GridAnt: A Client-Controllable Grid Workflow System

Gregor von Laszewski,[1,*] Kaizar Amin,[1,2] Mihael Hategan,[1]
Nestor J. Zaluzec[1] Shawn Hampton,[3] Albert Rossi,[3]
[1]Argonne National Laboratory, Argonne, IL, USA.
[2]University of North Texas, Denton, TX, USA.
[3]NCSA, University of Illinois at Urbana-Champaign, Champaign, IL, USA.

Argonne National Laboratory Preprint ANL/MCS-P1098-1003
also in 37th Hawai'i International Conference on System Science, Jan 5-8, 2004, Island of Hawaii, Big Island,
http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--gridant-hics.pdf

Abstract

Process management is an extremely important concept in both business and scientific communities. Several workflow management tools have been proposed in recent years offering advanced functionality in various domains. In the business world, workflow vendors offer commercial and customized solutions targeting specific users. In the scientific world, several open-source workflow management tools are freely available. However, they are directed toward service aggregation rather than distributed process management. Little consideration is given to the needs of the client in terms of mapping the process flow of the client. In the Grid community it is essential that the Grid users have such a tool available enabling them to orchestrate complex workflows on the fly without substantial help from the service providers. At the same time it is important that the Grid user not be burdened with the intricacies of the workflow system. With the perspective of the Grid user in mind, an extensible client-side workflow management system, called GridAnt, *has been developed. This paper discusses the design principles, functionality, and application of the proposed GridAnt workflow manager.*

1 Introduction

The term *workflow* can be defined as the orchestration of a set of activities to accomplish a larger and sophisticated goal, referred to as a *business process* [1]. Significant research has been conducted in recent years to automate these activities using advanced workflow management tools. Some of the most popular and sophisticated workflow systems available in market include Staffware [2], COSA [3], Inconcert [4], Eastman Software [5], Domino Workflow [6], Websphere MQ Workflow [7], Visual Workflo [8], and I-Flow [9]. These products offer extensive functionality and support a variety of workflow patterns [10]. However, all of them are propriety softwares and commercially motivated, making it infeasible to use them for open-source research projects. Although a few general-purpose open-source workflow toolkits [11] are available for use, they are platform-dependent and offer limited functionality. While analyzing some of the flow patterns in correlation with hardware computing patterns we notice an increased complexity and sophistication of the flowpatterns that enable collaborative workflow management. In future we expect that that this will even include the derivation of knowledge flow patterns.

Much of the work reported here has been heavily influenced by the authors' experience with scientific workflows [12, 13, 14, 15]. The concepts of workflow and process orchestration are extremely important in the context of Grid computing. Grid computing focuses on secure and collaborative resource sharing across multiple, geographically distributed institutions. In the Grid environment, workflow management becomes more complex because interprocess communications span institutional boundaries, requiring support for multiple security policies. A large number of orchestrating tools have been proposed and used by the Grid community. Webflow [16] and Triana [17] offer a visual programming model for dynamic orchestration of high-performance computing applications from a group of predefined commodity software modules. Symphony [18] provides an abstract code composition and manipulation framework for Grid metacomputing systems in

*corresponding author: gregor@mcs.anl.gov

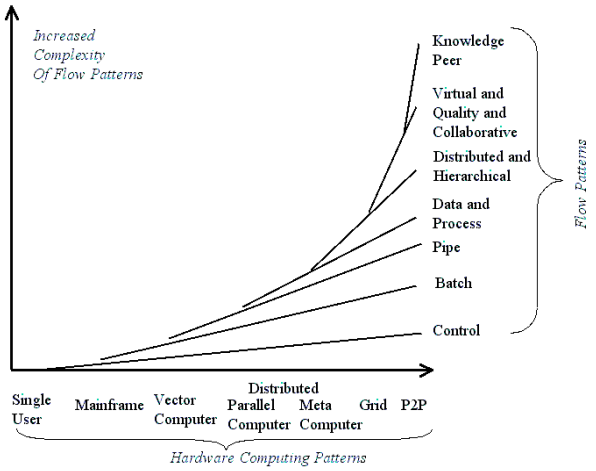

Figure 1. Computing flow patterns are evolving from simple control flows to sophisticated knowledge flow patterns.

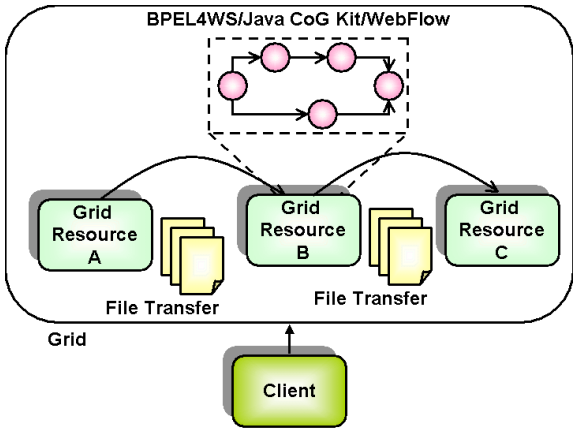

Figure 2. Use Case for Client-Controllable Workflow

a secure environment. The XCAT [19] framework, based on the Common Component Architecture (CCA) [20], provides mechanisms to formulate reusable distributed software components to be used in CCA-compliant runtime frameworks. The GridFlow [21] system provides a flexible agent-based mechanism for dynamic scheduling of Grid jobs within a global Grid workflow service. Ptolemy II [22] is a Java-based component assembly architecture with a rich visual interface. LabVIEW [23] offers a powerful scientific environment for data acquisition, analysis, and presentation. With the shift in the Grid architecture toward a service-oriented framework [24], several new specifications such as Business Process Execution Language for Web Services (BPEL4WS) [25] and Grid Services Flow Language (GSFL) [26] are being researched in the Grid context.

