

# A Taxonomy of Workflow Management Systems for Grid Computing

Jia Yu and Rajkumar Buyya<sup>\*</sup>

*Grid Computing and Distributed Systems (GRIDS) Laboratory, Department of Computer Science and Software Engineering, The University of Melbourne, Melbourne, Australia*  
E-mail: raj@cs.mu.oz.au

Received 28 May 2005; accepted in revised form 6 December 2005

*Key words:* Grid computing, resource management, scheduling, taxonomy, workflow management

## Abstract

With the advent of Grid and application technologies, scientists and engineers are building more and more complex applications to manage and process large data sets, and execute scientific experiments on distributed resources. Such application scenarios require means for composing and executing complex workflows. Therefore, many efforts have been made towards the development of workflow management systems for Grid computing. In this paper, we propose a taxonomy that characterizes and classifies various approaches for building and executing workflows on Grids. We also survey several representative Grid workflow systems developed by various projects world-wide to demonstrate the comprehensiveness of the taxonomy. The taxonomy not only highlights the design and engineering similarities and differences of state-of-the-art in Grid workflow systems, but also identifies the areas that need further research.

## 1. Introduction

Grids [51] have emerged as a global cyber-infrastructure for the next-generation of e-Science applications by integrating large-scale, distributed and heterogeneous resources. Scientific communities, such as high-energy physics, gravitational-wave physics, geophysics, astronomy and bioinformatics, are utilizing Grids to share, manage and process large data sets. In order to support complex scientific experiments, distributed resources such as computational devices, data, applications, and scientific instruments need to be orchestrated while managing the application workflow operations within Grid environments [92].

Workflow is concerned with the automation of procedures whereby files and data are passed between participants according to a defined set of rules

to achieve an overall goal [35]. A workflow management system [5] defines, manages and executes workflows on computing resources. Imposing the workflow paradigm for application composition on Grids offers several advantages [117] such as:

- Ability to build dynamic applications which orchestrate distributed resources.
- Utilization of resources that are located in a particular domain to increase throughput or reduce execution costs.
- Execution spanning multiple administrative domains to obtain specific processing capabilities.
- Integration of multiple teams involved in managing of different parts of the experiment workflow – thus promoting inter-organizational collaborations.

Figure 1 shows the architecture and functionalities supported by various components of the Grid workflow system based on the workflow reference model

<sup>\*</sup> Corresponding author.

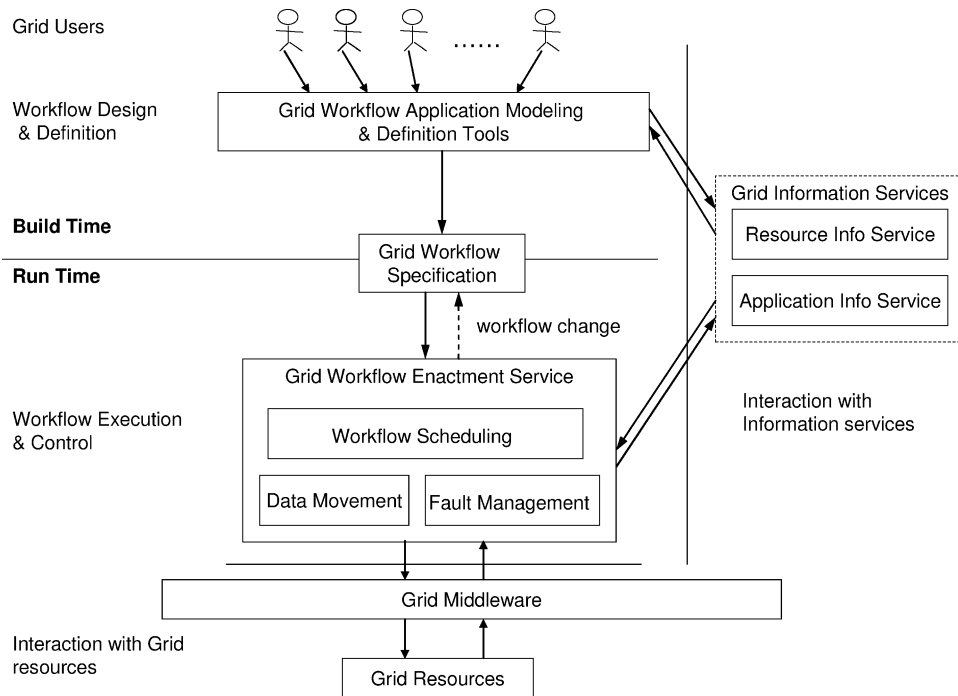


Figure 1. Grid workflow management system.

[35] proposed by Workflow Management Coalition (WfMC) [137] in 1995. At the highest level, functions of Grid workflow management systems could be characterized into *build time* functions and *run time* functions. The build-time functions are concerned with defining, and modeling workflow tasks and their dependencies; while the run-time functions are concerned with managing workflow executions and interactions with Grid resources for processing workflow applications. Users interact with workflow modeling tools to generate a workflow specification, which is submitted to a run-time service called the workflow enactment service for execution. Major functions provided by the workflow enactment service are scheduling, fault management and data movement. The workflow enactment service may be built on the top of low level Grid middleware (e.g., Globus toolkit [59], UNICORE [128] and Alchemi

[86]), through which the workflow management system invokes services provided by Grid resources. At both the build-time and run-time stages, the information about resources and applications may need to be retrieved using Grid information services.

In the recent past, several Grid workflow systems [112] have been proposed and developed for defining, managing and executing scientific workflows. In order to enhance our understanding of the field, we propose a taxonomy that primarily (a) captures architectural styles and (b) identifies design and engineering similarities and differences between them. There are a number of proposed taxonomies for distributed and heterogeneous computing such as [20, 29, 73, 108]. However, none of these focuses on distributed workflow managements. The taxonomy provides an in-depth understanding of building and executing

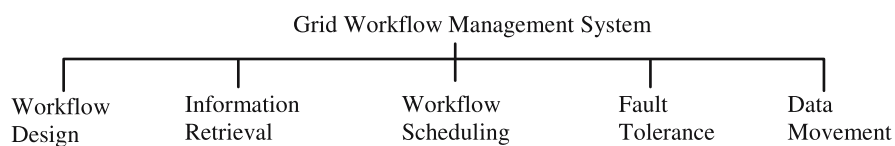


Figure 2. Elements of a Grid workflow management system.

