Work Coordination for Grid Computing

Gregor von Laszewski[1,2]*Mike Hategan[2] Deepti Kodeboyina[1]
[1]Argonne National Laboratory, Mathematics and Computer Science Division
Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60440
[2]University of Chicago, Computation Institute,
Research Institutes Building #402, 5640 S. Ellis Ave., Chicago, IL 60637-1433

Contents

*Corresponding author: gregor@mcs.anl.gov

1 Introduction

Scientific computing has been one of the main drivers for the evolution of computing technologies and frameworks. We see a trend that on the one hand, motivates the development of new infrastructure to support the requirements of the ever more complex scientific applications and on the other hand the availability of new technology influences the design of scientific applications. Hence, they provide a motivation for growth in both domains. In this paper we will focus on one of the models that has recently spawned a great deal of interest in the research community: the coordination of work as part of the scientific discovery process or in short, scientific workflows. In particular, we will discuss scientific workflows that pertain to the Grid environment.

Although Grid computing has become a valuable asset to various domains for integrating distributed resources as part of virtual organizations, many challenges still remain. One of the challenges in Grid computing that needs to be addressed, is the coordination of tasks expressed as workflows. This challenge is motivated by requirements posed by scientific and business communities that demand an even higher level of abstraction than is provided by the current generation of Grid middleware. Such workflows enable rapid prototyping and reuse of the Grid infrastructure in an easy fashion. Focusing on scientific applications, workflows will help to coordinate not only traditionally Grid related resource management, but will bridge the gap between commodity tools and Grid middleware as pioneered by the Java CoG Kit project.

An important factor that contributed to the success of Grids is the development of sophisticated middleware based on standards. We have seen over the last decade a shift from API-oriented middleware to service-oriented middleware. However, the standards are still evolving and a higher level workflow framework can protect the application user against the complexities related to this evolution. Sophisticated workflow frameworks can provide the necessary abstractions making the use of Grids possible, even for scientists with little desire to learn the many intransient aspects of the Grid.

The chapter is organized as follows. First, we give a perspective on why workflows are at present so important and provide a commonly used definition. Next, we provide a simple definition for Grid workflows. We identify what different mechanism exist to coordinate work within a Grid followed by a short survey of a selected number of workflow-related tools and systems that explicitly address Grid computing. Next, we focus in more detail on the Java CoG Kit workflow solutions in order to highlight some of its features that satisfy many scientific application requirements. We conclude the chapter by identifying future research issues.

1.1 Why Workflows?

A number of flow patterns were developed over time that helped shaping today's infrastructure and are influencing future compute infrastructures leading to knowledge networks. Figure 1 depicts some important flow patterns and their complexities as applied to the evolving cyber infrastructure.

We observe that the development of software and hardware is a direct result of the desire to solve complex problems. With the increased ability to build hardware and the software to effectively use the hardware, more complex problems can be addressed today. The availability of such infrastructure brings new opportunities that result in a change of collaborative patterns. While in the past, single specialized computers may have been accessed in sequential fashion, the Grid approach allows us to extend the use model for resources provided on the Internet. Sharing limited resources in a collaborative fashion is one of its goals. We see that this increased demand for coordination expressed through flow patterns is accompanied by an increased complexity of hardware. The motivation for scientific workflows is based on the need to organize the*compute flows* that are part of the sophisticated scientific problem solving process. Workflows will be a much needed tool to manage the complexities of this new infrastructure available to us.

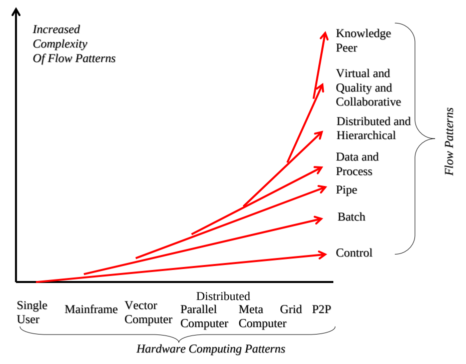

Increased Complexity Of Flow Patterns
Knowledge Peer
Virtual and Quality and Collaborative
Distributed and Hierarchical
Data and Process
Pipe
Batch
Control
Flow Patterns
Single User
Mainframe
Vector Computer
Distributed Parallel Computer
Meta Computer
Grid
P2P
Hardware Computing Patterns

Figure 1: Increased complexity of flow patterns.

When taking a closer look at the evolutionary process we observe the changes in use modality, resource access, and the necessary technologies

Table 1: Workflows are the next step in the theoretical and technological evolution toward development of knowledge networks for e-Science.

Concepts	Use modality	Resource access	Technologies
Sequential programs	single	exclusive	Programming Languages, pipes
Parallel programs	single	exclusive	Parallel programming languages, MPI, ...
Batch processes	single	shared	LSF, PBS, Condor, ...
Parameter study	single	shared	Nimrod, first gen. Grid, ...
Problem solvers	multiple	shared	Portals, Gateways
Workflows	multiple	shared, distributed	BPEL, process networks, Google
Knowledge network	community	community	semantic Grids

developed to support the scientific problem solving process integrate the. We depict this evolution in Table 1. Due to the ever increasing complexity of the scientific problems, we see the need for large collaborative efforts to be applied to a scientific quest performed by a group of researchers. Hence "programming in the small" by a single researcher is no longer adequate. Scientific progress of the most challenging problems is based on the concept of "programming in the large" where a collaborative effort is involved. The same is valid for the handling of resources as part of virtual organizations which leads us to believe that such research needs the support of a grid-based framework.

1.2 What is the Grid Approach?

Grid computing has been envisioned since the very early days of computer science. We have found references to the vision of a computational Grid infrastructure from as early as 1969 in [24]:

> *We will probably see the spread of computer utilities, which, like present electric and telephone utilities, will service individual homes and offices across the country.*

Influenced by research in parallel and distributed computing in the early 1990's and using metacomputing as a stepping stone, the term Grid was introduced in 1998. However its definition has evolved since than. The term was first formalized in the book "The Grid: Blueprint for a New Computing Infrastructure" [16] where a Grid is defined as

> *A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.*

However, von Laszewski projected earlier that year that we are in the need of an extended infrastructure that includes a

> *shared computing infrastructure of hardware, software, and knowledge resources.*

Important to note is that one of the pillars of this infrastructure contains also "humans" as part of the set of knowledge resources [46].