All of the Grid-enabled frameworks described above focus on the aggregation of activities into a unified, robust, and flexible backend functionality. The workflow is designed and implemented by experts and administrators providing the consolidated application to the end users, but it cannot be controlled by the end users. There is no convenient tool available to the Grid community that allows the user to express and control the execution sequence without having any expertise in sophisticated workflow systems or control of the backend functionality.

For example, consider the simplistic, yet very common, scenario shown in Figure 2. A Grid user needs to run an experiment on a Grid resource *B*. To do so, the user needs to transfer a set of files from resource *A*; and the output produced at resource B needs to be stored on resource *C*. Independent of the mechanism by which the execution framework at resource *B* is developed, the Grid user needs a workflow system to automate such a flow of *control* and *data*. Currently, such a flow is managed explicitly under human control or through shell scripts. Explicit human control gets unwieldy once the level of complexity of such flows increases. On the other hand, shell scripts are platform-dependent and do not provide sufficient means for data communication between dependent activities. Little research has been done in the Grid community to provide with a platform-independent, robust, and convenient mechanism for client-controllable workflow systems. The Condor DAGMan meta-scheduler [27] allows for the expression of dependencies of Condor jobs. However, it functions only in the context of Condor jobs, and its applicability in the domain of generic Grid tasks needs to be researched. The Grid Workflow specification document [28] provides with a flexible vocabulary for orchestrating user-controlled experimental runs. Nevertheless, it requires a development of a workflow engine whose functionality needs to be thoroughly tested within the community. Motivated by the need to develop a simple, extensible, platform-independent, and client-controllable workflow mechanism, we present the *GridAnt* [29] system.

The rest of this paper is structured as follows. Section 2 outlines the details of the GridAnt workflow system. It discusses the workflow engine, the GridAnt runtime environment, the Grid-centric workflow vocabulary, and the GridAnt monitoring tool. Section 3 discusses the application of the GridAnt framework to an advanced scientific application. Section 4 gives an overview of the limitations of the basic GridAnt mechanisms and proposes a service-oriented solution to overcome it. Section 5 summarizes the paper and outlines future research.

2 GridAnt

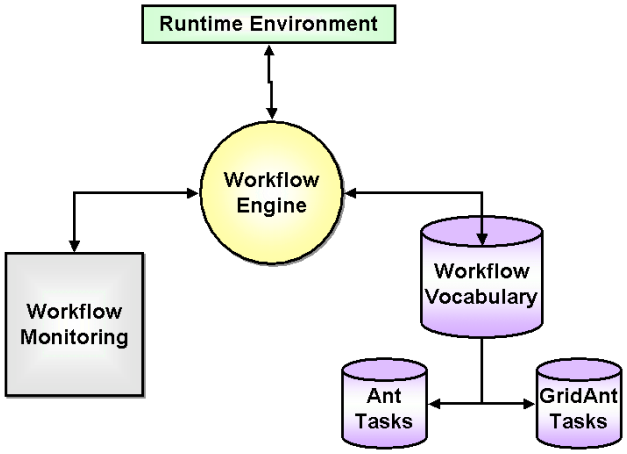

Figure 3. GridAnt Workflow System

The proposed GridAnt architecture consists of the following major components (see Figure 3):

- Workflow engine
- Runtime environment
- Workflow vocabulary
- Workflow monitoring

2.1 Workflow Engine

At the heart of the GridAnt system is its workflow engine. The engine is responsible for directing the flow of control and data through multiple GridAnt activities. It is the central controller that handles task dependencies, failure recoveries, performance analysis, and process synchronization. Rather than developing the the engine from scratch, we reuse an existing commodity tool called *Ant*[1] [31]. Ant is an extremely popular build tool in the Java community. It is open source, written in Java, and freely distributed by the Apache Software Foundation [32]. Traditionally Ant provides a flexible mechanism to express script dependencies in a project build process. In GridAnt, however, we use Ant to manage dependencies between different Grid-enabled tasks. Some of the most prominent reasons for using Ant as our workflow engine are as follows:

- Ant is written in Java, providing platform independence and immediate integration into the newly adopted Grid services framework.
- Ant tasks and dependencies can be conveniently expressed in XML.
- Ant is highly extensible, allowing us to incorporate Grid-specific functionality.
- It is highly modular and can be embedded into other Java applications.
- Ant distribution contains a large number of core and contributed tasks that provide extensive functionality similar to those available in shell scripts.
- Ant is a community project, thereby enjoying continuous use, testing, and improvement over years. Hence, it offers a reliable foundation for a sophisticated workflow system.