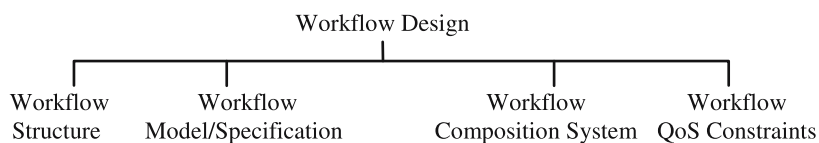


Figure 3. Workflow design taxonomy.

workflows on Grids. It compares different approaches and also helps users to decide on minimum subset of features required for their systems.

The rest of the paper is organized as follows: Section 2 presents the taxonomy that classifies approaches based on major functions and architectural styles of Grid workflow systems. In Section 3, we provide a detailed survey of several selected Grid workflow systems and the mapping of the proposed taxonomy to the systems. We conclude in Section 4 with a discussion and identification of areas that need further work.

## 2. Taxonomy

The taxonomy characterizes and classifies approaches of workflow management in the context of Grid computing. As shown in Figure 2, it consists of five elements of a Grid workflow management system: (a) workflow design, (b) information retrieval, (c) workflow scheduling, (d) fault tolerance and (e) data movement. In this section, we look at each element and its taxonomy in detail.

### 2.1. Workflow Design

As shown in Figure 3, workflow design includes four key factors, namely (a) workflow structure, (b) workflow model/specification, (c) workflow composition system, and (d) workflow QoS (Quality of Service) constraints.

#### 2.1.1. Workflow Structure

A workflow is composed by connecting multiple tasks according to their dependencies. The workflow structure, also referred as workflow pattern [2, 3, 6], indicates the temporal relationship between these tasks. Figure 4 shows the workflow structure taxonomy. In general, a workflow can be represented as a *Directed Acyclic Graph (DAG)* [110] or a *non-DAG*.

In DAG-based workflow, workflow structure can be classified as *sequence*, *parallelism*, and *choice*. Sequence is defined as an ordered series of tasks, with one task starting after a previous task has completed. Parallelism represents tasks which are performed concurrently, rather than serially. In choice control pattern, a task is selected to execute at runtime when its associated conditions are true.

In addition to all patterns contained in a DAG-based workflow, a non-DAG workflow also includes the *iteration* structure in which sections of workflow tasks in an iteration block are allowed to be repeated. Iteration is also known as *loop* or *cycle*. The iteration structure is quite frequently used in scientific applications, where one or more tasks need to be executed repeatedly [91]. For example, in a promoter identification workflow [85] as shown in Figure 5, step 5 to step 8 are executed iteratively to create and refine a promoter model.

These four types of workflow structure, namely sequence, parallelism, choice and iteration, can be used to construct many complex workflows. Moreover, sub-workflows can also use these types of workflow structure as building blocks to form a large-scale workflow.

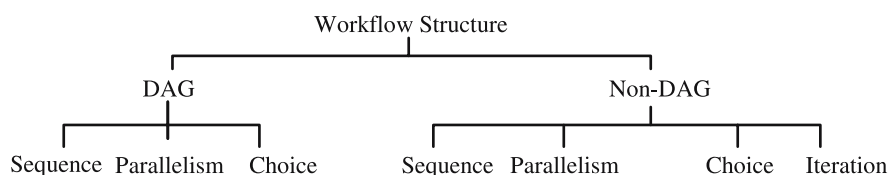


Figure 4. Workflow structure taxonomy.

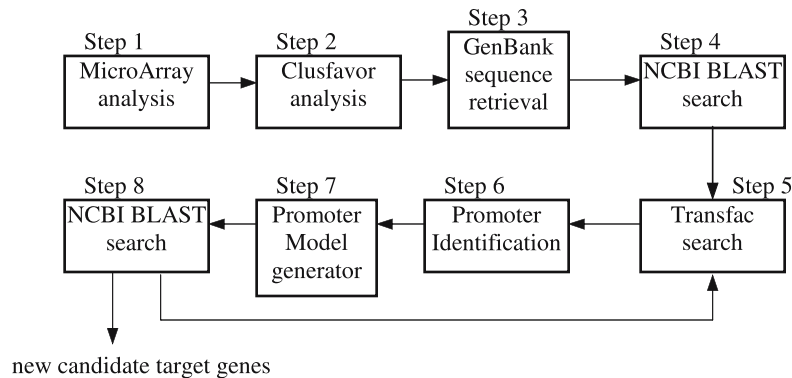


Figure 5. Promoter identification workflow [85].

### 2.1.2. Workflow Model/Specification

Workflow Model (also called workflow specification) defines a workflow including its task definition and structure definition. As shown in Figure 6, there are two types of workflow models, namely *abstract* and *concrete*. They are also referred to as abstract workflows and concrete workflows [40, 42]. In some literature (e.g., [84]), concrete models are referred to as executable workflows.

In an abstract model, a workflow is described in an abstract form in which the workflow is specified without referring to specific Grid resources for task execution. An abstract model provides a flexible way for users to define workflows without being concerned about low-level implementation details. Tasks in an abstract model are portable and can be mapped onto any suitable Grid services at run-time by using suitable discovery and mapping mechanisms. Using abstract models also eases the sharing of workflow descriptions between Grid users [42]; in particular it benefits the participants of Virtual Organizations (VOs) [52].

In contrast, a concrete model binds workflow tasks to specific resources. In some cases, a concrete model may include tasks acting as data movement to transfer data in and out of the computation and data publication to publish newly derived data into VO [42]. In other situations, tasks in a concrete model

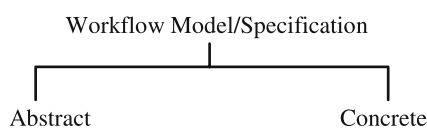


Figure 6. Workflow model taxonomy.

may also include necessary application movement to transfer computational code to a data site for large scale data analysis.

Given the dynamic nature of the Grid environment, it is more suitable for users to define workflow applications in abstract models. A full or partial concrete model can be generated just before or during workflow execution according to the current status of resources. Additionally, in some systems [144], every task in a workflow is concretized only at the time of task execution. However, concrete models may be used by some end users who want to control the execution sequence [75].

### 2.1.3. Workflow Composition System

Workflow composition systems are designed for enabling users to assemble components into workflows. They need to provide a high level view for the construction of Grid workflow applications and hide the complexity of underlying Grid systems. Figure 7 shows the taxonomy for the workflow composition systems. *User-directed* composition systems allow users to edit workflows directly, whereas *automatic* composition systems generate workflows for users automatically. In general, users can use workflow languages for *language-based modeling* and the tools for *graph-based modeling* to compose workflows.