In 2000, Foster refined the earlier definition of Grids to include social and policy issues, stating that Grid computing is concerned with "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations." Here, the key concept enables resource-sharing among a set of participating parties. This includes the recognition of a virtual organization that governs rules for sharing resources under quality of service constraints, based on community standards.

However, this definition is still not expressive enough by emphasizing that a human in this infrastructure can itself be a resource. In addition, we saw that the term Grid was used interchangeably in the community to refer to an idea and a physical instantiation of a hardware infrastructure. To distinguish better between these two usages of the term Grid, von Laszewski has defined the term "Grid Approach" in 2000 [42, 49, 43]. The important factor here is that Grid computing promotes an approach to conduct scientific research and business collaborations that must not stop by focusing on hardware and software, but that enables a vision for a shared infrastructure including humans.

> *We define the terms (a) the* Grid approach*, or paradigm, that represents a general concept and idea to promote a vision for sophisticated international scientific and business-oriented collaborations and (b) the physical instantiation of a* production Grid *based on available resources and services to enable the vision for sophisticated international scientific and business-oriented collaborations.*

This distinction is important as the Grid approach includes the vision of organizational facilities to support collaborations with the help of workflows. Hence, workflows are an essential part of the Grid approach.

1.3 Workflow

Different definitions of workflow can be found in literature. The most common definition can be found in [51]. We quote

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

In order to execute such a workflow we are in the need of a *workflow management system.* According to [51] it is defined as follows

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

If we are in the business of doing science or working on the Grid, the same definitions would apply to Grid workflows. Then the question that arises is " Is there anything different between business and scientific workflows?"

The answer to this question is not as straightforward as some other researchers claim. In fact, some point out wrongly that only scientific workflows run for multiple days and require a great number of resources. However, even in [51] such scenarios have been envisioned to be part of business workflows. We find that the real difference is in (a) integrating Grid middleware into the workflow management system, and (b) focusing on the definition of workflow models that target use cases utilizing the Grid infrastructure.

2 Grid Workflow Management Systems

As is obvious from Section 1.1, the development of Grid workflow systems is a natural progression based on a combination of interwoven factors between available hardware, software, and the applications targeted. Each of these categories is driving progress within the other categories. Recently we have seen the emergence of the Web and Grid technologies that together provide a path towards research activities in workflows. This progression can be classified into a number of phases which have been described here.

2.1 Pre Web Services Phase

As part of this progress, the desire to automate processes has lead to the development of early office automation software (seventies to mid eighties) leading to business (mid eighties to today) and scientific workflow management systems (late eighties till today). We share this view with [**?**] in which a detailed historical perspective of this early development of workflow management systems is given and arranges the systems in chronological order.

2.2 Pre-Grid Phase

The Grid Phase originated from the definition of the metacomputer and already at that early stage a number of workflow systems were developed that have had significant impact not only in the definition of scientific workflows, but also in the definition of Grids and their underlying middleware. Such systems include for example HenCE, a precursor the GECCO and Java CoG Kit, Webflow, and SCIRun.

2.3 Early Grid Phase

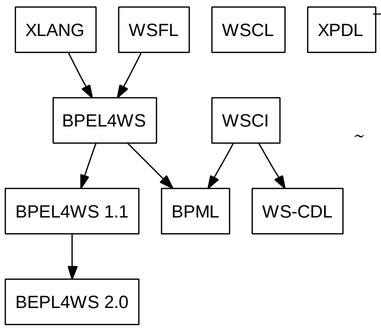
Originally, only a few Grid-based systems that dealt with workflows were available. The earliest of these systems was called GECCO and has now evolved to the Java CoG Kit. Shortly after this, Condor-G wasintroduced and Unicore has been designed. Systems such as SCIRun have been augmented to include Grid scheduling facilities.

2.4 Grid Standards Phases

In the recent past, Grid computing has been through several phases in defining Grid standards. The latest phase is still ongoing and several standards have been submitted to OASIS. During these phases, the definition of Grid workflows based on the evolving standards was difficult and it was discovered that higher level of abstractions can provide a convenient interface to workflows. Such high level abstractions are defined for example by the Java CoG kit not only on the workflow specification level, but also on the interface and API level. Other systems such as Condor or Chimera have introduced language level abstractions that make it technically possible to adapt to the evolving standards as has been demonstrated through the last two years.

2.5 Web Services and Grid Phase

In the late 1990s, web services were standardized and this led to a renewed interest in the development of standards for workflows and interoperable web services involving coordination and choreography. In addition, with the introduction of the Web Services Resource Framework and its associated standards, the Grid community has come closer to the Web services community. A large amount of overlap between both communities exists and several technologies contributed by both groups enhance each other. Due to evolving standards not only in the Grid community, but also in the Web services community it is still difficult to develop frameworks that will be widely accepted and of lasting impact that combine the three technologies: Grids, Web services and workflows. To illustrate the difficulty in developing integrated approaches, let us focus on Figure 2 in which we have listed just a small number of relevant standards in relationship to web services. We observe that even in a matter of only three years a number of efforts have begun, but that through merging the number has actually become smaller. Today, efforts such as BPEL4WS and WS-CDL have attained strong momentum. Due to the evolution of the Grid standards, efforts such as GSFL [25] that were designed prior to BPEL and WS-CDL have been halted. We expect that further efforts in the Grid community will be initiated in the near future. This is evident from the creation of workflow-related research groups in the Global Grid Forum and the active participation of companies interested in making Web services-based workflow standards a success in the forum.



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

Figure 2: Evolution of the standards with respect to key web-related technologies

2.6 Grid and Web Upperware

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

3 Grid Coordination Paradigms

This section describes a number of paradigms that are directed towards coordination of work conducted on the Grid. They are targeted towards job execution with the help of Grid resources. Many of these paradigms are drawn from parallel and distributed programming concepts.

3.1 Grid Queues and Sets

Queues are well-known data structures that allow organization and execution of tasks in sequential order. Queues have a retrieval policy attached with them that determines in which order the tasks are retrieved from the queue in relationship to how they were put into it. In case no order is specified, we also refer to it as a set which in many cases has a random order for retrieval. Batch queues are defined as queues that can run the tasks administered by them to completion without human interaction. Grid Queues are augmented with specialized single sign-on authorization capabilities, making it also possible to monitor the progress of a job in the queue directly from the client. Often the term Grid metascheduler is used to refer to Grid queues that defer tasks based on load or other resource information among multiple Queues, one for each resource. Examples of queuing systems are PBS [30], LSF [26], MAUI [27], MOAB [28], GridEngine [35].