Although Ant provides an "out-of-the-box" solution for expressing task dependencies and directing the flow of control in simple workflows, it lacks several aspects in supporting sophisticated workflow requirements. Apart from expressing task dependencies, it also supports sequential and parallel execution containers that allow subtasks to be executed in sequence or in parallel, respectively. However, even the most simplest process orchestration would require functionality beyond that provided by Ant. To support arbitrary control flows through dependency graphs, one must support constructs that allow conditional execution, block iteration (looping), exception and error handling, and several other workflow patterns [10]. Further, Ant provides mechanisms only to direct the flow of control. It lacks the infrastructure to support workflow composition allowing the output of one activity to become the input to another. However, through additional components in the GridAnt system, we overcome this restriction, enabling GridAnt to support workflow orchestration and composition. Irrespective of its limitations, Ant offers a robust, extensible, community-supported solution that can be reused without any modification as the GridAnt engine.

2.2 Runtime Environment

As discussed in the preceding section, Ant lacks the functionality to support workflow compositions. This is certainly an unacceptable restriction for complex Grid workflow activities that require intertask communication. It is imperative to have a communication mechanism between individual Grid activities to enable the exchange of runtime data. A variety of communication models can be employed over Ant based on the level of flexibility, control, and efficiency desired. A simple solution could be to use the native file system for communication. Although it provides considerable flexibility, the overhead associated with such a

[1] Another option was the Jakarta Jelly [30] project. However, it is in its development phase and is not a mature product.

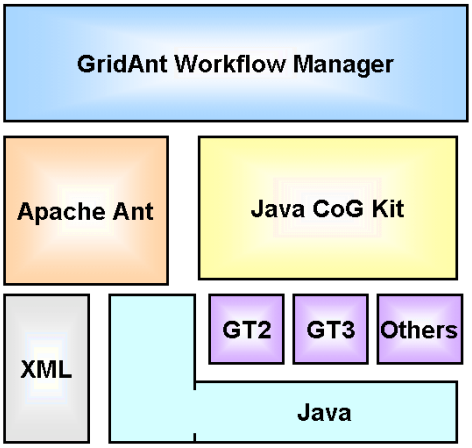

Figure 4. Layered Commodity Architecture

mechanism would render it unusable. Ant provides mechanisms to publish global user-defined properties that can be accessed by all the tasks. However, these properties are immutable and therefore significantly restrict the GridAnt communication model. Mechanisms are available that allow the generation of mutable internal properties in Ant that can be exploited for intertask communication. However, this requires modification to the Ant-core, which is undesirable. Further, Ant properties can take only string values, a restriction that imposes considerable performance overhead when runtime transformation between simple strings and complex data structures is involved.

In order to overcome the deficiencies of Ant in the context of an advanced workflow system, the GridAnt architecture introduces a runtime environment that offers a globally accessible whiteboard-style communication model. The runtime environment is capable of hosting arbitrary data structures that can be read and written by individual GridAnt tasks. The mechanism to identify the required information and process it eventually lies with the respective Grid tasks. Additionally, the runtime environment supports important constructs such as constants, arithmetic expressions, global variables, array references, and literals. Such artifacts elevate the functionality of the GridAnt system to support complex multimodal intertask communication similar to those supported in contemporary programming languages. Although the implementation specifics of the GridAnt runtime environment are beyond the scope of this paper, we mention that the runtime environment is deployed in the system without any modifications to the Ant-core. This allows the GridAnt system to be compatible with future releases of Apache Ant without any software engineering efforts.

2.3 Workflow Vocabulary

A workflow system must either identify the individual workflow activities at runtime [25] or predefine them at compile time. The GridAnt system works with a set of predefined activities or tasks[2], thereby establishing a workflow vocabulary against which complex workflows can be developed. The extensibility of Ant allows us to enhance the existing Ant functionality into the Grid domain. GridAnt vocabulary can be incrementally improved by contributions from both the Ant and GridAnt communities. We note that GridAnt tasks can be categorized into two domains, Ant-compatible and Ant-incompatible tasks. The Ant-compatible tasks include those tasks that exclusively rely on the functionality offered by Ant, thereby making them compatible with any distribution of Ant. The Ant-incompatible tasks include those that rely on the extended GridAnt runtime environment to achieve its functionality. These tasks cannot run with the basic installation of Ant and need GridAnt at all times.

An extensive description of all the tasks developed as a part of the GridAnt system is beyond the scope of this paper. However, we briefly outline a partial list of the most important tasks thatt we defined by the GridAnt framework and we propose to implement.

- *grid-setup*: Establishes the Grid environment based on client preferences. Specifically, it sets the client machine to use the Java CoG Kit [33], which forms the basis of our Grid interactions.

- *grid-authenticate*: Initializes the system for subsequent authentication as part of a single sign on process by the user to the Grid. In case of a Globus hosting environment, a security proxy is created that has limited lifetime and is reused on behalf of the user to authenticate to the system. The authentication mechanism is dependent on the hosting environment. All Grid tasks need this authentication preparation step.

- *grid-execute*: Executes a Grid task on the remote Grid resource.

- *grid-copy*: Copies a file from one Grid resource to another. Third-party transfers are supported through the use of the GridFTP protocol.

- *grid-cancel*: Cancels the execution of the given job on the remote machine.

- *grid-query*: Queries the Grid information services to check the availability and capability of a Grid resource.

[2]Following the Ant terminology, we refer to individual workflow activities as tasks.

