2005	[31, 32, 4]	Java CoG kit interfaces with GT4, and Condor through providers

3.2 Key Concepts

Next, we will introduce several concepts of the Java CoG Kit and explain how they are helpful for the definition of a sophisticated workflow system that supports experiment management. Based on our interactions with developers and users we have identified that many desire to have an integrated but modular system that allows them (a) to program the Grid in familiar higher level frameworks that permit rapid prototyping, (b) to have a framework in which workflows on the Grid can easily expressed, (c) to have a framework that supports monitoring the state of a workflow through visual components, and (d) to have a system that is easy to maintain and deploy. As depicted in Figure 3 the Java CoG Kit integrates a variety of concepts to address the varying functionality and usability aspects of the user communities touched by a Grid workflow system. Hence, end users will be able to access the workflow system through standalone applications, a workflow desktop, or portals. Command line tools allow users to define workflow scripts easily. Workflow programming is achieved through services, abstractions and APIs. Additionally, we integrate commodity tools, protocols, approaches, methodologies, while accessing the Grid through commodity technologies and Grid toolkits. Through this integrated approach the Java CoG Kit provides significant

enhancements to the Globus Toolkit. The Java CoG kit is based on a modular design allowing easy integration of enhancements developed by the community. Today, the Java CoG Kit distributed in part with the Globus Toolkit version 3 and version 4. However, many of the features demonstrated in this paper are only available through the software found on the Java CoG Kit Web pages [10].

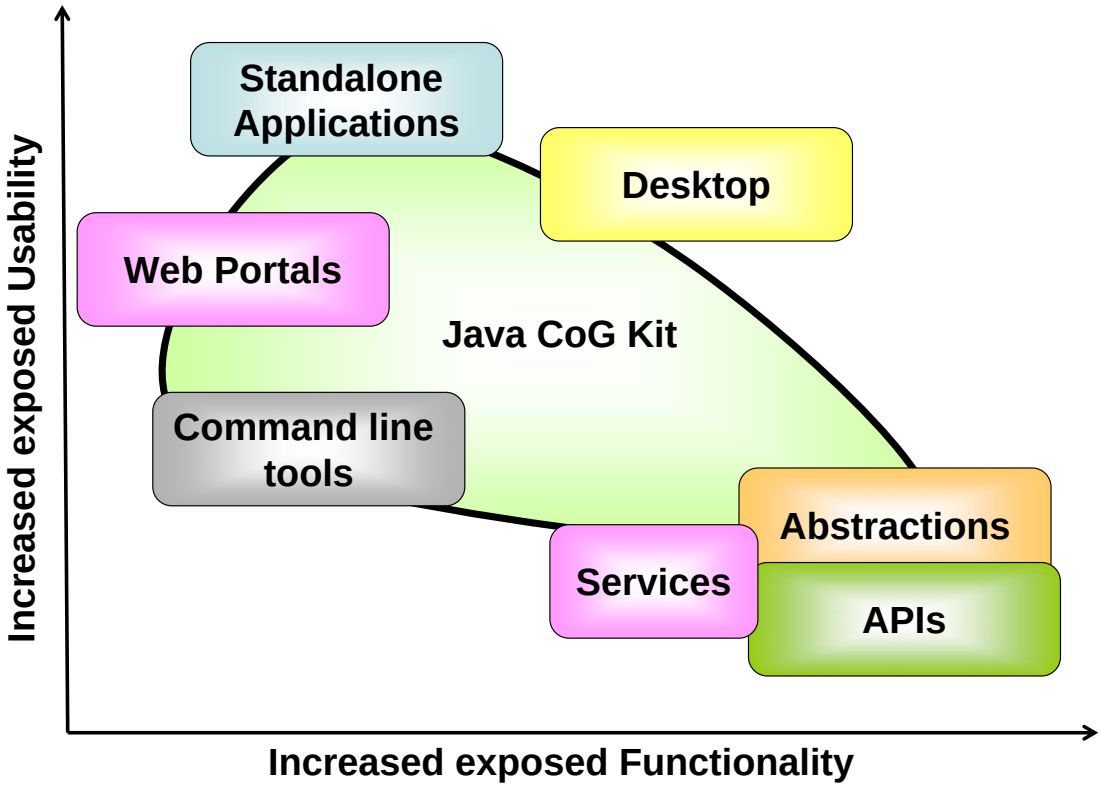

Figure 3: The Java CoG Kit integrates several mechanisms that together build a powerful workflow management system for Grids.

3.3 Architectural Design

In order to support our vision of integration workflows management system and tools to be part of the Java CoG Kit we use a layered architecture that allows easy separation between concerns. This architecture allows for the gradual enhancement of workflow capabilities within our application (see Figure 4). We describe the architecture shortly while proceeding from the bottom up.

The bottom of the architecture integrates with typical Grid middleware. Above it we have designed an abstraction layer that focuses on job submission file transfer and authentication. With the help of providers, we enable access to different Grid middleware. More sophisticated abstractions to enable task and workflow management are part of the next level. This architecture defines APIs, tools, and services that help in the coordination of such tasks. Coordination of the tasks is handled either by a workflow engine, a grid shell, or simply by abstractions that define workflow graphs. Gridfaces build the next higher level of abstractions. They are visual abstractions shared amongst stand-alone applications or portals. We show our layered architecture in Figure 4.