3.2 Grid Pipes

A pipeline in computer science allows information to flow in one direction similar to water flowing in a pipe. Often the notation `a | b` is used to denote the flow of information between a and b. It is obvious that pipes can be utilized easily to express elementary simple flow patterns. Various performance enhancing implementations exists. One strategy is to blast all information from a to b as quickly as possible and once the information is available at b to start the process b. However, in order to increase parallelism the pipe is most often implemented in such a fashion that process a and b are started in parallel and that the output from a is directly written into the input of b without waiting for the output to be completely written to b. In a Grid, we must make use of advanced features for establishing pipes. In the Globus Toolkit GT2 this feature is provided through the GASS libraries. Other systems such as the Storage Resource Broker (SRB) have such capabilities.

3.3 Grid Programming with Dependencies

One of the most popular programming paradigms for coordination of work within Grids is the use of dependencies between entities. Such entities can be Grid tasks, which are program instructions loaded into program memory for accessing a Grid resource, or Grid processes which are running instances of a Grid program, that include all variables values and state information. Hence, a Grid process contains the memory that is associated with the resource containing the task-specific data including operating system resources, file descriptors, security attributes, and the states.For the purposes of this chapter, a Grid task is defined as an abstract work item formulated through a program, and a Grid Process, its instantiation on a Grid resource through a concrete mapping.

Dependencies between tasks, such as, "task a has to be done before task b" are very often depicted in a graphical form to identify the control or dataflow between tasks. This is denoted with `|a->b|`and many systems exist that make use of graphical tools for specifying dependencies between tasks. The mapping onto Grid resources and the instantiation into Grid Processes is done typically through a specialized resource mapper that minimizes the cost of execution or the duration while considering other scheduling factors. A Grid metascheduler is a simple form of such a Grid resource broker.

3.4 Grid Components with Strongly Typed dependencies

In contrast to simple dependencies between components, several systems exist that include strongly typed objects between components. In many cases, such types are associated between input and output variables and therefore resemble a mixture between pipes and programming with dependencies. Many traditional GUI toolkits that use this paradigm exist including but not limited to AVS, IBM xplorer, and LabView. Grid systems that use this metaphor are XCAT. The advantage of such a system is that for complex component dependencies a typed system provides an elementary program verification to check the dependencies.

3.5 Grid Dataflow

The term data flow is most often used in two contexts. First, it represents the classical dataflow system, that changes the values of variables and objects while forcing their recalculation if other variables or objects they depend on change. Second, it is also used in connection to dataflow networks that specifies concurrently executing processes while communicating through channels to send data between each other. In the case of Grids a dataflow system may become quite complex as the objects may belong to different administrative domains and special care has to be put in place to make sure the flow is not interrupted through possible network outages and failures. Naturally the same is valid for Grid dataflow networks. Examples for such systems are VDL and the Java CoG Kit Karajan Workflow Framework.

3.6 Service-Oriented Grid Models

Recently, service-oriented architectures have become the cornerstones of Grid middleware development. Hence, it is appropriate to analyze how work can be coordinated with such an architectural model. The centerpiece of a service-oriented architecture is the interplay of three entities: a registry, a provider, and a consumer. A provider registers with the registry a service that can be consumed by the consumer. To do so the consumer, looks up in the registry an appropriate service and binds to it so that others may not consume the same service. While in the Web services world the services are assumed to be stateless, Grid Services have state associated with them that are typically represent the state of a Grid resource. Hence it is possible to integrate the state as an important item as part of the consumer or client. This naturally has advantages and disadvantages not all of which are not relevant to this discussion. One of the advantages, is that a great deal of standards specification work conducted within the Web and Grid community is to design a Open Grid Services Architecture. The introduction of such standards will ultimately help the complex creation of interoperable Grid services. We must learn therefore the lessons from CORBA, DCOM, and others in order to avoid the creation of yet another framework that promises interoperability but cannot deliver it due to the incompatibility issues of the actual implementations that do not entirely adhere to the standard. In order to be successful the standard must be complete, yet simple to implement. Evolution of the standard itself must be considered during its implementation within the community. To make sure we understand why

SOAs are good for grids and how they can be used for work coordination we need to take a closer look at what some of the mechanisms to create and connect to services. A service is typically described through a WSDL specification exposing possible methods with which we can interface to such a service. This in itself is nothing new as we have done the same thing already in object-oriented languages for decades. The definition of a Java interface comes very close to what a WSDL document can represent. However, it does not define the semantics nor does it tell the user in which order other services need to be called.

Hence, we require orchestration and choreography. Orchestration defines the service sequence as well as additional logic to process data. It is not related to the actual data representation. Choreography defines how an entity interacts with another one. This may be done through message exchanges in order to establish a *conversation* between the parties involved.

As can be seen, a service-oriented architecture is defined on such a low level that we still require higher level abstractions and services providing the actual functionality of coordinating the work for application users. Even with the introduction of BPEL which allows us to define such interactions, we still need more tools since no application scientist should ever have to use XML to specify a program. XML is just an intermediate format that is useful for the middleware designer, but not for the high-level Grid architect. As such defining a task graph with dependencies as outlined in section will be much more convenient and realistic than to define a complex BPEL specification. Automatic tools must be created that make this possible.

3.7 Parallel Programming Languages

One convenient way to specify high level dependencies between entities is to use a parallel programming language in which parallel constructs like `||` allow execution of two tasks in parallel. Having such a language, allows the users to focus on the parallel aspects of the problem at hand. Instead of focusing on writing complex WSDL or BPEL descriptions, tools must be developed that hide this process entirely for the user of a Grid.

3.8 The Visual Programming Pattern

Often visual programming languages are used as an entry point to the complex Grid infrastructure. However, we must be aware that not every semantic relationship in a complex Grid program can be expressed through a visual representation easily. Hidden or implicit dependencies can be formulated more easily through programming language constructs.

Nevertheless, visual programming environments make the creation or monitoring of complex Grid workflows easier to some extent. In Figure 3 we depict a small number of dataflow patterns that are typically found in visual programming languages. While comparing them to their programming language counterparts, some of them seem more unintuitive that the actual program. We observed this while creating workflows with tools such as Kepler [1]. Although they provide a nice user interface, the complexity involved in assembling is often greater than what can be achieved with a scripting or parallel programming language. Hence, some systems provide just a task model such as the Java CoG Kit and allow display of the task dependencies and their status through hierarchical viewers (see Figure F:graph).