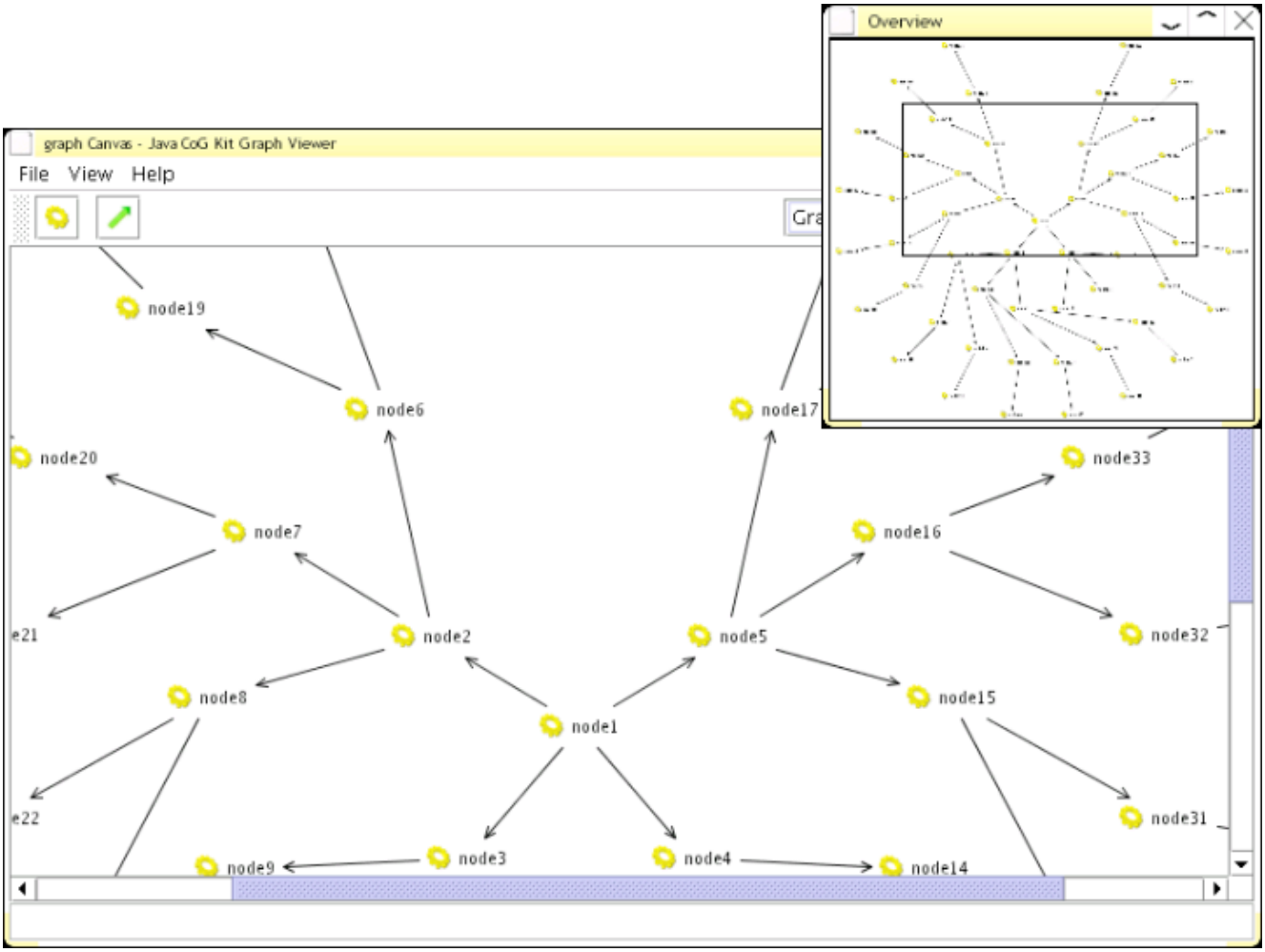

Figure 5. GridAnt Monitoring Interface

- *grid-status*: Queries the remote Grid resource to resolve the execution status of the given job.
- *grid-ps*: Queries the remote Grid resource to resolve the execution status of all the jobs submitted by the user.

Further, all important Ant tasks and task containers are extended to use the GridAnt runtime environment facilitating inter-task synchronization and communication. For a detailed description of GridAnt tasks, refer to [34]. Figure 6 shows a sample XML specification for expressing GridAnt dependencies as depicted in Figure 2.

As shown in Figure 4, GridAnt adopts a layered architecture containing commodity tools to implement its Grid functionality. GridAnt tasks described above use the abstractions provided by the Java CoG kit to interface with Grid systems. The current CoG Kit implementation provides execution facilities on Grid resources using the Globus Toolkit version 2 [35] and version 3 [24]. Using the CoG kit enables the GridAnt system to seamlessly integrate other Grid architectures such as [36], Condor [37], and Legion [38] whenever they are supported by the CoG Kit.