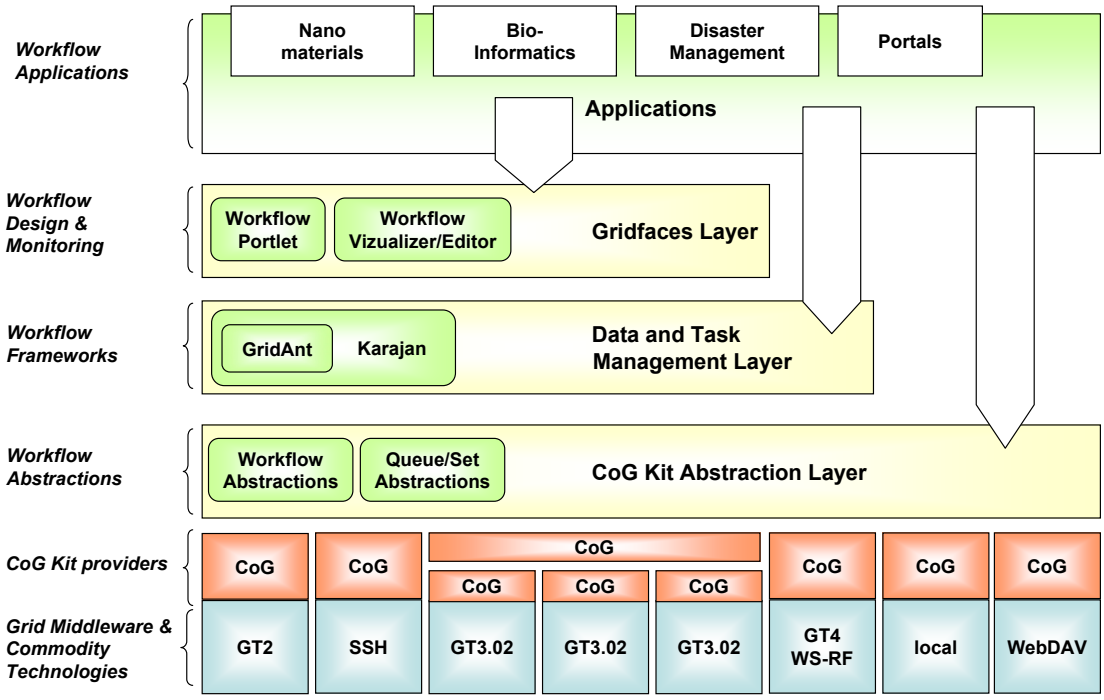

Figure 4: The layered approach of the Java CoG Kit provides mechanisms for incrementally enhancing workflow management components.

Our architecture has the characteristic of supporting multiple Grid middleware. This has the practical implication that workflows expressed with our model can adapt to (a) the evolving standards, (b) the evolving Grid middleware, (c) and the evolving deployment

of middleware once appropriate providers are available. Hence, it will allow us to enable switching between different Grid middleware in a running workflow.

4 Java CoG Kit Workflow Concepts for Programmers

We focus now our attention on a more detailed description of the concepts useful to developers. In general we provide support for workflows through

- the CoG Kit patterns and abstractions,
- the CoG Kit XML-based workflow engine (Karajan/Gridant), and
- the CoG Kit Gridshell.

Which of the solutions to chose depends on the user's requirements; see Table 2.

Table 2: Current features of the Java CoG Kit solutions.

	Abstraction API	Karajan	Gridant	GridShell
Java API	●	○	–	○
XML	●	●	●	–
If, While, For	●	●	–	○
Caching	–	●	–	–
Checkpointing	●	●	–	○
Logging	–	○	–	●
Viewer	○	●	–	●
Batch	●	●	–	●
Uses Karajan	–	●	–	–●
Uses Abstr. API	●	●	●	●

● = feature available,
○ = feature under development,
– = feature not available.
The CoG Kit is under development, and these features are subject to change.

4.1 Providers

We introduced the concept of Grid providers allows different Grid middleware to be integrated into the Java CoG Kit. Abstractions (defined in the subsequent sections) allow the developer to choose at runtime to which Grid middleware services tasks related to job submission and file transfer are submitted. This capability is enabled through customized dynamic class loading facilitating late binding against an existing production Grid.

Benefits. The benefits for Grid workflows are that workflow models can be designed that abstract the Grid infrastructure. The instantiation of all or part of the workflow can be done at runtime. It will be possible to design a workflow engine as part of the workflow system that can conduct the tasks associated with late binding including locating optimized resources and their reservation.

4.2 Pattern

The CoG submission pattern provides a convenient methodology for developing event-based Grid workflow tasks in Java. First, one needs to specify a job or task as part of the creation method. Second, the job or task needs to be submitted. Third, the job or task needs to react upon status changes. Fourth, we have a simple object class that can be used to instantiate such an object. In Figure 5 we list a pseudo code for creating such a pattern. Please note that the semantics of this pattern does not make any assumptions about the sophistication of the create, submit, or status changed method. Hence, the CoG pattern can be used to integrate advanced prediction or task-to-resource mapping strategies as part of the implementation.

```
public class COG implements StatusListener {
   public void create()  { ... }
   public void submit () { ... }
   public void statusChanged (StatusEvent e) { ... }
   public static void main (String arg[]){
     try {
       COG cog = new COG();
       cog.create();
       cog.submit();
     catch (Exception e) {
       logger.error("Error:", e);
   }
}
```

Figure 5: The CoG pattern assists users in developing simple event-based Grid applications.

Benefits. The existence of such a pattern allows the integration of a various similar structured solutions into a workflow management system. The implementation of such tasks may differ, but the way we access them is the same. Hence, the software engineering effort to integrate custom-designed tasks is lowered. However, most developers will want to use our abstractions and higher-level abstractions.

4.3 Abstractions

We have identified a number of useful basic and advanced abstractions that help in the development of Grid workflows. These abstractions include job executions, file transfers, workflow abstractions, and 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.

In Figure 6 we show how easy it is to use the abstractions to define a task, the elementary unit of our workflows. Here our task has an associated executable with options and a standard output associated with it. To coordinate the execution order tasks, one can use our simple interface to create dependencies between tasks, as shown in Figure 7. To map the tasks onto a Grid service, one can use service and security abstractions, as shown in Figure 8. The service abstractions are used to specify the appropriate service and establish a contact to it, while the security abstractions are used to authenticate with the service. Status updates can be programmed through the use of customized handlers, as shown in Figure 9. A handler is typically used as part of the submit method in the CoG pattern.