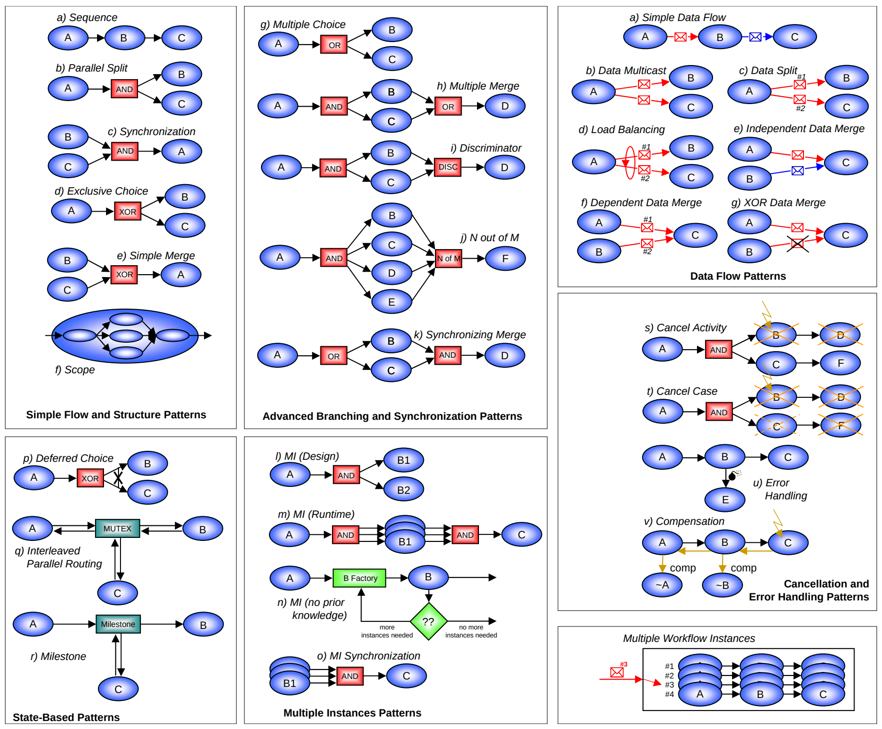


Figure 3: Patterns that are often used in visual dataflow programming languages.

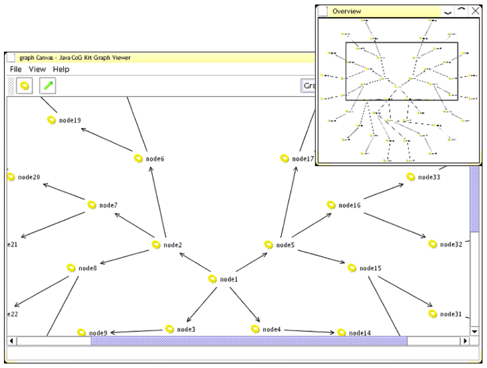


Figure 4: The Java CoG Kit Workflow viewer enables to display and monitor workflows.

4 Select Grid Workflow Systems

A number of research groups have worked on the creation of workflow systems for Grid and non-Grid environments. We will enumerate a subset of these systems in order to highlight features that have been found useful by the community. The systems outlined here are Condor DAGMan, Globus, Pegasus and Chimera, Unicore, Triana and Kepler, SCIRUN, and BPEL4WS. We will look at the Java CoG Kit workflow in more detail as it projects an integrated approach to Grid workflow thus fulfilling requirements of many user domains. Several other workflow systems are listed in Table[].

List is not entireley complete and consistent

4.1 Condor DAGMan

DAGMan (Directed Acyclic Graph Manager) [9] is a recent addition to the Condor [10, 39] software. It is a meta-scheduler that extends the condor scheduler by allowing dependencies between condor jobs. The representation of dependencies is specified as part of a direct acyclic graph, where the nodes represent programs and the edges, dependencies between the programs. This abstraction is similar to the one introduced earlier in [48]. The DAG is described in a simple text file in which each node represents a condor job that takes a number of input files and produces a number of output files. Besides specifying dependencies, DAGMan can support elementary error recovery and reporting. In each node, a user can define a pre and a post script that

Table 2: List of Scientific Workflow Environments

Name	**Grid Library**	**Programming Paradigm**
Condor DAGMan [8, 9]	Condor, can run on top of GT2 (Condor-G)	Directed Acyclic Graphs (DAGs)
DiscoveryNet [14, 19]	Globus Toolkit	Discovery Process Markup language with Task objects
GRID superscalar	GT2.x, GT4, Ninf-G, SSH/SCP	Parallel Execution using Taskgraphs
GridAnt [50, 3]	Grid tasks via CoG Kit	Targets and tasks based on Apache ant
GriPhyN VDS (Chimera) [6]	Globus Toolkit(GT)	uses DAGs produced by Pegasus
GridPort []	uses GT3 via CoG Kit, CSF, GridFTP	Composition using core components and services
GridNexus [18]	GT3	Internal DAG with JXPL
Grid Service Broker [17]	GT2,Alchemi	parametric computing for compute and data grids
ICENI [20]	ICENI Grid based on JINI	Component-based dataflow using Execution Plans
I-Lab [21]	Globus ToolKit	GADL is based on Petri Nets
Karajan [22]	GT2.4, GT3.02, SSH (future targets: Condor, GT4.0, Unicore)	Mike: will have to describe
Pegasus [31, 11]	Condor	Abstract Workflow (AW) submitted to Condor's DAGMan
Kepler[23, 33]	parallel programming with native libraries	actor-oriented model with MoML
P-GRADE [32]	GT2, Condor	P-GRADE Workflow Language (DAG based)
SCIRun [34]	SDSC SRB, CoG	Specializes in dataflow visualization
Taverna [36, 29]	via Talisman toolkit that allows scriptable Web UI (not active)	Scufl language that is built to support scientific applications
Triana [37]	WSFL-like task graphs , Part of GridLab	GUI Tool to build UML model
Teuta [38]	GridLab	UML Activity Diagrams mapped to AGWL
Unicore [40]	UniGrid GPE, GT, PBS, CCS, SSH	DAGs with communication via AJOs (Abstract Job objects)

www-bpel4ws [5] also I need to take a look at SEDNA, Cactus is to be run prior and post execution of the condor job respectively. Other frameworks simply use a separate graph node to illustrate this separation. DAGMan and Condor are implemented in C. A Grid Services interface based on the GT4 standard is currently under development. It is not open source and extensions to this framework are therefore not possible by the general user community. The advantage of the system is the ease of specification of DAGs as ASCII files, the integration with the well known Condor software. The disadvantages are that no XML specification language was available, the system is closed source, and that it is more difficult to integrate your own customized schedulers and Grid mapping frameworks. No Graphical user interface is provided. Condor is well suited to interface to Grid backends.