2.4 Workflow Monitoring

GridAnt is a fully functional command line tool where the XML workflow specifications can be edited with any text editor. Nevertheless, a graphical visualization tool is also provided, allowing real-time monitoring of the progress of the workflow. The visualization tool integrates seamlessly with GridAnt, acting as a front-end to the workflow engine. A GridAnt workflow specification can be loaded within the visualization tool, which can also initiate the execution of the workflow. Upon loading of the XML workflow specification, the visualization tool maps the XML elements into a hierarchical graph. Various trans-

```
<target name="sampleWorkflow">
  <sequential>
      <grid-setup/>
      <grid-authenticate/>
      <grid-copy
            name="copyInputFile"
            provider="GT2"
            security="xmlSignature"
            delegation="limited"
            from="gsiftp://server1/inputFile"
            to="gsiftp://server2/inputFile"
            parallelStreams="4"
            tcpBuffer="16384"/>
      <grid-execute
            name="myApplication"
            provider="GT2"
            server = "server2:1234"
            security="xmlEncryption"
            delegation="full"
            executable = "myApplication"
            arguments="-file inputFile"
            directory="/home/test"
            localExecutable="false"
            redirect="false"
            outputFile="outputFile"
            errorFile="outputFile"/>
      <grid-copy
            name="copyOutputFile"
            provider="GT2"
            security="xmlSignature"
            delegation="limited"
            from="gsiftp://server2/outputFile"
            to="gsiftp://server3/outputFile"
            parallelStreams="4"
            tcpBuffer="16384"/>
  </sequential>
</target>
```

Figure 6. GridAnt XML Specification

formations can then be used to restructure the hierarchical graph into more meaningful user-friendly formats. For example, a graph transformation can convert the dependencies between GridAnt targets into graph edges connecting the nodes representing the targets. Another transformation converts <sequential> and <parallel> recursive elements into a flow graph. The resulting graphs can be displayed using several views available in the visualization tool. A particularly significant view is the "Graph View", which can employ multiple layout engines to arrange the nodes in the graph. After the workflow specification is loaded, the visualization tool can be used to start the execution of the workflow. The feedback mechanism provided by the GridAnt engine is used to monitor the status of each element

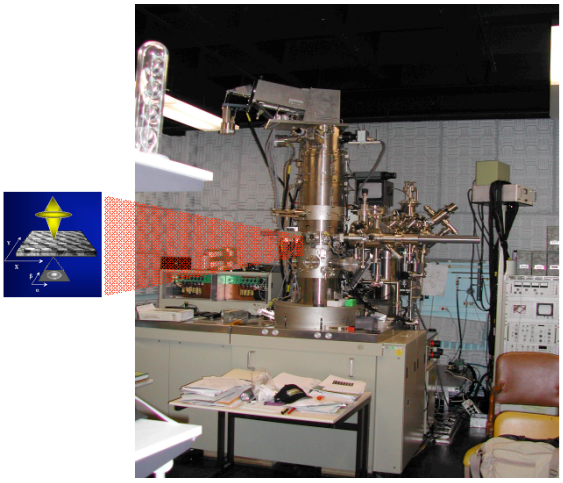

Figure 7. Argonne National Laboratory's Advanced Analytic Electron Microscope

in the workflow, and the visual appearance of the graphical elements displayed by the visualization tool are changed accordingly.

The visualization tool can scale to accommodate large specifications. Preliminary scalability tests have revealed the possibility of displaying graphs having thousands of elements (nodes and edges). The use of heuristic layout algorithms with linear execution times allows displaying of such graphs in relatively short times, in the order of tens of seconds.

3 Application: Position-Resolved Diffraction

To validate our initial workflow design, we have applied the GridAnt framework to a real-life scientific application. A new experimental technique, named position-resolved diffraction, is being developed to study nanoscale structures as part of Argonne National Laboratory's advanced analytical electron microscope. With this technique, the electron beam from an analytical electron microscope is used to scan a nanosized, disc-shaped magnetic specimen. At each sampling point a two-dimensional image representing the resulting diffraction pattern is acquired and stored (see Figure 7). The analysis of the spatial variation in the diffraction pattern allows the researcher to study the form and direction of the field lines in the magnetic field of the sample.

As much as one terabyte of data can be taken during such an experiment. This analysis of the data requires a resource rich Grid infrastructure to fulfill the real-time constraints. The results need to be archived, remote compute resources need to be reserved and made available during an experiment, and the data needs to be moved to the compute re-

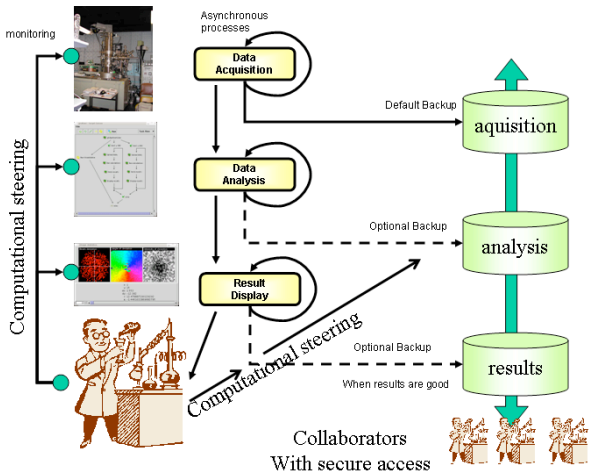

Figure 8. Asynchronous processes define a workflow that is steered by the scientist to support the problem-solving process with the help of abstract Grid tasks.

sources where they will be analyzed. The results need to be gathered and presented in a form that is meaningful to the scientist. GridAnt provides a convenient abstraction for formulating these tasks while reusing the patterns for file transfer, job execution, and job management (see Figure 9). At the same time it hides much of the complexity that the Grid application user may not want to deal with. The overall application presents one of many scientific use patterns that occur in high-end instrument scenarios. This includes a high amount of interaction during an experiment that must be dealt with in an adaptive and flexible way. Unexpected and unpredicted experiment conditions must be considered. Hence the instrument operators interface to the Grid must be as simple as possible while at the same time providing the much needed flexibility to interactively modify the experiment setup. This is achieved by reusing our graphical components and integrating them in a scientific problem solving environment that targets the flexible use of such an instrument.