Benefit. Our abstractions allow the definition of direct acyclic workflow graphs and provide a simple, but sophisticated, programming model for defining workflows based on task dependencies.

4.4 Data structures

Based on the simple graph workflow abstraction, we are developing a number of useful abstractions around data structures that users may desire. Such data structures include set of tasks, queues, parameter studies, and hierarchical graphs.

Benefit. The benefit to the workflow developer is the availability of simple solutions that may address the users' requirements. In addition, these solutions can be viewed as pattern for customized solutions.

4.5 Gridfaces

An additional abstraction we have introduced into the Java CoG Kit is the concept of *Gridfaces*. Gridfaces are abstractions for visual components that have similar functionality defined through a uniform programming interface but that may be rendered in different visualization engines. We can define Gridfaces to browse a remote Grid directory for a stand-alone application or a Grid portal. The abstraction for such a browser contains three areas: the location of the file, a directory tree, and the status of accessing the remote location. Defining Gridfaces allows the application developer or portal developer to simplify the long term development of reusable graphical components, while relying on the common model-view-controller pattern to separate the model functionality from the view and the controls. Hence, adaptations to other visual frameworks become easier while reusing logic that is unrelated to the visual framework. Special Gridfaces for the display of graphs and their monitoring need to be developed.

Benefits. Providing similar-looking visual components for a variety of use cases, including portals and stand-alone applications, makes it possible to address the workflow user community requirements. While portals provide advantages in deployment and ease of use through a familiar paradigm for the novice, stand-alone applications with more sophisticated interfaces, including easy integration of the users' desktop machine capabilities, will fulfill the expert users' requirements. The deployment support that is provided by the latter strategy can be augmented through Java Webstart to conduct updates on the fly.

```
Task task1 = new Task();
JobSpecification spec = new JobSpecificationImpl();
spec.setExecutable("/share/bin/scheduleExperiment");
spec.addArguments("-date 30 August 2006
                   "-owner cn=Gregor von Laszwski");
spec.setStdOutput("confirmation.txt");
task1.setSpecification(spec);
```

Figure 6: Example of using the task abstraction for pseudo code to conduct an experiment reservation

```
TaskGraph tg = nhew TaskGraphImpl();

public void create () {
  // specify tasks
  .....
  /* Add the tasks to the TaskGraph
  tg.add(task1);
  tg.add(task2);
  tg.add(task3);
  tg.add(task4);
  tg.addDependency(task1, task2);
  tg.addDependency(task1, task3);
  tg.addDependency(task2, task4);
  tg.addDependency(task3, task4);
}
```

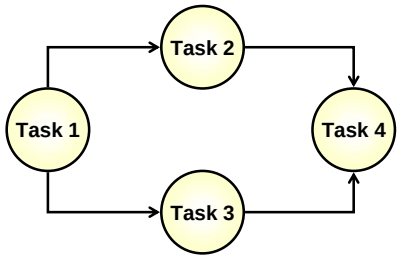

Figure 7: The CoG pattern assists users in creating dependencies between tasks.

```
Service service = new ServiceImpl(Service.JOB_SUBMISSION);

service.setProvider(‘‘GT3_2_1’’);

// Set Security Context - e.g. certificates and such
SecurityContext securityContext
   = CoreFactory.newSecurityContext(‘‘GT3_2_1’’);
securityContext.setCredentials(null);
service.setSecurityContext(securityContext);

// Set Contact - e.g. where to go to
ServiceContact serviceContact =
    new ServiceContactImpl(
      http://hot.mcs.anl.gov:8080/ogsa/services/base/gram/,
      MasterForkManagedJobFactoryService");
service.setServiceContact(serviceContact);

task1.setService(Service.JOB_SUBMISSION_SERVICE, service);
```

Figure 8: The CoG pattern assists users in mapping tasks onto a Grid service.

```
public void submit() {
  TaskGraphHandler handler =  new TaskGraphHandlerImpl();
      try {
        handler.submit(tg);
      } catch (Exception e) {
        logger.error(``Some Error occured'', e);
        System.exit(1);
      }
}
```

Figure 9: The CoG pattern assists users in obtaining status updates in categorized handlers.

5 Java CoG Kit Workflow Concepts for Component Developers

So far we have targeted the programmer who builds Java-based solutions for experiment management. Next, we focus on developers who can specify workflows as part of a simple XML workflow language interpreted by a workflow engine that is part of our workflow management system. The workflow engine interfaces capabilities provided by the different layers within our architecture. Hence, it hides much of the internal designs of the Java CoG kit workflow solutions by projecting to the component developer an integrated set of tools that can be accessed through convenient specification (see Figure 10). Additionally, programmers can integrate the capabilities of the workflow engine into their frameworks through abstractions and APIs.

5.1 Gridant

Gridand was conceived by the author in 2002 as a convenient and quick way to provide a workflow system and engine for Grid computing based on an XML language. At that time, the development of systems of BPEL4WS was still in its infancy. As ant is used by millions of users a day and provided the de-facto standard for coordinating build processes in Java it was convenient to integrate extensions to ant that call the Java CoG Kit. The system was specified in [27], and a small subset of the features was implemented [1]. During the implementation phase it quickly became obvious that although Gridant fulfilled its promises, it did have limitations based on the scalability of ant. Although we still provide the Gridant elements as part of the Java CoG Kit, we have replaced the engine as described in the next section.

Benefit. Gridant is a reliable tool that can be easily extended through accessing the Java CoG Kit task abstractions. It is best suited for small workflows. Conditions and iterations are more difficult to design.

5.2 Karajan

Today's Java CoG Kit workflow engine is based on a derivative of GridAnt and ant. It is called *Karajan*, after a name of a conductor of the Berlin Symphony to reflect the act of coordination in its name.