4.2 Pegasus and Chimera

Chimera [13] and Pegasus [12, 31] are systems that work hand in hand to define a workflow model and to instantiate it within a Grid environment. The workflow model is centered on the concept of virtual data which specifies data as part of a number of transforms. Hence, the input to chimera can be expressed as partial workflow descriptions that specify input files on which logical transformations are applied, leading to output files. Instead of referring to an output file, the workflow designer can refer to the partial workflow that creates this output file. This concept is useful if the operations to obtain the modified data are cheaper than maintaining a new copy of the data. The specifications of the transformations are defined with the help of a virtual data language.Chimera is associated with tools and methods to define the virtual data workflow models and Pegasus is responsible for instantiating the workflows within a Grid environment. Pegasus supports the mapping of partial workflows into the Grid environment to allow late binding of the resources in order to adapt to the changing Grid environment during the workflow instantiation and execution. The advantage is that transformations may be cheaper than the storing of the result of transformations. The disadvantage is that the system to express such transformations can become quite complex and that intermediary data may not be available at a time it may be needed by other components.

4.3 Unicore

UNICORE (Uniform Interface to Computing Resources) [40, 41] is providing a seamless, secure, and intuitive access to distributed resources. It deals with job management including creation, submission, and monitoring, data management, application support, and single sign-on. It integrates resources as part of a meta computing environment. A graphical user interface is available through which the user can specify the programs to be executed on compute resources. As part of the UNICORE abstract job model, DAG-based workflows can be specified. In addition it contains conditionals and repetitive execution of job groups or tasks as part of the workflow model. The advantage of the system is that it is easy to define workflows to generate programs that work on remote machines.The disadvantage is that it is that the integration with the newest Grid standards is still in progress.

4.4 Triana

Triana [37] presents itself as a problem solving environment integrating a visual interface with data analysis tools. Triana workflows are based on the notion that every piece of work is part of a specific experiment and its execution needs to be coordinated as part of a workflow. Experiments are generated by a scientist which is supported by a Triana workflow management system to handle the experiments as efficiently and effectively as possible. As other workflow systems, Triana is based on a layered architecture that separates the visualization and representation from the instantiation of the workflow. Triana workflows are WSFL like representation of task graphs task graph consist of three types of elements: tasks, control links and data links, where a task represents an operation, a control link defines the sequence of tasks in the model, and a data link describes the flow of data between tasks. Triana is a arge software and contains a sophisticated visualization framework for workflow graphs. The advantage of Triana is the visualization framework which makes it possible to maintain moderately sized workflows in graphical fashion. The disadvantage is that the use may not as targeted to the Grid like other efforts such as Condor or Java CoG Kit.

4.5 SCIRUN

SCIRUN [34] is a scientific workbench that focuses on the presentation of components and their integration as part of a graphical user interface similar to AVS. It allows users to construct, manage, and debug scientific simulations in a variety of scientific disciplines. In SCIRUN, components are defined that can be connected with each other through the description of input and output relationships of the components. The assembly of such components can be viewed as a workflow since SCIRUN allows for parallel and conditional execution of tasks. SCIRUN has components integrated that allow running of the computational intense tasks on the Grid. Besides integration in Grids, distributed computing frameworks such as CORBA are also supported. It includes a sophisticated GUI workflow modeler as well as controls to modify a workflow while it is running. The advantage of SCIRUN is its component model and the ability to generate sophisticated visualization of scientific data. The disadvantage is that the user obtaines the power of SCIRUN through its user interface.

4.6 Kepler

Kepler [23] is a scientific workflow system that was built on top of Ptolemy II [33] developed at University of California Berkley. Kepler aims to streamline the process of workflow definition, monitoring and execution for scientists with emphasis on application-specific modules to aid them. The system is based on an actor-oriented modeling paradigm where actors correspond to re-usable workflow components that may be generic or specific to a particular domain. The system consists of several actors that provide access to remote data using web and grid services, serve as generic scientific workflow tools that are useful over many disciplines, are used to prototype workflows before execution, visualize the workflow and those that provide user interaction and communication for collaborative usage. The specification of the workflow is done using Keplers workflow modeling language MoML(Modeling Markup Language). The system is designed to be extensible with provisions for developers or end users to add actors that are required for their domain specific needs. They have addressed problems of scalability by running the jobs in parallel over a cluster and made provisions for long running workflows using detached execution of the workflow. The project currently is in collaboration with a variety of application projects such as SEEK(Science Environment for Ecological Knowledge), GEON(Infrastructure for geosciences), ROADNet(Realtime Observatories, Applications, and Data management Network project), SKIDL(SDSC Knowledge and Information Discovery Lab), distributed data integration with NLADR(National Laboratory of Advanced Data Research)

4.7 Taverna

Taverna [36] is a workflow bench designed as part of a UK e-Science pilot project: myGrid [29]. The focus of the project is on providing an environment to link together third party biology and bioinformatics applications (both remote and local) that are familiar to the scientist, using a language and tools designed for the scientist. The collaboration is directed towards issuing federated queries over distributed queries and consolidating the results for further analysis. The workflow workbench is used to execute data-intensive, in silico experiments in molecular biology. This enables the scientific user to create and run workflows written in their Simpliflied conceptual workflow language (Scufl) which presents a higher level view of workflows. These are executed using the Freefluo workflow enactment engine. Users can create workflows to store, retrieve and analyse data from varied data sources. Processors in Taverna refer to either the data resources or the data analysis tools. The advanced model explorer(AME) is used to load, edit and save workflows and functions as the primary visualization tool. It displays the workflow, metadata associated with it, services available, status of the workflow, renderers if possible, results browsing and saving of the results in different formats . Taverna's features include: support for iteration, browsing of results from the executed workflows, storage of provenance information, fault tolerance using failure handlers and reschedulers for failed tasks with other replicated processors. Taverna collaborates with numerous biology related services such as seqhound and BioMART that are biological datawarehouses, BioMoby which provides solutions for accessing distributed resources, BIOTeam Inquiry for grid in a box services.