Within language-based modeling, users may express workflow using a *markup* language such as Extensible Markup Language (XML) [132] (e.g., GridAnt [75], WSFL [79], XLANG [125], BPEL4WS [14], W3C XML-Pipeline language [135], and Gridbus workflow [144]) or other formats (e.g., Condor DAGman [120]). Language-based

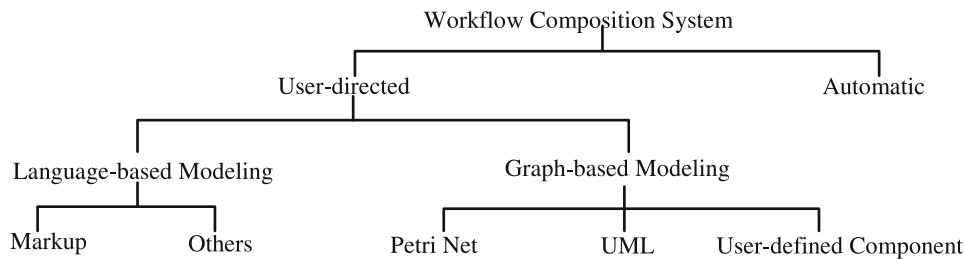


Figure 7. Workflow composition system taxonomy.

modeling may be convenient for skilled users, but they require users to memorize a lot of language-specific syntax. In addition, it is impossible for users to express a complex and large workflow by scripting workflow components manually. However, workflow languages are more appropriate for sharing and manipulation, whereas the graphical representations are intuitive but they require to be converted into other forms for manipulation. So in most Grid systems, workflow languages are designed to bridge the gap between the graphical clients and the Grid workflow execution engine [62]. XML-based languages are used widely for workflow specification as it facilitates information description in a nested structure. Moreover, many tools are provided to validate XML syntax and verify XML documents against XML schema [134] or DTD (Document Type Definition) [132]. Furthermore, many XML parsing tools (e.g., JDOM [69] and dom4j [44]) are widely available.

Graph-based modeling allows graphical definition of an arbitrary workflow through a few basic graph elements. It allows users to work with a graphical representation of the workflow. Users can compose and review a workflow by just clicking and dropping the components of interest. It avoids low-level details and hence enables users to focus on higher levels of abstraction at application level [64]. The major modeling approaches are *Petri Nets* [97, 104], *UML* (Unified Modeling Language) [99] and *user-defined component*. Graph-based modeling is preferred by users as opposed to language-based modeling.

Petri Nets are a special class of directed graphs that can model sequential, parallel, loops and conditional execution of tasks [62, 65]. They have been used in many workflow management systems such as Grid-Flow [62], FlowManager [78], and XRL/Flower [131]. UML activity diagrams [102] have also been extended and applied as a workflow specification

language [17, 45, 105]. Compared with UML activity diagrams, Petri Nets have formal semantics and have been used widely for constructing several workflows [1, 46]. A vast number of algorithms and tools for Petri Nets analysis have been developed along the years [89]. However, Eshuis et al. [46] argue that Petri Nets may be unable to model workflow activities accurately without extending its semantics and this drawback has been addressed in UML activity diagrams. Rather than following the standard syntax and semantics of Petri Nets and UML, many workflow editors for Grid workflow tools create their own graphical representation of workflow components. For example, Triana [123] allows users to predefine software components and reuse them to design DAG-based workflows. Kepler [12] provides graphical environment and a framework that supports the design and reuse of Grid workflows. These tools are more convenient for users to manipulate their workflow applications, as they provide a more user-friendly programming environment. They have also been integrated into underlying local applications, Grid middleware and monitoring systems. For example, P-GRADE [71, 83] interoperates with a wide range of parallel applications in addition to Condor and Globus based Grid middleware. It also allows users to access and modify program code of a workflow task through a graphical editor. However, lack of standards hinders the collaboration between these projects. Many works are thus replicated such as different user interfaces developed by different projects for the same functionality. Moreover, workflow structures supported by most of them are limited to only sequence and parallelism.

Graph-based modeling is very intuitive and can be handled easily even by a non-expert user. However, the layout of workflow components on a display screen can become very huge and difficult to manage [101]. One of the solutions to overcome this

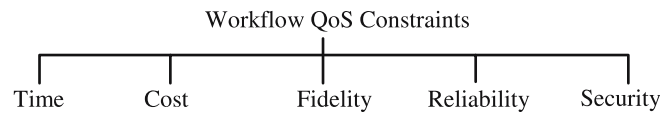


Figure 8. Workflow QoS constraints taxonomy.

limitation is to use hierarchical graph definition [65]. Another solution is to have a system which composes workflows automatically. Pegasus [42] is one such automatic composition system for Grid computing; it has to be adapted to particular applications, because the composition is based on application-dependent metadata. It receives a metadata description of desired data products and initial input values from users. The tasks are then composed automatically to form a workflow by querying a virtual data catalog [53] that contains information for data derivation of application components. Compared with user-directed systems, automatic composition systems are ideal for large scale workflows which are very time consuming to compose manually. However, the automatic composition of application components is challenging because it is difficult to capture the functionality of components and data types used by the components [27, 101].

#### 2.1.4. Workflow QoS Constraints

In a Grid environment, there are a large number of similar or equivalent resources provided by different parties. Grid users can select suitable resources and use them for their workflow applications. These resources may provide the same functionality, but optimize different QoS measures. In addition, different users or applications may have different expectations and requirements. Therefore, it is not sufficient for a workflow management system to only consider functional characteristics of the workflow. QoS requirements such as time limit (deadline) and expenditure limit (budget) for workflow execution also need to be managed by workflow management systems. Users must be able to specify their QoS expectations of the workflow at the design level. Then, the actions conducted by workflow systems using run-time must be chosen according to the initial QoS requirements.

Figure 8 shows the taxonomy of Grid workflow QoS constraints based on a QoS model for Web services based workflow provided by Cardoso et al. [28] and QoS of Web services [88, 103]. It includes five dimensions: *time*, *cost*, *fidelity*, *reliability* and *security*. Time is a basic measure of performance.

For workflow systems, it refers to the total time required for completing the execution of a workflow. Cost represents the cost associated with the execution of workflows including the cost for managing workflow systems and usage charge of Grid resources for processing workflow tasks. Fidelity refers to the measurement related to the quality of the output of workflow execution. Reliability is related to the number of failures for execution of workflows. Security refers to confidentiality of the execution of workflow tasks and trustworthiness of resources.