In our workflow engine, we added support for flow control constructs such as conditions

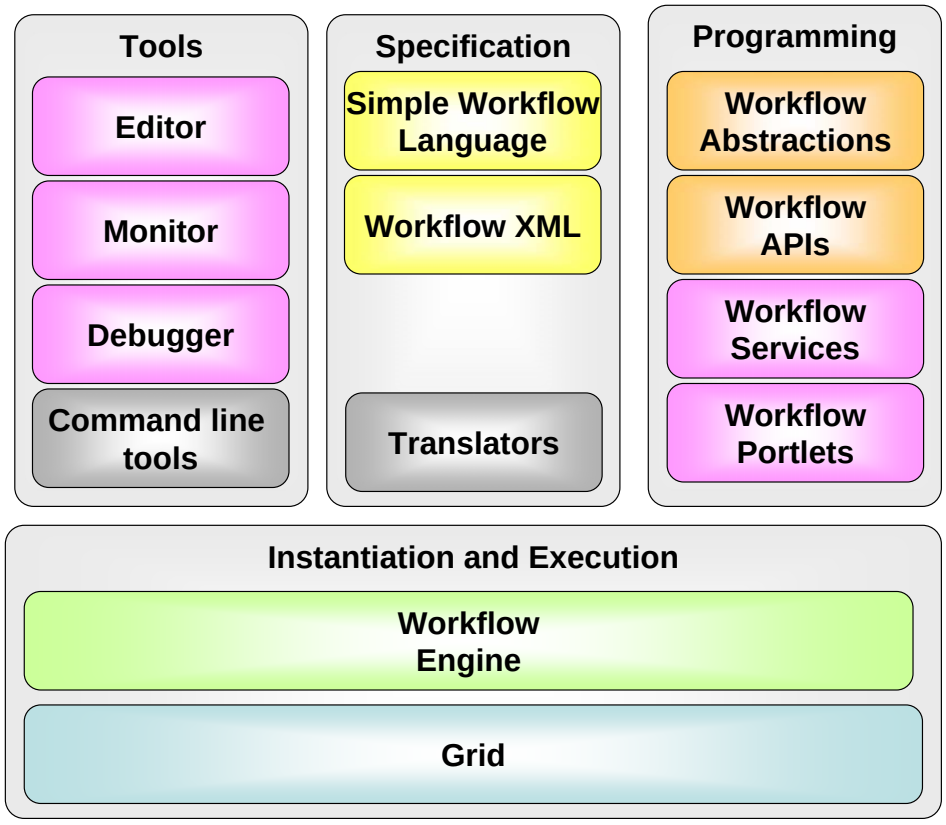

Figure 10: Components of the Java CoG Kit workflow management system.

and loops. We also have reimplemented features that traditional ant provides, such as sequential and parallel blocks. Figure 11 depicts a number of the workflow patterns supported within the Java CoG Kit workflow engine. Specialized Grid-enabled tasks for job submission file transfer and authentication are available that can be augmented with appropriate providers.

Next, we highlight a subset of examples that illustrate some of the capabilities of Krajans workflow engine and its specification language.

5.2.1 Modularization

To build components for our workflow system, we are using concepts of projects, elements, and include files as is used in Java ant.

Projects. The concept of named projects allows us to refer to different workflow projects by name.

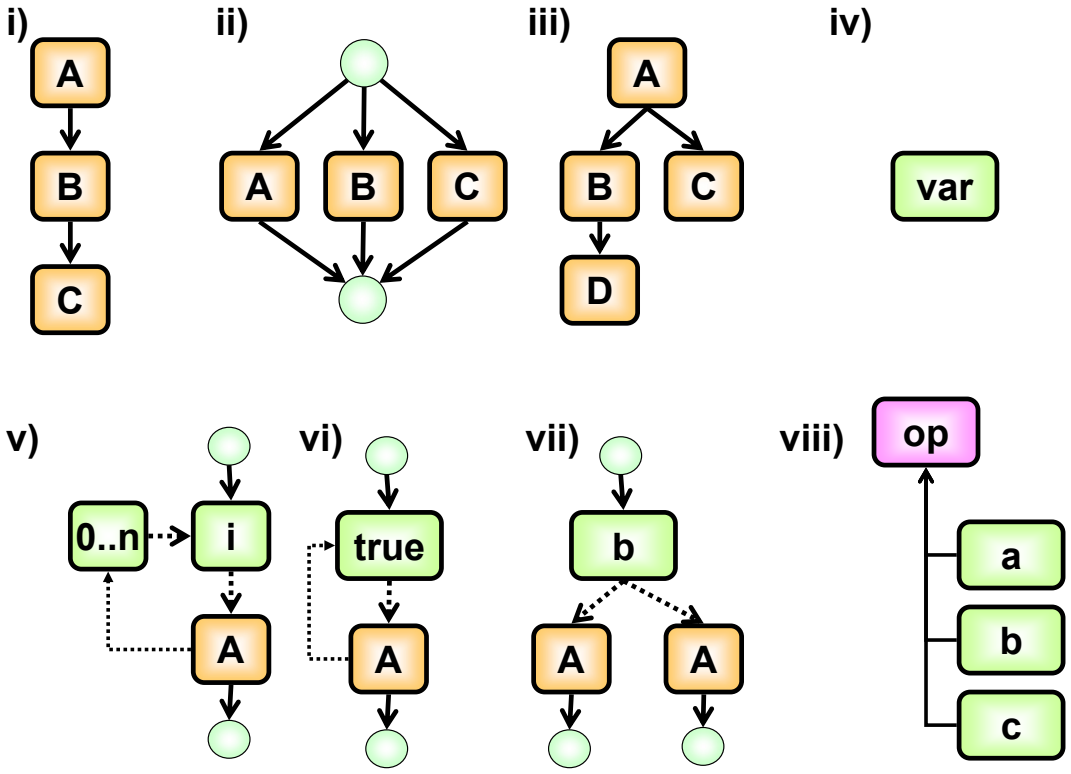

Figure 11: Selected workflow patterns that are supported by the Java CoG Kit: (i) sequence, (ii) parallelism, (iii) fork, (iv) variables, (v) for loop, (vi) choice, (viii) operators

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

Elements. The concept of elements is one of the most important features of our workflow language as it 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="printExperimentFiles"
         arguments="input,output">
   <echo message="Input = {input}"/>
   <echo message="Output = {input}"/>
</element>
```

Once we have defined a new element, we can invoke it in the following way.