4.8 Askalon

The Askalon [4, 15] projects' goal is to to simplify the development and optimization of applications that harness the power of Grid computing. Askalon provides an environment for workflow composition, specification, scheduling, resource management, performance monitoring and prediction. The composition is done using a graphical modeler to create a UML-based representation. The model can be transformed into an XML representation also by the Model traverser that investigates the possibility to traverse a path to every modeling element. The model checker is used to validate the model. The AGWL(Abstract Grid Workflow Language) is a XML-based language that is used to describe the workflow applications at a higher level without having to specify low-level grid middleware details. The resource manager, GridARM discovers resources and schedules jobs depending on the resource availability and job requirements. The scheduler uses the specified workflow to map resources to it after converting the workflow into simple DAGs(Directed Acyclic graphs). Performance analysis is done by instrumenting parameters such as communication efficiency, load imbalance, synchronization and scalability. This data is used by a Performance prediction service to predict execution times for workflows using past execution data for training it. Thus, the project attempts through tools, services, and methodologies to make Grid application development and optimization for applications an everyday practice.

5 Java CoG Kit Workflow

5.1 Workflow solutions of the Java CoG Kit

The Java CoG Kit provides a number of solutions to address the coordination of work within Grid applications. It includes the definition of simple APIs to access the Grid (jglobus and Grid Abstractions) [45, 2], a parameter study prototype [48], and a grid desktop prototype [44].

However, for this chapter we like to focus on the description of the Karajan Workflow framework that is part of the Java CoG Kit.

5.2 Karajan

Karajan obtained its name in analogy to a conductor at the Berlin opera. It symbolizes that the workflow framework is used to direct the grid tasks. Although, we have already the ability to distribute the task of directing amongst several services, this is however not reflected in the name.

5.2.1 Architecture

Figure 5 shows the high level architecture of Karajan. The structural language specification defines the nature and interaction of the basic building blocks of the language. The structural specification is syntax-less, allowing interchangeable bindings to both XML and a native syntax which provides certain convenience features not possible in XML (such as operators).

A set of libraries implement various functionality. The core libraries (Kernel and System) define the fundamental elements of the system (parallelism, variable manipulation, conditional execution, etc.). Other libraries interface with either the core libraries and/or various Java APIs in order to implement additional functionality. The Task Library interfaces with the Java CoG Kit Abstractions API which provides access to Grid middleware libraries. The Java library allows generic access to Java classes and methods directly from Karajan. The forms library, provided as an example, implements a XUL-like set of elements on top of Java Swing. The HTML library allows Karajan workflows to generate HTML code.

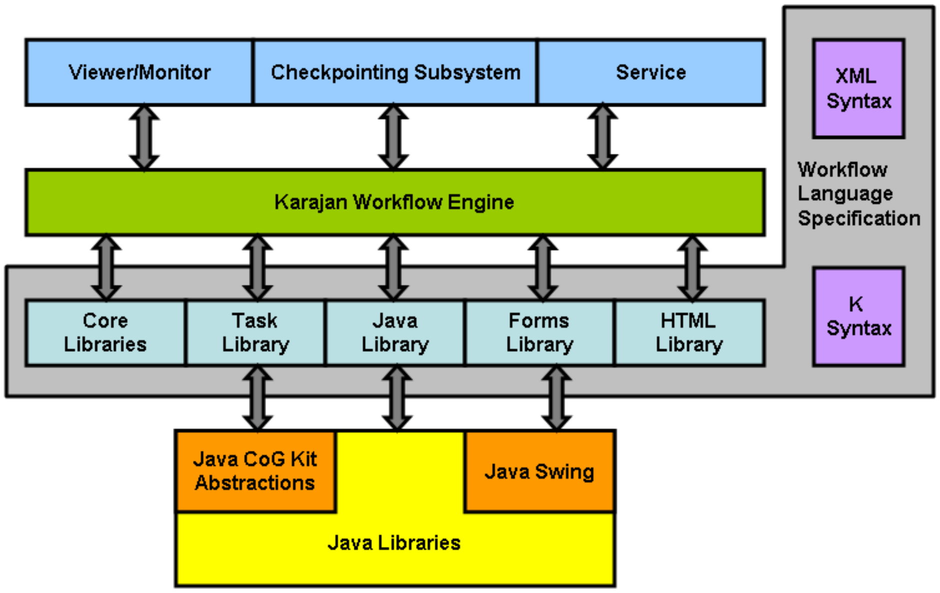


Figure 5: The Karajan Architecture

The execution engine uses a lightweight threading interpreter in order to execute elements defined in the libraries and provide the glue between elements that is specified by the language specification. The engine interacts with the various sub-systems that provide additional low-level functionality, such as the viewer, the checkpointing subsystem, and the service (although the execution engine does not depend on, and can function without any of the subsystems).

5.2.2 Language

The Karajan language is a declarative style parallel language. It tries to favor structured parallelism, both in terms of control flow and data flow. The basic unit in Karajan is the element. An element can take arguments and it can return values. An element can also fail. Each element is responsible for the way in which it evaluates its arguments. Although most elements evaluate their arguments in sequence, some do not. Although most elements evaluate their arguments before evaluating their definitions, some do not.

Moreover, through element composition, the strategy and evaluation order of arguments can be changed. Arguments generally consist of unitary data values. At a generic level no assumption is made on the number of values an element will return, or that return values will all be returned at the same time. This allows the flexible manipulation of the concurrency aspects of the workflow without compromising the data consistency. Furthermore, it allows certain concurrency patterns to be expressed in a simple manner, and applied to any of the available elements.

The Karajan language has a well defined and simple structure. It has no keywords. Instead, elements in various libraries implement all necessary functionality. The lack of keywords allows concurrency manipulation to be applied equally to system provided elements as well as user defined elements. Additionally, it allows the user to extend what otherwise would have been part of the syntax.

The Karajan system library contains a number of elements which implement conditional execution, iterations (both sequential and parallel), loops, error handling, some basic data types (list, map), and other features such as a discriminator which are lacking from many other workflow systems.

To show some of the high level constructs of the language we refer to figures 6 and 7. Here, a parallel element is used to transfer the standard output and standard error files in parallel to the local host. While in 6 we use our the XML syntax, we have also developed what we believe to be an easier to use syntax. Figure 7 shows this syntax that we have termed “K” and has the same structure as the XML syntax. More extensive examples can be found at [7].