The need for such a flexible infrastructure is demonstrated through a simple experiment flow depicted in Figure 8. The elementary logic of the instrument control can be expressed in a sequence of processes that depend on each other. We distinguish the following processes

data acquisition that gathers the time-delayed images from the electron microscope.

backup that backs up the incoming data.

data analysis that performs scientific calculations on the time-delayed images.

result display that gathers the results from the data analysis in a form easy to interpret by the scientist to make further judgments for steering the experiment.

By using the GridAnt system, the scientific user can conveniently express a Grid-enabled experimental run in the form of an XML specification. By monitoring the experiment through GridAnt tools, the user can decide when, what, and where to back up data gathered during the course of the experiment.

4 GridAnt Workflow Service

Using the GridAnt system in an advanced scientific application has taught us several important lessons. One of the most important among them is that the GridAnt system described in Section 2 requires that the client be connected to it throughout the duration of the workflow run. This might be an acceptable requirement for interactive or short batch jobs. For jobs that require extensive execution times, however, such a requirement is unacceptable. Clearly, a mechanism is needed that allows users to submit their workflow specifications, disconnect from the system, and reconnect at a later time, retrieving all the information generated during the disconnected period. Such a mechanism requires an additional level of redirection through a proxy service that acts on behalf of the user, ubiquitously maintaining a connection to the GridAnt system. Ongoing research is focusing on the transformation of the GridAnt system into a Grid service [24] that provides the functionality of such a proxy.

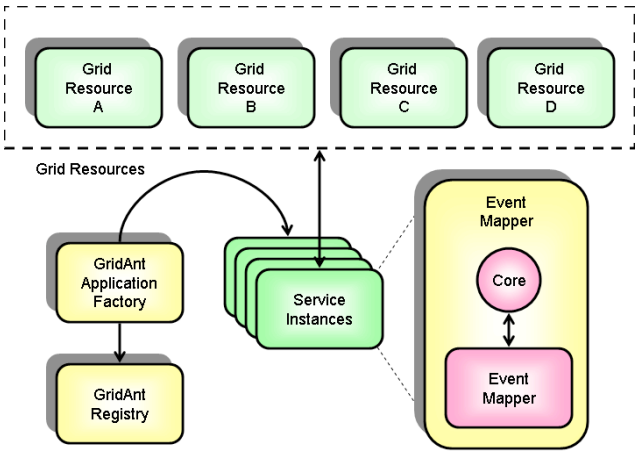

Figure 10. GridAnt Service Architecture

The GridAnt Grid service accepts the remote submission of an XML workflow specification, controls the flow between the various activities, and provides the Grid service-enabled GridAnt client with a set of events to enable the