```
<printExperimentFiles input="input.dat" output="output.dat">
```

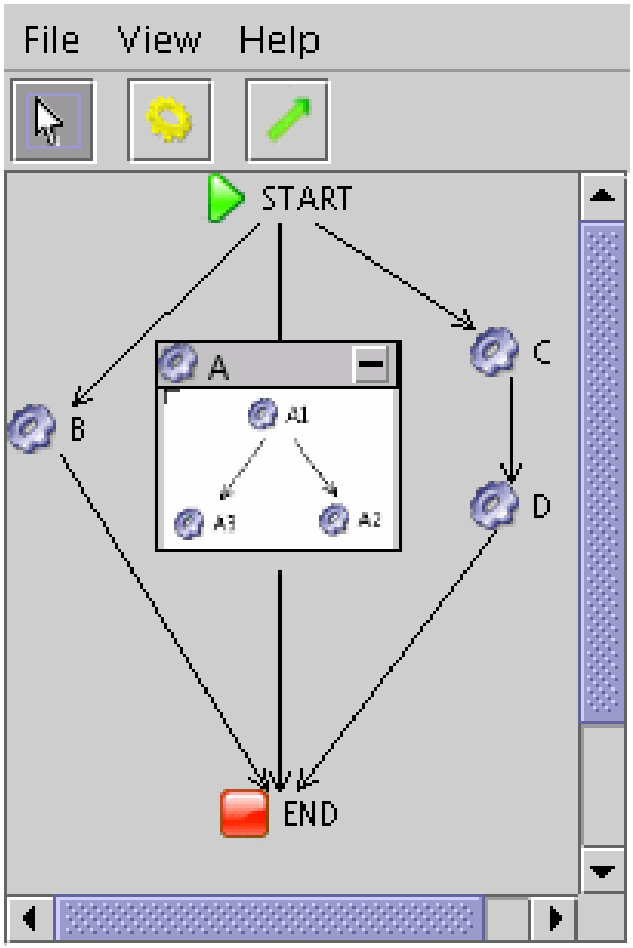

Figure 12: The Java CoG Kit is planned to handle hierarchical graphs.

Descriptions and Annotations. With the description concept, we can use an annotation and description to provide information about the use of an element. The description should include a detailed textual explanation of what this element is for. In experiments, the annotation can be used in visualization tools and may include a label for the element 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"/>
```

Namespaces. Through the concept of namespaces we can define elements that are augmented automatically with a prefix. This enables multiple workflow developers to develop components in parallel while using predefined namespaces. It also allows one to conveniently

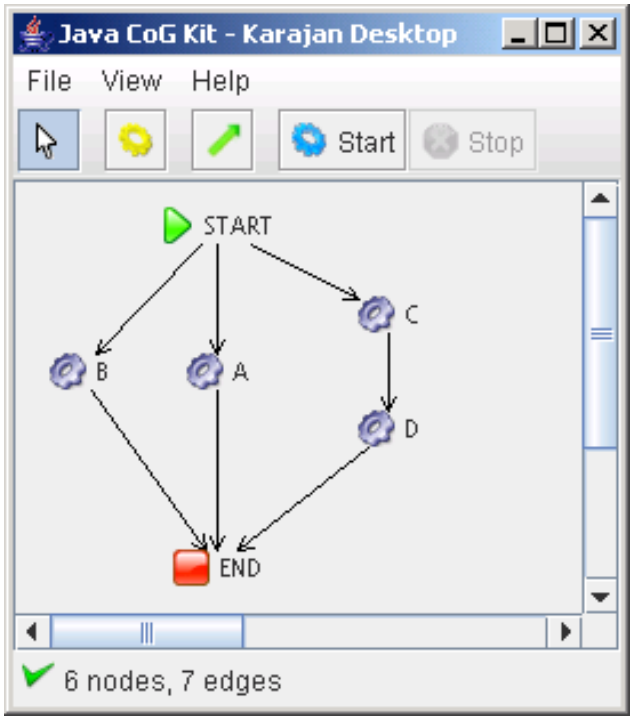

Figure 13: Parallel and sequential constructs simplify the definition of graphs.

rename a number of elements in a file in order to avoid name clashes. 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. The include concept allows us to structure element definitions and other Java CoG Kit workflow scripts in separate files and directories in order to organize them appropriately. The concept is similar to the include pragma in the C programming language. Hence, files can be included in a workflow as follows:

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

Repository. Key to the shared workflow management system is a convenient service to store, retrieve, and modify workflow components defined by the community. At present, we are defining a simple extensible framework to design, build, and deploy such a workflow repository service. A repository is intended to be used in ad hoc Grids or in community Grids. More information about our current repository design can be found in [32]. The information included in the repository is formulated as XML metadata that can be searched conveniently.

5.2.2 Variables and Operators

The concepts of variables and operators are used to store states as part of dynamically executing workflows.

Variables. Our concept of variables is untyped, similar to that of python. Its values are interpreted in the context of other workflow expressions.

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

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

Lists, Ranges, and Maps. We have also introduced additional concepts that are well-known data structures and make the programming of advanced workflows easier. These concepts include lists, ranges, and maps (or hash tables). Such data structures are important to easily formulate parameter studies and other repetitive tasks as part of the experiment management. For more information on these data structures, see [10].