As indicated in Figure 9, there are two different ways to assign QoS constraints in a workflow model. One way is to allow users to assign QoS constraints at *task-level*. The overall QoS can be assessed by computing all individual tasks. For example, a user assigns desired execution time for every task in a workflow. The deadline for the entire workflow execution can be calculated by a workflow reduction algorithm (e.g., SWR(w) algorithm [26]). Another way is to assign QoS constraints at *workflow-level*, allowing users to define the overall workflow QoS requirements. However, QoS constraints for each task may be required by schedulers for resource allocation at run-time. For the time dimension, users are likely to specify a deadline for the entire workflow execution rather than for every single task. In order to fulfill the deadline for the entire workflow, the scheduler needs to decide how fast each task has to be processed using a deadline assignment approach (e.g., Ultimate Deadline, Effective Deadline, Equal Slack, and Equal Flexibility strategies in [72]).

## 2.2. Information Retrieval

A Grid workflow management system does not execute the tasks itself, but it merely coordinates the

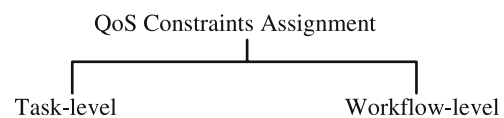


Figure 9. QoS constraints assignment taxonomy.

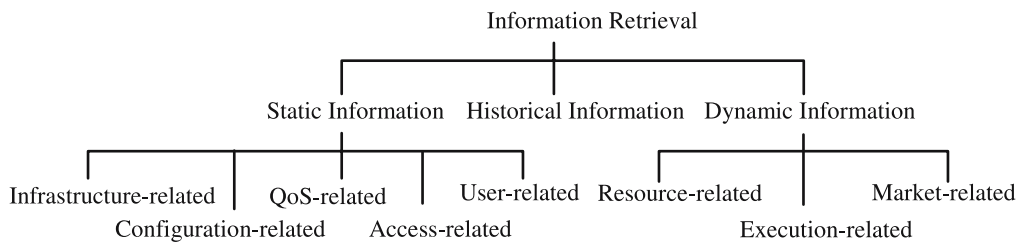


Figure 10. Information retrieval taxonomy.

execution of the tasks by the Grid resources. To map tasks onto suitable resources, information about the resources has to be retrieved from appropriate sources [141]. As indicated in Figure 10, there are three dimensions of information retrieval: *static information*, *historical information* and *dynamic information*.

Static information refers to information that does not vary with time. It may include *infrastructure-related* (e.g., the number of processors), *configuration-related* (e.g., operating system, libraries), *QoS-related* (e.g., flat usage charge), *access-related* (e.g., service operations), and *user-related* information (e.g., authentication ID). Generally, static information is utilized by Grid workflow management systems to pre-select resources during the initiation of the workflow execution.

As Grid resources are not dedicated to the owners of the workflow management systems, the Grid workflow management system also needs to identify dynamic information such as resource accessibility, system workload, and network performance during execution time. Unlike static information, dynamic information reflects the status of the Grid resources, such as load average of a cluster, available disk space, CPU usage, and active processes. It also includes task execution information and market related information such as dynamic resource price.

Historical information is obtained from previous events that have occurred such as performance history and execution history of Grid resources and application components. Generally, workflow management systems can analyze historical information to predict the future behaviors of resources and application components on a given set of resources. Historical information can also be used to improve the reliability of future workflow execution. For example, the user can correct the logic of a failed workflow according to the log of the workflow system.

Several information services are available for accessing static and dynamic information about Grid

resources. For example, Monitoring and Discovery System (MDS) [109] provides static hardware information such as CPU type, memory size and software information such as operating system information, and some dynamic information such as CPU load snapshot. Network Weather Service (NWS) [136] provides additional dynamic information about availability of CPU, memory, and bandwidth. An object oriented model for publication and retrieval of electronic resources is given in [33].

### 2.3. Workflow Scheduling

Casavant et al. [29] categorized task scheduling in distributed computing systems into ‘local’ task scheduling and ‘global’ task scheduling. Local scheduling involves handling the assignment of tasks to time-slices of a single resource whereas global scheduling involves deciding where to execute a task. According to this definition, workflow scheduling is a kind of global task scheduling as it focuses on mapping and managing the execution of inter-dependent tasks on shared resources that are not directly under its control.

The workflow scheduler needs to coordinate with diverse local management systems as Grid resources are heterogeneous in terms of local configuration and policies. Taking into account users’ QoS constraints is also important in the scheduling process so as to satisfy user requirements. In this section, we discuss workflow scheduling taxonomy from the view of (a) scheduling architecture, (b) decision making, (c) planning scheme, (d) scheduling strategy, and (e) performance estimation as shown in Figure 11.

#### 2.3.1. Scheduling Architecture

The architecture of the scheduling infrastructure is very important for scalability, autonomy, quality and performance of the system [63]. Three major categories of workflow scheduling architecture as shown

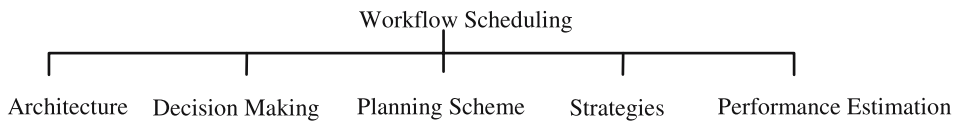


Figure 11. Workflow scheduling taxonomy.

in Figure 12 are centralized, hierarchical and decentralized scheduling schemes.

In a *centralized* workflow enactment environment, one central workflow scheduler makes scheduling decisions for all tasks in the workflow. The scheduler has the information about the entire workflow and collects information of all available processing resources. It is believed that the centralized scheme can produce efficient schedules because it has all necessary information [63]. However, it is not scalable with respect to the number of tasks, the classes and number of Grid resources. It is thus only suitable for a small scale workflow or a large scale workflow in which every task has the same objective (e.g., same class of resources).

Unlike centralized scheduling, both *hierarchical* and *decentralized* scheduling allow tasks to be scheduled by multiple schedulers. Therefore, one scheduler only maintains the information related to a sub-workflow. Thus, compared to *centralized* scheduling, they are more scalable since they limit the number of tasks managed by one scheduler. However, the best decision made for a partial workflow may lead to sub-optimal performance for the overall workflow execution. Moreover, conflict problems are more severe [90]. One example of conflict is that tasks from different sub-workflows scheduled by different schedulers may compete for the same resource.