5.2.3 Grid Integration

The Grid integration in Karajan is provided by the task library. The task library is built on top of the Java CoG Kit Abstractions API which provides a middleware agnostic view of common task patterns: job submission, file transfer and file operations.

This particular feature gives Karajan the possibility of describing late-binding Grid tasks, for which the exact resources and choice of middleware that is used for submission can be decided at run-time, based on up-to-date availability information on an individual basis. It also enables interoperability between different Grid services. A scheduler can be used in order to provide the strategy of mapping tasks to resources.

5.2.4 Karajan Service

The Karajan Service can be used to expose the Karajan engine on remote resources. An attempt at providing a pictorial representation of the inter-

```
<project>
  <include file="cogkit.xml"/>
  <task:execute executable="/bin/ls" arguments="-al"
                stdout="stdout" stderr="stderr"
                host="hot.mcs.anl.gov" provider="GT2"/>
  <echo message="Job completed. Transferring stdout and stderr"/>

  <parallel>
    <task:transfer srchost="hot.mcs.anl.gov" srcfile="stdout"
                   desthost="localhost" provider="gridftp"/>
    <task:transfer srchost="hot.mcs.anl.gov" srcfile="stderr"
                   desthost="localhost" provider="gridftp"/>
  </parallel>
  <echo message="Stdout and stderr transferred"/>
</project>
```

Figure 6: One way of expressing concurrency in Karajan is to use it in an explicit declarative fashion through the sequential and parallel tags.

```
project (
  include("cogkit.k")
  task:execute("/bin/ls", arguments="-al", stdout="stdout",
    stderr="stderr", host="hot.mcs.anl.gov", provider="GT2")
  echo("Job completed. Transferring stdout and stderr")

  parallel(
    task:transfer(srchost="hot.mcs.anl.gov", srcfile="stdout",
      desthost="localhost", provider="gridftp")
    task:transfer(srchost="hot.mcs.anl.gov", srcfile="stderr",
      desthost="localhost", provider="gridftp")
  )
  echo("Stdout and stderr transferred")
)
```

Figure 7: Through the K-language we have an equivalent description that is less wordy and can be easier read.

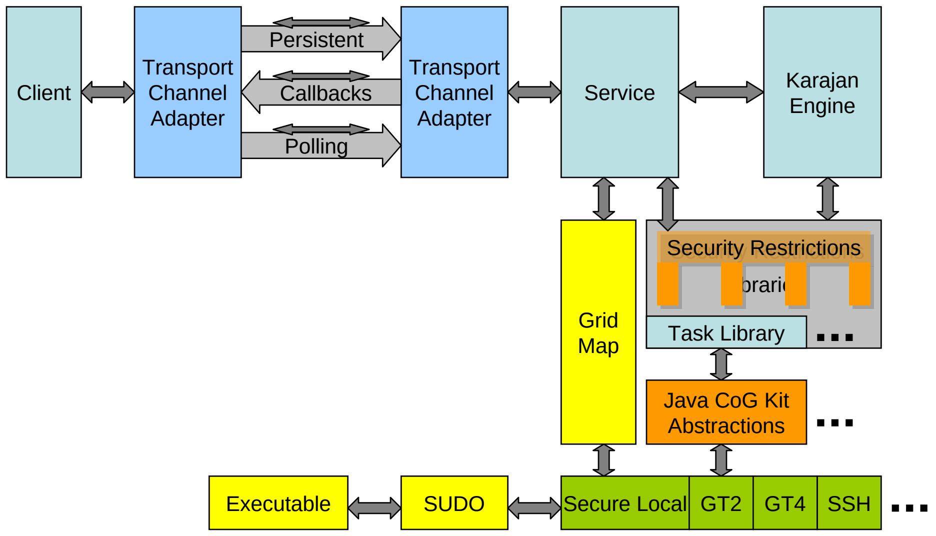


Figure 8: The Karajan Service Architecture

action of the components that are relevant in using the service is shown in Figure 8.

A client communicates with the service through a transport library. The transport library features configurable communication channels, allowing it to adapt to different usage scenarios. Persistent connections, callbacks and polling can be used and even combined on a host or domain basis. There are certain advantages and disadvantages to each strategy. For example, persistent connections will be efficient and allow the client to work from behind a firewall, but they will consume resources even if the connection is idle. By contrast, callbacks are more resource-efficient, but they will not work properly if the client is behind a firewall that does not allow incoming connections. Lastly, polling can work if the client is behind a firewall, the resource usage is slightly higher than that of callbacks, but it introduces a latency that is dependent on the polling interval. The transport library can use a single connection for multiple concurrent submissions.

Transport level security and authentication is provided by the use of GSI connections. Simple authorization is provided by a grid-map file, commonly used within the Globus Toolkit.

The use of the service places certain restrictions on what workflow elements are available remotely. Since the service can be used by multiple users concurrently, elements that could be used to gain privileges not normally available are not accessible. Such elements include direct file-system access elements and elements in the Java library.

Local executables are accessible through a special provider that uses a grid-map file and (on Unix-like systems) SUDO. Consequently arbitrary executables will be run under specific user accounts indicated by the mapping in the grid-map file.

5.2.5 Repository

In order to manage components dynamically within the workflow, we also provide a workflow component repository [47]. This service is used to store, retrieve, and search for components that can be integrated into a Java CoG Kit workflow. The repository service promotes reusability of components that can either be maintained by an individual researcher, or by a shared community of peers with similar interests.

6 Conclusion

In this chapter we have given a small overview about the coordination of work within a Grid. We have identified different paradigms to support it. Important to recognize is that at present Grid workflow is an active area of research. It is also important to know that none of the existent workflow frameworks provide all the answers. Systems such as the Java CoG Kit, Kepler, and Taverna all provide their own strength and weaknesses. It is important to recognize that at times workflow systems that strongly support a visual representation may in many cases provide too complex of an interface. We still need high level interfaces supported through programming languages and message passing systems. BPEL in itself does not provide a solution either as its specifications are rather complex. BPEL is also a low level component and should not directly use by programmers. Higher level tools are necessary in order to guide the users to design complex Grid workflows.

Acknowledgement

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, NSF NMI, NSF DDDAS.