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

Operators. We have defined a number of standard operators that allow the integration within conditional statements. Our operators include mathematical and Boolean operators such as sum, product, equal, and not. An example of 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>
```

5.2.3 Workflow Structural Elements

Since we have to adapt to various conditions within the experiment workflow, it is necessary to define a conditional statement and loops. Hence, our workflow engine supports not only hierarchical graphs but also conditions, choices, loops, and even recursion.

Condition. Our workflow language contains an if construct. The syntax is defined by an if tag immediately followed by a Boolean expression that determines the condition. The statements to be executed are enclosed in then or else tags.

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

Choice. The concept of a choice will execute child elements in succession until one completes without error. A choice has transactional behavior and buffers its return values to allow us to more easily parallelize a choice, as demonstrated in a forthcoming section. In our example we simply print two messages; however, since the statements in the first sequential block do not return an error, the second block will not be executed.

```
<choice>
  <sequential>
    <print> Starting Experiment A </print>
    <generateError message="Error"/>
  </sequential>
  <sequential>
    <print>Starting Experiment B</print>
  </sequential>
</choice>
```

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

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

5.2.4 Parallelism

Our workflow language contains multiple constructs to define parallelism. The simplest concept is taken from ant 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 parallel.

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

Scope Variables are scoped based on their position within sequential and parallel blocks. This concept is especially useful if workflows are enhanced through the include statement in various blocks. It keeps the code short and avoids side effects. The following example demonstrates that the value of the variable is dependent on its scope.

```
<set name="a" value="Start"/>
<parallel>
  <sequential>
    <set name="a" value="Experiment A"/>
    <echo>a is {a}</echo> <! prints Experiment A -->
  </sequential>
  <sequential>
    <set name="a" value="Experiment B"/>
    <echo>a is {a}</echo> <! prints Experiment B -->
  </sequential>
</parallel>
<echo>a is {a}</echo> <!prints Start -->
```

Parallel Loops. Many experiments, parameter studies occur that do not introduce side effects. Therefore, it is important to support parallel loops. A loop can be augmented with a mode that describes the way the statements in the loop are executed. If it is set to parallel, it executes the statements in parallel.[1]

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

Parallel Choice. We have also designed the feature of a parallel choice that is specified through the mode argument.[2] The parallel choice can be important 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>Experiment A</print>
  </sequential>
  <sequential>
    <wait delay="50"/>
    <print>Experiment B</print>
   </sequential>
</parallelChoice>
```

Grid Tasks. Naturally, a Grid workflow system must access Grid functionality. To support this capability, we have developed a number of elementary Grid tasks such as execution and file transfer. The functionality is provided already by our Grid abstractions, and the workflow system uses the abstractions to access the Grid internally. This also allows us to support different providers as long as they have been implemented and a provider is available.

We distinguish the tasks grid:execute, which executes a program on local or remote grid resources; grid:transfer, which transfers files between resources; and grid:authentication, which authenticates to the Grid.

It is possible to define a provider that determines the protocol and mechanism used to execute the appropriate Grid task. The elegance of our concept is that the developers do not have to deal with the internal workings of the Grid services. Developers need to know only which version of grid middleware they run on their Grids. The rest is provided by the Java CoG Kit. This approach significantly simplifies the use of Grids as we not only abstract above the Grid middleware but also enhance it by providing a sophisticated workflow framework. Furthermore it would be simple to integrate metaschedulers into the framework to provide

[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.

even more simplifications for the users and hide events the bookkeeping associated with a particular Grid middleware provider.

The following example illustrates the simplicity of our concept. The example authenticates to a Grid, copies a program in source form to a remote machine, and compiles the 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="experiment.java" />
  <task:Execute host="{host}" provider="gt2"
    executable="{path}/bin/javac"
    arguments="experiemnt.java"/>
</project>
```

5.2.5 Error Handling

The concept of error handling allows us to integrate strategies for errors and exceptions into the workflow. Through the definition of an *onError* element, we include the ability to catch errors and react to them. A match argument specifies a regular expression that, when true, triggers the execution of a block enclosed in the onError element. Our example shows how to start a graphical user interface for entering the credentials in case they have expired or cannot 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. The concept of checkpointing enables us to store intermediate states of the workflow executions in order to roll back to it in case a problem occurs later in the workflow execution. Additionally, breakpoints can be set to provide debugging support for interactive running workflows.

5.2.6 Forms

As we work often with the input of interactive data, our workflow system enables the concept to define forms as part of workflow tasks. We have defined a simple form language that allows us to define graphical user interfaces. Figure 14 shows an example that is created with the following code segment.

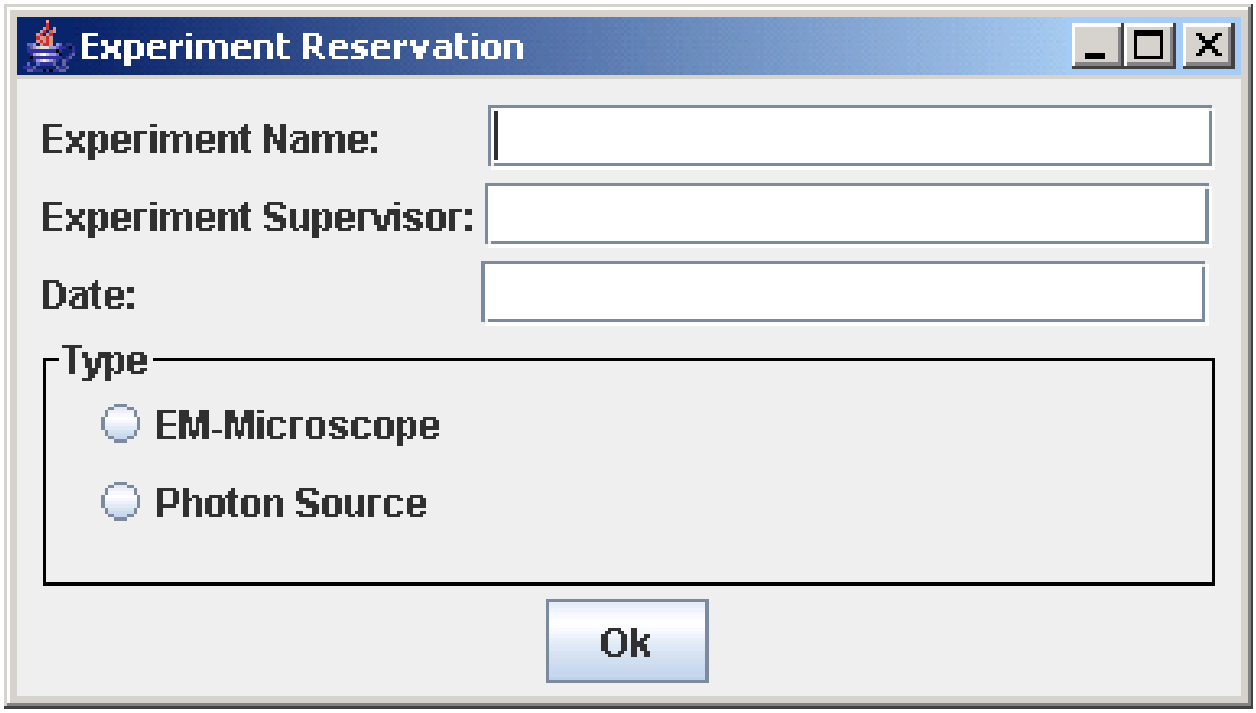

Figure 14: Example for a form created on the fly with the workflow form specification elements.