For *hierarchical* scheduling, there is a central manager and multiple lower-level sub-workflow schedulers. This central manager is responsible for controlling the workflow execution and assigning the sub-workflows to the low-level schedulers. For example, in GridFlow project [25], there is one workflow manager and multiple lower-level sched-

ulers. The workflow manager schedules sub-workflows onto corresponding lower-level schedulers. Each lower-level scheduler is responsible for scheduling tasks in a sub-workflow onto resources owned by one organization. The major advantage of using the hierarchical architecture is that the different scheduling policies can be deployed in the central manager and lower-level schedulers [63]. However, the failure of the central manager will result in entire system failure.

In contrast, there are multiple schedulers without a central controller in decentralized scheduling. Every scheduler can communicate with each other and schedule a sub-workflow to another scheduler with lower load. Compared to hierarchical scheduling, decentralized scheduling is more scalable but faces more challenges to generate optimal solutions for overall workflow performance and minimize conflict problems.

### 2.3.2. Decision Making

There is no single best solution for mapping workflows onto resources for all workflow applications, since the applications can have very different characteristics. It depends to some degree on the application models to be scheduled. In general, decisions about mapping tasks in a workflow onto resources can be based on the information of the current task or of the entire workflow and can be of two types, namely *local* decision and *global* decision [40] as shown in Figure 13. Scheduling decisions made with reference to just the task or sub-workflow at hand are called local decisions whereas scheduling decisions made with reference to the whole workflow are called global decisions.

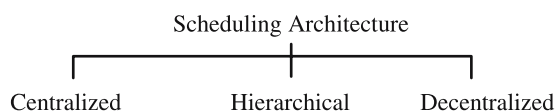


Figure 12. Scheduling architecture taxonomy.

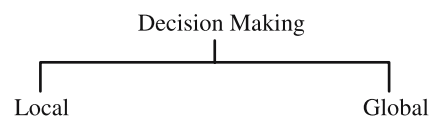


Figure 13. Decision making taxonomy.

Local decision based scheduling only takes one task or sub-workflow into account, so it may produce the best schedule for the current task or sub-workflow but could also reduce the entire workflow performance. An example given by Deelman et al. [40] assumes that there is a data-intensive application where the overall run-time is driven by data transfer costs. Consider a situation where the output of a task is very large. If the selection of a resource for a task is based only on a local decision without consideration of data transfer between other resources, when selection of a resource for child tasks need to be made, the initial selection may be found to be a poor choice if latency between the nodes is very high. This would lead to higher data transfer costs for this child task and hence the entire workflow.

Scheduling workflow tasks using global decision improves the performance of entire workflow. There are some algorithms for scheduling task graphs in parallel systems that could be applied to Grid workflow scheduling. Li et al. [80] developed the Forward-Looking Analysis Method (FLAM). It analyses dependencies of the entire graph to resolve the conflicts of parallel tasks which compete for the same resource. It is believed that global decision based scheduling can provide a better overall result. However, it may take much more time in scheduling decision making. Thus, the overhead produced by global scheduling could reduce the overall benefit and may even exceed the benefits it will produce [40]. Therefore, the choice of decision making for workflow scheduling should not be made without considering balance between the overall execution time and scheduling time. However, for some applications such as a data analysis application where the outputs of tasks in the workflow are always smaller than the inputs, using local decision based scheduling is sufficient.

### 2.3.3. Planning Scheme

A planning scheme is a method for translating abstract workflows to concrete workflows. As shown in Figure 14, schemes for the schedule planning of workflow applications can be categorized into either *static* scheme or *dynamic* scheme. In a static scheme, concrete models have to be generated before the execution according to current information about the execution environment and the dynamically changing state of the resources is not taken into account. In contrast, a dynamic scheme uses both dynamic information and static information about resources to make scheduling decisions at run-time.

Static schemes, also known as *full-ahead* planning, include *user-directed* and *simulation-based* scheduling. In user-directed scheduling, users emulate the scheduling process and make resource mapping decisions according to their knowledge, preference and/or performance criteria. For example, users prefer to map tasks to resources on which they have not experienced failures. In simulation-based scheduling, the ‘best’ schedule is achieved by simulating task execution on a given set of resources before a workflow starts execution. The simulation can be processed based on static information or the result of performance estimation. For example, in GridFlow [25], the ‘best’ resource selected for scheduling a task is based on the predictive task execution time that resource provides.

Dynamic schemes include *prediction-based* and *just in-time* scheduling. Prediction-based dynamic scheduling uses dynamic information in conjunction with some results based on prediction. It is similar to simulation-based static scheduling, in which the scheduler is required to predict the performance of task execution on resources and generate a near optimal schedule for the task before it starts execution. However, it changes the initial schedule dynamically during the execution. For example, GrADS [32]

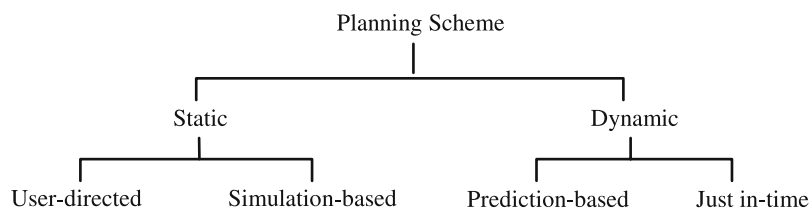


Figure 14. Planning scheme taxonomy.

generates preliminary mapping by using prediction results, but it migrates a task execution to another resource when its initial contract is broken or a better resource is found for execution. Sakellariou et al. [110] developed a low-cost rescheduling policy for the mapping of workflows on Grids. It considers rescheduling workflow tasks at a few carefully selected points during execution in a dynamically changing Grid environment, since the initial schedule built using inaccurate predictions can affect performance significantly.

Rather than making a schedule ahead, just in-time scheduling [42] only makes scheduling decision at the time of task execution. Planning ahead in Grid environments may produce a poor schedule, since it is a dynamic environment where utilization and availability of resources varies over time and a better resource can join at any time. Moreover, it is not easy to accurately predict the execution time of all application components on Grid resources. However, as the technology of advance reservation [119] for various resources improves, it is believed that the role of static and prediction-based planning will increase [40].

#### 2.3.4. Scheduling Strategy

In general, scheduling workflow applications in a distributed system is an NP-complete problem [50]. Therefore, many heuristics have been developed to obtain near-optimal solutions to match users' QoS constraints. As shown in Figure 15 we categorize strategies of major scheduling approaches into *performance-driven*, *market-driven* and *trust-driven*.

Performance-driven strategies try to find a mapping of workflow tasks onto resources that achieves optimal execution performance such as minimize overall execution time [96]. Most of Grid workflow scheduling systems falls in this category. GrADS [32] optimizes DAG-based workflows using Min–Min, Max–Min and Suffrage heuristics, hoping to obtain minimum completion times. Prodan et al. [106] use classical genetic algorithms with cycle elimination techniques to minimize non-DAG based workflow execution on Grids.