References

[1] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock. Kepler: An Extensible System for Design and Execution of Scientific Workflows. In *16th Intl. Conference on Scientific and Statistical Database Management (SSDBM)*, page 423. IEEE Computer Society, 2004. Available from: `http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/ssdbm/{\&}toc=comp/proceedings/ssdbm/2004/2146/00/2146toc.xml{\&}DOI=10.1109/SSDM.2004.1311241#additionalInfo`.

[2] Kaizar Amin, Mihael Hategan, Gregor von Laszewski, and Nestor J. Zaluzec. Abstracting the Grid. In *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2004)*, pages 250–257, La Coruña, Spain, 11-13 February 2004. Available from: `http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--abstracting.pdf`.

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

[4] Askalon Programming Environment for Grid Computing. Available from: `http://www.dps.uibk.ac.at/projects/askalon/`.

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

[6] The Chimera Virtual Data Systsm. Web Page. Available from: http://www.griphyn.org/chimera/.

[7] Java CoG Kit Documentation. Available from: http://wiki.cogkit.org/index.php/Java_CoG_Kit_Documentation.

[8] Condor: High Throughput Computing. Web Page. Available from: http://www.cs.wisc.edu/condor/.

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

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

[11] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman. Pegasus: Planning for Execution in Grids. Technical Report TR-2002-20, ISI, UCLA, November 2002. Available from: http://www.isi.edu/~deelman/pegasus.htm.

[12] 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.

[13] 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.

[14] DiscoveryNet. Web Page, 2003. Available from: http://www.discovery-on-the.net.

[15] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek. Askalon: A grid application development and computing environment. In *6th International Workshop on Grid Computing*. IEEE Computer Society Press, November 2005.

[16] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, July 1998.

[17] Grid Service Broker. Web Page. Available from: http://www.gridbus.org/broker/.

[18] GridNexus. Web Page. Available from: http://www.gridnexus.org/.

[19] Pei Hao, Yuan-Yuan Li, Wei zhong He, and Yi-Xue Li. Studying the molecular evolution of the sars-coronavirus on the the discoverynet environment, 2004. Available from: http://www.jsbi.org/journal/GIW04/GIW04S08.pdf.

[20] ICENI - Imperial College e-Science Networked Infrastructure. Web Page. Available from: http://www.lesc.ic.ac.uk/iceni/.

[21] I-lab. Web Page. Available from: http://www.fhrg.fhg.de/index_en.html.

[22] Java CoG Kit Karajan Guide. Web Page. Available from: http://www.cogkit.org/current/manual/workflow.pdf.

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

[24] Leonard Kleinrock. UCLA to Build The First Station in Nationwide Computer Network. Press Release, 1969. Available from: http://www.lk.cs.ucla.edu/LK/Bib/REPORT/press.html.

[25] Sriram Krishnan, Patrick Wagstrom, and Gregor von Laszewski. GSFL: A Workflow Framework for Grid Services. In *Preprint ANL/MCS-P980-0802*, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, U.S.A., 2002. Available from: http://www-unix.globus.org/cog/papers/gsfl-paper.pdf.

[26] Load Sharing Facility. Web Page, Platform Computing, Inc. Available from: http://www.platform.com/.

[27] MAUI Scheduler. Available from: http://mauischeduler.sourceforge.net.

[28] Moab Scheduler. Available from: http://supercluster.org/moab.

[29] myGrid. Web Page. Available from: http://mygrid.man.ac.uk/.

[30] Portable Batch System. Web Page, Veridian Systems. Available from: http://www.openpbs.org/.

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

[32] P-Grade. Web Page. Available from: http://www.lpds.sztaki.hu/pgrade/.

[33] Ptolemy II. Web Page. Available from: http://ptolemy.eecs.berkeley.edu/ptolemyII/.

[34] SciRun. Web Page. Available from: http://software.sci.utah.edu/scirun.html.

[35] gridengine: Home. Available from: http://gridengine.sunsource.net.

[36] myGrid Taverna Workbench. Web Page. Available from: http://www.mygrid.org.uk/index.php?module=pagemaster&PAGE_user_op=view_page&PAGE_id=44&MMN_position=53:51:52.

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

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

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

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

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

[42] Gregor von Laszewski. Grid Computing: Enabling a Vision for Collaborative Research. In Juha Fagerholm, Juha Haataja, Jari Järvinen, Mikko Lyly, Peter Raback, and Ville Savolainen, editors, *The Sixth International Conference on Applied Parallel Computing*, volume 2367 of *Lecture Notes in Computer Science*, pages 37–52, Espoo, Finland, 15-18 June 2002. Springer. *(Invited Talk)*. Available from: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--para4.pdf.

[43] Gregor von Laszewski and Kaizar Amin. *Grid Middleware*, chapter Middleware for Communications, pages 109–130. Wiley,

2004. Available from: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--grid-middleware.pdf.

[44] Gregor von Laszewski, Matthew W. Bone, Ishrath Fatima, Mikhail Sosonkin, Robert Winch, Nithya N. Vijayakumar, Pankaj Sahasrabudhe, Kaizar Amin, Mihael Hategan1, Jonathan DiCarlo, and David Angulo. Towards the Development of a Bioinformatics Grid Desktop. Preprint ANL/MCS-ANL/MCS-P1189-0804, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, U.S.A., August 2004. in partial fulfillment of the REU 2004 Site on Grid Computing and Bioinformatics. Available from: http://www.cogkit.org.

[45] Gregor von Laszewski, Ian Foster, Jarek Gawor, and Peter Lane. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13(8-9):643–662, 2001. Available from: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--cog-cpe-final.pdf.

[46] Gregor von Laszewski, Ian Foster, George K. Thiruvathukal, and Brian Toonen. A Computational Framework for Telemedicine. *Journal of Future Generation Computer Systems*, 14:10–123, 1998. Available from: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--telemed.pdf.

[47] 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.

[48] Gregor von Laszewski, Tan Trieu, Phillip Zimny, and David Angulo. The Java CoG Kit Experiment Manager. Technical report, Argonne National Laboratory, June 2005. Available from: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-exp.pdf.

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

[50] Gregor von Laszewski, Nestor Zaluzec, Mihael Hategan, Kaizar Amin, Shawn Hampton, and Al Rossi. GridAnt: Client Side Workflow Management in Grids (with application to Position Resolved Diffraction). In *Midwest Software Engineering Conference*, page 193, Chicago, June 5th 2003. DePaul University.

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

Copyright