```
<set name="formData"
   annotation="Reservation form for an experiment at ANL">
   <form:form title="Experiment Reservation"
                id="form"
             waitOn="IDOk">
     <form:vbox>
         <form:vbox>
            <form:hbox>
             <form:label text="Experiment Name: "/>
             <form:textField id="IDexpname" columns="20"/>
           </form:hbox>
            <form:hbox>
             <form:label text="Experiment Supervisor: "/>
             <form:textField id="IDexpsupevisor" columns="20"/>
           </form:hbox>
           <form:hbox>
             <form:label text="Date: "/>
             <form:textField id="IDexpdate" columns="20"/>
           </form:hbox>
            <form:radioBox caption="Type" id="IDSex">
             <form:radioButton caption="EM-Microscope"
                                id="IDmicroscope"/>
             <form:radioButton caption="Photon Source"
```

```
                                    id="IDxray"/>
            </form:radioBox>
          </form:vbox>
        <form:button id="IDOk" caption="Ok"/>
      </form:vbox>
    </form:form>
 </set>
```

5.2.7 Java and Python Language Support

Since many experiments may require considerable customization, we have introduced into the workflow system several concepts of integrating with native applications. The easiest one is projected through the task:execute element. However, to extend our workflow language, we also have the ability to call Java methods directly. We show how easy it is to extend our language by adding a method that returns the current time in milliseconds by calling the appropriate system method in Java. First, we define an element called currentTime.

```
<include file="java.xml"/>
<element name="currentTime">
  <java:invokeMethod method="currentTimeMillis"
                     classname="java.lang.System"
                     static="true"/>
</element>
```

To demonstrate how to call it, we call it to set the value of a variable so we can access it at a later time.

```
<set name="time">
   <currentTime/>
</set>
```

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"/>
```

In principle, we can also integrate arbitrary Java code blocks with a `<java>` tag. However, we have not distributed this feature with our current version.

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

5.2.8 Performance Augmentations

As many experiments need to measure the time when certain actions have been conducted, we have included a simple timer. This also allows us to conduct elementary performance experiments and collect time information as part of parameter studies and the mapping of tasks to remote resources. Such data can be included in customized metaschedulers. To allow more than one timer, we have defined named timers that can be used as follows.

```
<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.

To simplify the execution of real-time tasks, which are important as part of experiment workflows, we intend to provide an extension to our tasks with an absolute time stamp.