Market-driven strategies employ market models to manage resource allocation for processing workflow tasks. They apply computational economy principle and establish an open electronic marketplace between workflow management systems and participating resource providers. Workflow schedulers act as consumers buying services from the resource providers and pay some notion of electronic currency for executing tasks in the workflow. The tasks in the workflow are dynamically scheduled at run-time depending on resource cost, quality and availability, to achieve the desired level of quality for deadline and budget. Unlike the performance-driven strategy, market-driven schedulers may choose a resource with later deadline if its usage price is cheaper. Market-driven strategies have been applied to several Grid systems such as Nimrod-G [21] and Gridbus data resource broker [130]. One example of the market-driven workflow scheduling proposed by Geppert et al. [58] utilizes market mechanisms during the task assignment. In the system, bids are collected from eligible resource providers for each task. The optimal bid is selected by computing the amount of time and cost saved or overdrawn up to the point. If the execution time has been minimized at the expense of an overdrawn cost, a bid with lower price will be chosen as the optimal bid. Consequently, scheduler assigns the task to the resource whose provider offers the optimal bid. A recent work on cost-based scheduling of workflow tasks on Grids is reported in [19].

Recently, trust-driven scheduling approaches (e.g., CCOF project in [147] and GridSec project in [114, 115]) in distributed systems are emerging. Trust-driven schedulers select resources based on their trust levels. For example, within GridSec, the scheduler accesses the trust level of Grid sites. It maps tasks onto resources whose trust level is higher than users' demand. Trust model of resources is based on attributes such as security policy, accumulated reputation, self-defense capability, attack history, and site vulnerability. By using trust-driven approaches, workflow management systems can reduce the chance of selecting malicious hosts, and non-

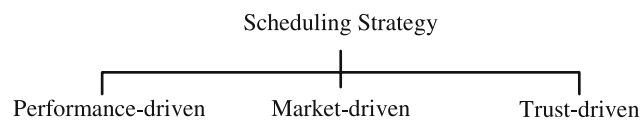


Figure 15. Scheduling strategy taxonomy.

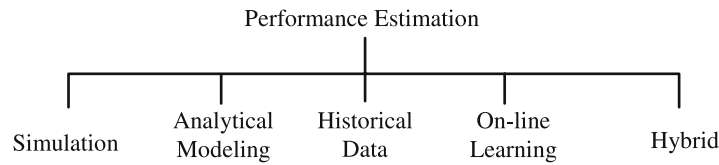


Figure 16. Performance estimation taxonomy.

reputable resources [147]. Therefore, overall accuracy and reliability of workflow execution will be increased.

### 2.3.5. Performance Estimation

In order to produce a good schedule, estimating the performance of tasks on resources is crucial, especially for constructing a preliminary workflow schedule. By using performance estimation techniques, it is possible for workflow schedulers to predict how tasks in a workflow or sub-workflow will behave on distributed heterogeneous resources and thus make decisions on how and where to run them. As indicated in Figure 16, there are several performance estimation approaches: *simulation*, *analytical modeling*, *historical data*, *on-line learning*, and *hybrid*.

Simulation approaches [43, 148] provide resource simulation environments to emulate the execution of tasks in the workflow prior to its actual execution. In analytical modeling [32, 37, 98], a scheduler predicts the performance of tasks in workflow on a given set of resources based on an analytic metric. For example, in GrADS [32], two types of performance models are developed, namely memory hierarchy performance model and computational model. By using these models, one can predict memory requirements and the execution time of an application component for a resource according to the associated problem size. The historical data approach [68, 91, 113] relies on historical data to predict the task's execution performance. The historical data related to a particular user's application performance or experience can also be used in predicting the share of available of resources for that user while making scheduling decisions based on QoS constraints. The on-line learning approach predicts task execution performance from on-line experience without prior knowledge of the environment's dynamics. For example, Buyya et al. [22] and Galstyan et al. [57] map a job onto a 'best' Grid resource by learning the completion time of most recent jobs submitted to resources. As historical and on-line learning ap-

proaches use experimental data, they can be broadly termed as *empirical modeling* approaches for performance estimation.

In certain conditions, these approaches could be used together in a hybrid approach for generating performance evaluation of workflow tasks. For instance, Bacigalupo et al. [16] use both layered queuing modeling and historical performance data to predict the performance of dynamic e-Commerce systems on heterogeneous servers. In addition, GrADS constructs computational models semi-automatically by emulating the execution of workflow components on small data sets. That is, it uses a combination of historical and analytical approaches for performance estimation.

## 2.4. Fault Tolerance

In a Grid environment, workflow execution failure can occur for various reasons: the variation in the execution environment configuration, non-availability of required services or software components, overloaded resource conditions, system running out of memory, and faults in computational and network fabric components. Grid workflow management systems should be able to identify and handle failures and support reliable execution in the presence of concurrency and failures.

As shown in Figure 17, Hwang et al. [66] divided workflow failure handling techniques into two different levels, namely *task-level* and *workflow-level*. *Task-level* techniques mask the effects of the execution failure of tasks in the workflow, while *workflow-level* techniques manipulate the workflow structure such as execution flow to deal with erroneous conditions.

Task-level techniques have been widely studied in parallel and distributed systems. They can be cataloged into *retry*, *alternate resource*, *checkpoint/restart* and *replication*. The *retry* technique [121] is the simplest failure recovery technique, as it simply

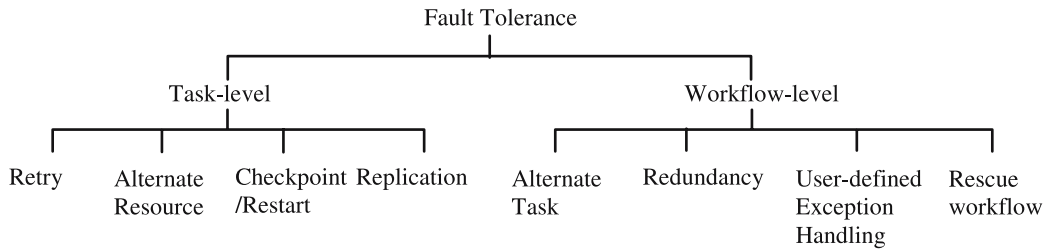


Figure 17. Fault tolerance taxonomy.

tries to execute the same task on the same resource after failure. The alternate resource technique [121] submits failed task to another resource. The checkpoint/restart technique [36] moves failed tasks transparently to other resources, so that the task can continue its execution from the point of failure. The replication technique [7, 66] runs the same task simultaneously on different Grid resources to ensure task execution provided that at least one of the replicas does not fail.