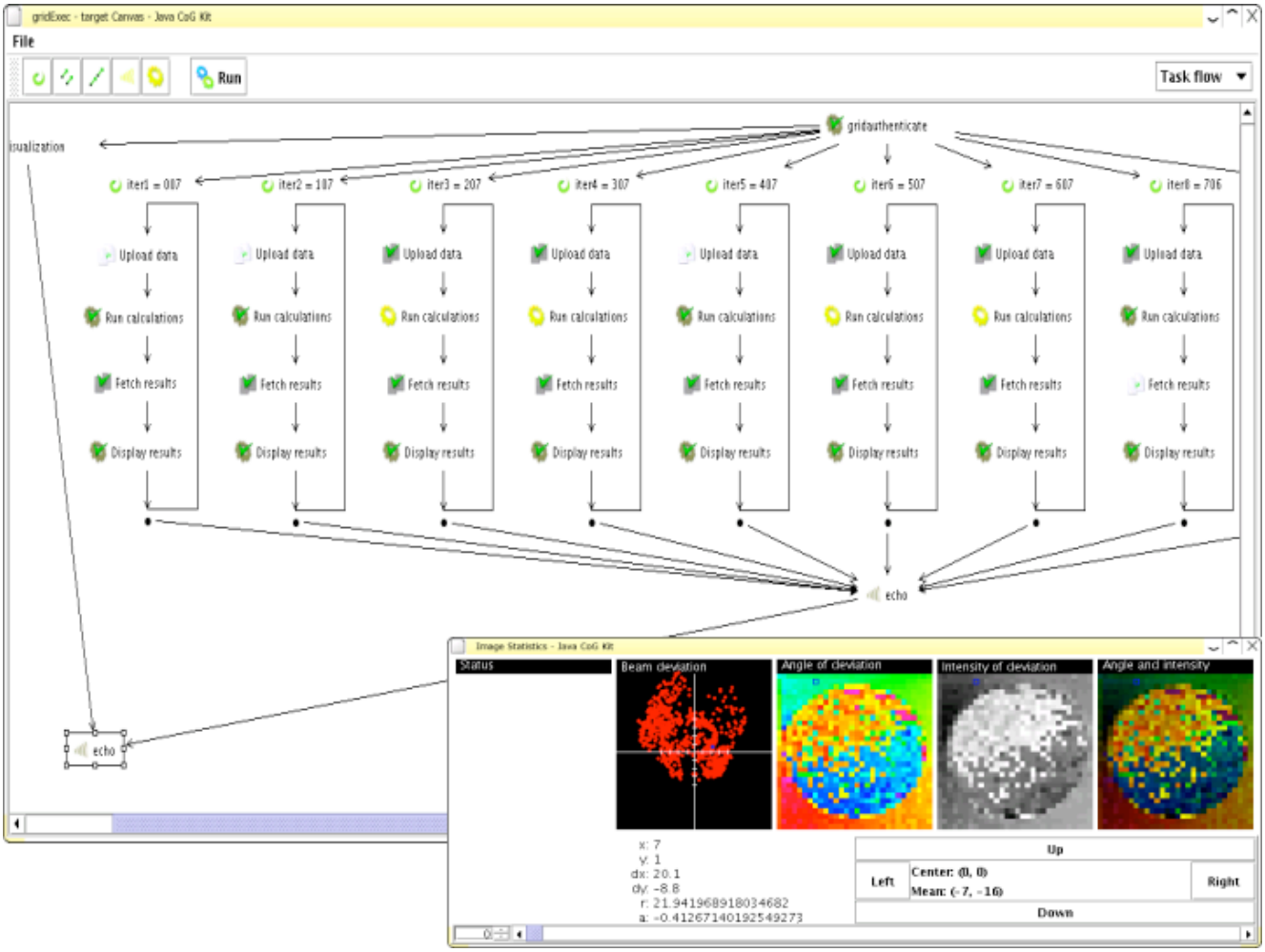

Figure 9. The data analysis for the electron microscope is formulated as workflow that uses Grid resources. The progress of the calculation is updated in real time

visualization of the workflow run. As shown in GridAnt service architecture (see Figure 10), the GridAnt client can connect to the GridAnt factory to instantiate a new service instance or query the GridAnt registry for existing workflow runs. The GridAnt service instance is composed of two important modules: the GridAnt core and the event mapper. The GridAnt core is responsible for accepting the XML specification, controlling the workflow, and intertask communication, and informing the event mapper about all the workflow-related events generated by GridAnt. The event mapper bridges the GridAnt event notification mechanism into the OGSA event notification mechanism. This is essential for the client applications to remotely monitor the workflow progress via Grid service artifacts. The event mapper logs all the events generated by it to be retrieved and replayed at a later time, thereby simulating the workflow progress.

5 Summary and Future Work

Numerous workflow control systems have been proposed in literature. Most of these assist in aggregating a set of services that clients can conveniently use. This paper proposes a workflow system, called GridAnt, that assists Grid users in orchestrating a set of Grid activities and expressing complex dependencies between them in XML. GridAnt essentially consists of four components: a workflow engine, a runtime environment, a workflow vocabulary, and a workflow monitor. Apache Ant is selected as the GridAnt workflow engine because of its extensibility and popularity in the Java community. Several communication mechanisms between GridAnt tasks are discussed. The current implementation

of GridAnt uses a global whiteboard style of communication mechanism that can exchange arbitrary formats of data between independent tasks. A set of important Grid-centric tasks are identified as the workflow vocabulary; being an extension to Ant, this workflow can be incrementally extended by the Grid community. An elaborate visual monitoring interface is also provided to the GridAnt system that enables users to monitor the progress of their workflows.

Initial experiences with the GridAnt system emphasized the need to extend our model into a service architecture. The GridAnt service architecture comprises an application factory, a registry, and the GridAnt service. The GridAnt service has a GridAnt core that handles the workflow control and an event mapper that converts GridAnt events into OGSA-specific events.

The GridAnt system presented in this paper is a work in progress. Although an initial prototype is available for evaluation, it will undergo significant extensions based on community feedback. Concurrent research is focusing on issues such as extending the available set of workflow tasks to incorporate more sophisticate control structures. Further, the GridAnt system needs to be extended into a service-oriented architecture in order to act as a proxy on behalf of the Grid-users. An advanced information exchange mechanism is being designed to incorporate dynamic publish/subscribe style of communication between independent tasks. The current implementation of GridAnt supports the Globus Toolkit versions 2 and 3; but efforts are being made to support other Grid environments such as Unicore, Legion, and Condor.

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.

References

[1] "Workflow Management Coalition," Web Page. [Online]. Available: http://www.wfmc.org

[2] "Staffware," Web Page. [Online]. Available: http://www.staffware.com

[3] *Software-Ley. COSA User Manual*, Software-Ley GmbH, Pullheim, Germany, 1998.

[4] "InConcert: Tibco Software," Web Page. [Online]. Available: http://www.tibco.com

[5] "Eastman Software," Web Page. [Online]. Available: http://www.eastmansoftware.com

[6] S. P. Nielsen, C. Easthope, P. Gosselink, K. Gutsze, and J. Roele, "Using Lotus Domino Workflow 2.0," IBM, Poughkeepsie, USA, Redbook SG24-5693-00, 2000.