```
<task:realtime name="experiment" date=5:20:12pm 06/01/2005>
  ...
</task:realtime>
```

Element Result Caching. As some experiments may be rather costly and complicated to execute, we have included the concept of result caching into our framework. It allows us to reuse elements without executing them once they have already been evaluated. This feature can be switched on; by default it is disabled.

5.2.9 Syntax Translators

We have designed our workflow language around the concepts of 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. To prove this point, we have developed a simplistic language that can be translated into our XML specification and can be directly executed by our workflow engine. In our simplistic language, we simply replace the beginning and end tags with operators that use beginning and closing braces $op(...)$ instead of $< op > ... < /op >$. It is feasible that other source-to-source translators could be developed, thus enabling other frameworks to make use of our workflow engine.

6 Related Research

Much related research in the area of workflow and Grid workflow exists; se for example [2] [19, 17] [16] [13, 11] [7, 6, 15] [20] [5] [21] [3] [18] [12] [9].

However, our approach is not a duplication of other approaches but provides an integrated approach of exposing workflow to the Grid community. It focuses on the definition of a powerful language and the implementation of simple but useful tools to manipulate this language.

7 Conclusion

This paper provides a detailed overview about the Java CoG Kit workflow system that is useful for scientific experiments. The value of the Java CoG Kit workflow solution lies in its simplicity and its ability to be integrated in a solution that can expose workflow to a variety of users. 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. Based on our experience with the Java CoG Kit, we concluded 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 care to project a future-oriented architecture that allows the gradual enhancement while addressing explicit problems with today's and tomorrow's Grid middleware. Examples of such issues are presented in the paper 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 includes not only the availability of a sophisticated language but also the development of more sophisticated tools that make the manipulation and handling of the workflows easier. Our future goal is to develop additional tools that can be exposed either as stand-alone applications or as part of a portal through portlets developed together with the Open Grid Computing Environments project [14].

Availability

The Java CoG Kit 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 NMI. I like to especialy acknowledge Mike Hategan for his role in the design and implementation of the Karajan workflow engine, Kaizar Amin for his role in the design of the Java CoG Kit abstractions, and Jarek Gawor for his role in the design and implementation of much of the jglobus library.

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] Tony Andrews, Francisco Curbera, Yaron Gol Hitesh Dholakia, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte (Editor), Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services. [BPEL4WS.], May 2003. Available from: `http://xml.coverpages.org/BPELv11-May052003Final.pdf`.

[3] H. P. Bivens. Grid WorkFlow: Grid Computing Environments Working Group Document. `http://www.gridforum.org`, 2001.

[4] The Java CoG Kit Source Code. CVS Repository. Available from: `cvs.cogkit.org`.

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

[6] 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`.

[7] 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`.

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

[9] Andreas Hoheisel and Uwe Der. An XML-Based Framework for Loosely Coupled Applications on Grid Environments. In *ICCS 2003*, volume 2657 of *LNCS*, pages 245–254. Springer, 2003. Available from: `http://www.andreas-hoheisel.de/docs/Hoheisel_and_Der_2003_ICCS.pdf`.

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

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

[12] B. Kiepuszewski1, A.H.M. ter Hofstede1, and W.M.P. van der Aalst. Fundamentals of Control Flow in Workflows. Technical report, Centre for Information Technology Innovation, Queensland University of Technology, November 2003. Available from: `http://tmitwww.tm.tue.nl/research/patterns/download/qut_expr_rep.pdf`.

[13] B. Ludaescher, A. Gupta, and M. E. Martone. *Bioinformatics: Managing Scientific Data*, chapter A Model-Based Mediator System for Scientific Data Management. Morgan Kaufmann, 2003. Available from: `http://citeseer.nj.nec.com/cache/papers/cs/27492/http:zSzzSzwww.sdsc.eduzSz~guptazSzpublicationszSzmbm-chapter-rev.pdf/a-model-based-mediator.pdf`.

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

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

[16] The Taverna Project. Web Page, December 2003. Available from: `http://taverna.sourceforge.net`.

[17] Ian Taylor, Shalil Majithia, Matthew Shields, and Ian Wang. Triana workflow specification. Technical report, GridLab.

[18] Teuta. Web Page. Available from: `http://dps.uibk.ac.at/projects/prophet/node4.html`.

[19] Triana Workflow. Web Page. Available from: `http://www.triana.co.uk`.

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

[21] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. Paper, Department of Mathematics and Computing Science, Eindhoven University of Technology, NL-5600 MB, Eindhoven, The Netherlands, 1998. Available from: `http://is.tm.tue.nl/staff/wvdaalst/publications/p53.pdf`.

[22] Gregor von Laszewski. *A Parallel Data Assimilation System and Its Implications on a Metacomputing Environment*. PhD thesis, Syracuse University, December 1996.

[23] Gregor von Laszewski. An Interactive Parallel Programming Environment Applied in Atmospheric Science. In G.-R. Hoffman and N. Kreitz, editors, *Making Its Mark, Proceedings of the 6th Workshop on the Use of Parallel Processors in Meteorology*, pages 311–325, Reading, UK, 2-6 December 1996. European Centre for Medium Weather Forecast, World Scientific. Available from: `http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--ecwmf-interactive.pdf`.

[24] 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`.

[25] Gregor von Laszewski. The Grid-Idea and Its Evolution. *to be published.*, 2005. Argonne National Laboratory, Argonne, IL 60439, U.S.A. Available from: `http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-grid-idea.pdf`.

[26] 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`.

[27] Gregor von Laszewski, Kaizar Amin, Shawn Hampton, and Sandeep Nijsure. GridAnt – White Paper. Technical report, Argonne National Laboratory, 31 July 2002. Available from: `http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-gridant.pdf`.

[28] Gregor von Laszewski, Steve Fitzgerald, Ian Foster, Carl Kesselman, Warren Smith, and Steve Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, pages 365–375, Portland, OR, 5-8 August 1997. Available from: `http://www.mcs.anl.gov/~gregor/papers/fitzgerald--hpdc97-mds.pdf`.

[29] Gregor von Laszewski, Ian Foster, Jarek Gawor, Warren Smith, and Steve Tuecke. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. In *ACM Java Grande 2000 Conference*, pages 97–106, San Francisco, CA, 3-5 June 2000. Available from: `http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--cog-final.pdf`.

[30] Gregor von Laszewski, Jarek Gawor, Sriram Krishnan, and Keith Jackson. *Grid Computing: Making the Global Infrastructure a Reality*, chapter Commodity Grid Kits - Middleware for Building Grid Computing Environments, pages 639–656. Communications Networking and Distributed Systems. Wiley, 2003. Available from: `http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--grid2002book.pdf`.

[31] Gregor von Laszewski and Mike Hategan. Grid Workflow - An Integrated Approach. In *To be published*, Argonne National Laboratory, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60440, 2005. Available from: `http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-workflow-draft.pdf`.

[32] Gregor von Laszewski and Deepti Kodeboyina. A Repository Service for Grid Workflow Components. In *International Conference on Autonomic and Autonomous Systems International Conference on Networking and Services*. IEEE, 23-28 October 2005. Available from: `http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-workflow-repository.pdf`.

[33] 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`.

[34] Gregor von Laszewski, Mei-Hui Su, Joseph A. Insley, Ian Foster, John Bresnahan, Carl Kesselman, Marcus Thiebaux, Mark L. Rivers, Steve Wang, Brian Tieman, and Ian McNulty. Real-Time Analysis,

Visualization, and Steering of Microtomography Experiments at Photon Sources. In *Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, TX, 22-24 March 1999. Available from: `http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--siamCmt99.pdf`.

[35] 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`.

[36] Patrick Wagstrom, Sriram Krishnan, and Gregor von Laszewski. GSFL: A Workflow Framework for Grid Services. In *SC'2002*, Baltimore, MD, 11-16 November 2002. (Poster). Available from: `http://www.mcs.anl.gov/~gregor/papers/gsfl-paper.pdf`.

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