Workflow-level techniques include *alternate task*, *redundancy*, *user-defined exception handling* and *rescue workflow*. The first three approaches proposed in [66] assume there is more than one implementation for a certain computation with different execution characteristics. The alternate task technique executes another implementation of a certain task if the previous one failed, while the redundancy technique executes multiple alternative tasks simultaneously. The user-defined exception handling allows the users to specify a special treatment for a certain failure of a task in workflow. The rescue workflow technique developed in Condor DAGMan system [36] ignores the failed tasks and continues to execute the remainder of the workflow until no more forward progress can be made. Then, a rescue workflow description called rescue DAG, which indicates failed nodes with statistical information, is generated for later submission.

## 2.5. Intermediate Data Movement

For Grid workflow applications, the input files of tasks need to be staged to a remote site before processing the task. Similarly, output files may be required by their children tasks which are processed on other resources. Therefore, the intermediate data has to be staged out to the corresponding Grid sites. Some systems require users to manage intermediate data transfer in the workflow specification, rather than providing *automatic* mechanisms to transfer intermediate data. As indicated in Figure 18, we categorize approaches of *automatic* intermediate data movement into *centralized*, *mediated* and *peer-to-peer*.

Basically a centralized approach transfers intermediate data between resources via a central point. For example, a central workflow execution engine can collect the execution results after task completion and transfer them to the processing entities of corresponding successors. Centralized approaches are easy to implement and suit workflow applications in which large-scale data flow is not required.

In a mediated approach, rather than using a central point, the locations of the intermediate data are managed by a distributed data management system. For example, in Pegasus system, the intermediate data generated at every step is registered in a replication catalog service [30], so that input files of

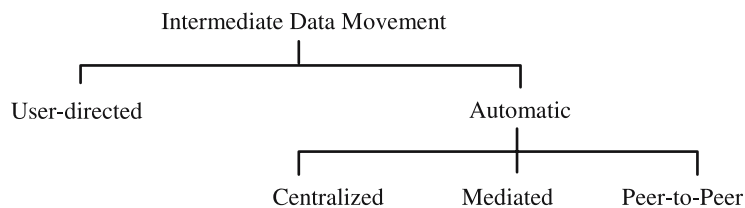


Figure 18. Intermediate data movement.

every task can be obtained by querying the replication catalog service. Mediated approaches are more scalable and suitable for applications which need to keep intermediate data for later use.

A peer-to-peer approach transfers data between processing resources. Since data is transmitted from the source resource to the destination resource directly without involving any third-party service, peer-to-peer approaches save the transmission time and reduce the bottleneck problem caused by the centralized and mediated approaches. Thus, they are suitable for large-scale intermediate data transfer. However, there are more difficulties in deployment because they require every Grid node to be capable of providing both data management and movement service. In contrast, centralized and mediated approaches are more suitable to be used in applications such as bio-applications, in which users need to monitor and browse intermediate results. In addition, they also need to record them for future verification purposes.

### 3. Grid Workflow Management System Survey

In this section, we present a detailed survey of existing Grid workflow systems in addition to mapping the proposed taxonomy. Table 1 shows the summary of selected Grid workflow management projects. A comparison of various Grid workflow systems and their categorization based on the taxonomy is shown in Tables 2–4.

#### 3.1. Condor DAGMan

Condor [81, 120, 124] is a specialized resource management system (RMS) developed at the University of Wisconsin-Madison for compute-intensive jobs. Condor provides a High Throughput Computing (HTC) environment based on large collections of distributed computing resources ranging from desktop workstations to super computers. Condor-G, a component within Condor, utilizes Globus GRAM serving as a uniform interface to heterogeneous batch systems, thus enabling large scale computational Grids. *Matchmaking* within Condor, matches jobs and available resources according to their job and resource classified advertisement. When more than one resource satisfies the job requirement, the re-

source with higher value of rank expression, which expresses the desirability of a match, is preferred.

The Directed Acyclic Graph Manager (DAGMan) [36, 120] is a meta-scheduler for Condor jobs. While Condor aims to discover available machines for the execution of jobs, DAGMan handles the dependencies between the jobs. DAGMan uses DAG as the data structure to represent job dependencies. Each job is a node in the graph and the edges identify their dependencies. Each node can have any number of “parent” or “children” nodes. Children cannot run until their parents have completed. Cycles, where two jobs are both descended from one another, are prohibited, because it would lead to deadlock. DAGMan does not support automatic intermediate data movement, so users have to specify data movement transfer through pre-processing and post-processing commands associated with processing job.

The individual job execution is managed by Condor scheduler. So if a job fails due to the nature of the distributed system, such as loss of network connection, it will be recovered by Condor while DAGMan is unaware of such failures. However, DAGMan is responsible for reporting errors for the set of submitted jobs, and generates a rescue DAG. In the case of a job failure, the remainder of the DAG continues until no more progress can be made. A failed node can be retried a configurable number of times. The rescue DAG indicates the uncompleted portions of the DAG with detail of failures. Users can correct the errors of failed jobs and resubmit the rescue DAG.

#### 3.2. Pegasus in GriPhyN

GriPhyN [61] aims to support large-scale data management in physics experiments such as high-energy physics, astronomy, and gravitational wave physics. Pegasus [40, 41, 42] (Planning for Execution in Grids) is a workflow manager in GriPhyN developed by the University of Southern California.