[7] "Websphere MQ Workflow," Web Page. [Online]. Available: http://www-3.ibm.com/software/integration/wmqwf/

[8] "Visual Workflo: FileNet," Web Page. [Online]. Available: http://www.filenet.com

[9] "I-Flow: Fijitsu," Web Page. [Online]. Available: http://www.i-flow.com

[10] W. M. P. van der Aalst, A. P. Barros, A. H. M. ter Hofstede, and B. Kiepuszewski, "Advanced Workflow Patterns," in *Conference on Cooperative Information Systems*, 2000, pp. 18–29.

[11] "wftk: Open-source Workflow Toolkit," Web Page. [Online]. Available: http://www.vivtek.com/wftk

[12] G. von Laszewski, M. Seablom, M. Makivic, P. Lyster, and S. Ranka, "Design Issues for the Parallelization of an Optimal Interpolation Algorithm," in *Coming of Age, Proceedings of the 4th Workshop on the Use of Parallel Processing in Atmospheric Science*, G.-R. Hoffman and N. Kreitz, Eds., European Centre for Medium Weather Forecast. Reading, UK: World Scientific, 21-25 Nov. 1994, pp. 290–302. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski94-4dda-design.pdf

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

[14] G. von Laszewski and I. Foster, "Grid Infrastructure to Support Science Portals for Large Scale Instruments," in *Proceedings of the Workshop Distributed Computing on the Web (DCW)*. University of Rostock, Germany, 21-23 June 1999, pp. 1–16, *(Invited Talk)*. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--rostock.pdf

[15] G. von Laszewski, M.-H. Su, J. A. Insley, I. Foster, J. Bresnahan, C. Kesselman, M. Thiebaux, M. L.

Rivers, S. Wang, B. Tieman, and I. 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 Mar. 1999. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--siamCmt99.pdf

[16] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski, and G. Premchandran, "WebFlow - A Visual Programming Paradigm for Web/Java Based Coarse Grain Distributed Computing," *Concurrency: Practice and Experience*, vol. 9, no. 6, pp. 555–577, 1997.

[17] "Triana Workflow," Web Page. [Online]. Available: http://www.triana.co.uk

[18] M. Lorch and D. Kafura, "Symphony - A Java-based Composition and Manipulation Framework for Computational Grids," in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, Berlin, Germany, May 2002, pp. 21–24.

[19] S. Krishnan, R. Bramley, D. Gannon, M. Govindaraju, J. Alameda, R. Alkire, T. Drews, and E. Webb, "The XCAT Science Portal," in *Supercomputing*, 2001.

[20] "Common Component Architecture," Web Page. [Online]. Available: http://www.cca-forum.org

[21] J. Coa, S. Jarvis, S. Saini, and G. Nudd, "GridFlow: Workflow Managament for Grod Computing," in *3rd International Symposium on Cluster Computing and the Grid*, 2003, p. 198.

[22] "Ptolemy II," Web Page. [Online]. Available: http://ptolemy.eecs.berkeley.edu/ptolemyII/

[23] "LabVIEW," Web Page. [Online]. Available: http://www.ni.com/labview

[24] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Web page, Jan. 2002. [Online]. Available: http://www.globus.org/research/papers/ogsa.pdf

[25] "BPEL4WS: Business Process Execution Language for Web Services Version 1.0," Web Page. [Online]. Available: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel

[26] P. Wagstrom, S. Krishnan, and G. von Laszewski, "GSFL: A Workflow Framework for Grid Services," in *SC'2002*, Baltimore, MD, 11-16 Nov. 2002, (Poster).

[27] "DAGMan (Directed Acyclic Graph Manager)," Web Page. [Online]. Available: http://www.cs.wisc.edu/condor/dagman/

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

[29] G. von Laszewski and K. Amin, "GridAnt," Project Page, July 2003. [Online]. Available: http://www-unix.globus.org/cog/projects/gridant/

[30] "Jelly: Executable XML," Web Page. [Online]. Available: http://jakarta.apache.org/commons/sandbox/jelly/

[31] "Ant – a Java-based Build Tool," Web Page. [Online]. Available: http://ant.apache.org

[32] "The Apache Software Foundation," Web Page. [Online]. Available: http://www.apache.org

[33] G. von Laszewski, I. Foster, J. Gawor, and P. Lane, "A Java Commodity Grid Kit," *Concurrency and Computation: Practice and Experience*, vol. 13, no. 8-9, pp. 643–662, 2001. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--cog-cpe-final.pdf

[34] G. von Laszewski, K. Amin, S. Hampton, and S. Nijsure, "Gridant – white paper," Argonne National Laboratory, Tech. Rep., 2003. [Online]. Available: http://www.globus.org/cog/grant.pdf

[35] "The Globus Project," Web Page. [Online]. Available: http://www.globus.org

[36] "Unicore," Web Page. [Online]. Available: http://www.unicore.de/

[37] "Condor: High Throughput Computing," Web Page. [Online]. Available: http://www.cs.wisc.edu/condor/

[38] A. S. Grimshaw and W. A. Wulf, "The Legion Vision of a Worldwide Virtual Computer," *Communications of the ACM*, vol. 40, no. 1, pp. 39–45, January 1997. [Online]. Available: http://legion.virginia.edu/copy-cacm.html