Pegasus performs a mapping from an abstract workflow to the set of available Grid resources, and generates an executable workflow. An abstract workflow can be constructed by querying Chimera [53], a virtual data system, or provided by users in DAX (DAG XML description). An abstract workflow describes the computation in terms of logical files and logical application components and indicates

Table 1. Summary of Grid workflow management projects.

| Name                   | Organization   | Prerequisite  | Grid integration  | Applications   | Availability                             |
|------------------------|--|---|---|--|--|
| DAGMan [120]           | University of Wisconsin-Madison, USA.<br><a href="http://www.cs.wisc.edu/condor/dagman/">http://www.cs.wisc.edu/condor/dagman/</a>   | Condor  | Condor which can run on top of Globus Toolkit version 2 (GT2) | Compute-intensive  | GPL (General Public License)             |
| Pegasus [41]           | University of Southern California, USA.<br><a href="http://pegasus.isi.edu">http://pegasus.isi.edu</a>   | Condor DAGMan, Globus RLS   | Condor and Globus   | Targeted for data-intensive, but supports other types                          | GTPL (Globus Toolkit Public License)     |
| Triana [123]           | Cardiff University, UK.<br><a href="http://www triana code.org/">http://www triana code.org/</a>   | Grid Application Toolkit (GAT)  | GAT (IXTA, Web services, Globus)                              | Compute-intensive  | The Apache Software License              |
| ICENI [93]             | London e-Science Centre, UK.<br><a href="http://www.lesc.ic.ac.uk/iceni/">http://www.lesc.ic.ac.uk/iceni/</a>  | Globus Toolkit  | Jini, IXTA, Globus  | Compute-intensive  | ICENI Open Source Code Licence           |
| Taverna [100]          | European Institutes and industries. Collaboration between several<br><a href="http://taverna.sourceforge.net/">http://taverna.sourceforge.net/</a>                             | Java 1.4+   | Web services, Soaplab, local processor, BioMoby, etc.         | Service Grids  | GNU Lesser General Public License (LGPL) |
| GridAnt [75]           | Argonne National Laboratory, USA.<br><a href="http://www.cogkit.org/">http://www.cogkit.org/</a>   | Apache Ant, Globus Toolkit  | GT2, GT3, GT4   | Client controllable workflow applications                                      | GTPL                                     |
| GrADS [18]             | Collaboration between several American Universities. <a href="http://www.hipersoft.rice.edu/grads/">http://www.hipersoft.rice.edu/grads/</a>                                   | Globus Toolkit, Autopilot, NWS  | Globus, Parallel Systems (e.g., MPI)                          | Compute-intensive and communication-intensive applications with MPI components | Not yet available in public              |
| GridFlow [25]          | University of Warwick, UK<br><a href="http://www.dcs.warwick.ac.uk/research/hpsg/workflow/workflow.html">http://www.dcs.warwick.ac.uk/research/hpsg/workflow/workflow.html</a> | Agent-based Resource Management System, Performance Analysis and Characterize Environment (PACE) Toolkit, Titan | Parallel Systems (e.g., MPI and PVM)                          | MPI and PVM based components   | Not yet available in public              |
| Unicore [11]           | Collaboration between German research institutions and industries.<br><a href="http://www.unicore.org/">http://www.unicore.org/</a>  | Unicore middleware  | Unicore   | Computational-intensive and MPI components                                     | Community Source License                 |
| Gridbus workflow [144] | The University of Melbourne, Australia.<br><a href="http://www.gridbus.org">http://www.gridbus.org</a>   | Globus Toolkit  | GT2   | Computational- and Data-intensive  | GPL                                      |
| Askalon [49]           | University of Innsbruck<br><a href="http://dps.uibk.ac.at/askalon">http://dps.uibk.ac.at/askalon</a>   | Globus Toolkit  | GT2, GT4, WSRF, Web services                                  | Performance-oriented applications  | GTPL                                     |
| Karajan [76]           | Argonne National Laboratory<br><a href="http://www.cogkit.org">http://www.cogkit.org</a>   | Java 1.4  | GT2, GT3, GT4, Condor, runtime exec, ssh, WebDAV              | Those required to access Grid middleware                                       | GTPL                                     |
| Kepler [12]            | A cross-project collaboration.<br><a href="http://kepler-project.org/">http://kepler-project.org/</a>  | Java  | Globus, Storage Resource Broker (SRB), EcoGrid, Web services  | Scientific workflow applications   | UC Berkeley License                      |

Table 2. Workflow design taxonomy mapping.

| Project name     | Structure | Model             | Composition systems                                | QoS constraints   |
|------------------|-----------|-------------------|--|---|
| DAGMan           | DAG       | Abstract          | User-directed<br>• Language-based                  | User specified rank<br>expression for desired resources         |
| Pegasus          | DAG       | Abstract          | User-directed<br>• Language-based<br>Automatic     | N/A   |
| Triana           | Non-DAG   | Abstract          | User-directed<br>• Graph-based                     | N/A   |
| ICENI            | Non-DAG   | Abstract          | User-directed<br>• Language-based<br>• Graph-based | Metrics specified by users                                      |
| Taverna          | DAG       | Abstract/concrete | User-directed<br>• Language-based<br>• Graph-based | N/A   |
| GridAnt          | Non-DAG   | Concrete          | User-directed<br>• Language-based                  | N/A   |
| GrADS            | DAG       | Abstract          | User-directed<br>• Language-based                  | Estimated application execution time                            |
| GridFlow         | DAG       | Abstract          | User-directed<br>• Graph-based<br>• Language-based | Application execution time                                      |
| Unicore          | Non-DAG   | Concrete          | User-directed<br>• Graph-based                     | N/A   |
| Gridbus workflow | DAG       | Abstract/concrete | User-directed<br>• Language-based                  | Deadline, cost minimisation                                     |
| Askalon          | Non-DAG   | Abstract          | User-directed<br>• Graph-based<br>• Language-based | Constraints and properties specified<br>by users or pre-defined |
| Karajan          | Non-DAG   | Abstract          | User-directed<br>• Language-based<br>• Graph-based | N/A   |
| Kepler           | Non-DAG   | Abstract/concrete | User-directed<br>• Graph-based                     | N/A   |

their dependencies in the form of Directed Acyclic Graph (DAG). Before mapping, Pegasus reduces the abstract workflow by reusing a materialized dataset which is produced by other users within a VO. Reduction optimization assumes that it is more costly to produce a dataset than access the processing results. The reduction algorithm removes any antecedents of the redundant jobs that do not have any unmaterialized descendents in order to reduce the complexity of the executable workflow.

Pegasus consults various Grid information services to find the resources, software, and data that are used in the workflow. A Replica Location Service (RLS) [30] and Transformation Catalog (TC) [39] are used to locate the replicas of the required data, and to find the location of the logical application components respectively. Pegasus also queries Globus

Monitoring and Discovery Service (MDS) [34] to find available resources and their characteristics.

There are two methods used in Pegasus for resource selection, one is through random allocation, the other is through a performance prediction approach. In the latter approach, Pegasus interacts with Prophecy [68, 140], which serves as an infrastructure for performance analysis and modeling of parallel and distributed applications. Prophecy is used to predict the best site to execute an application component by using performance historical data. Prophecy gathers and stores the performance data of every application. The performance information can provide insight into the performance relationship between the application and hardware and between the application, compilers, and run-time systems. An analytical model is produced based on the performance data

Table 3. Workflow scheduling taxonomy mapping.

| Project name     | Architecture  | Decision making | Planning scheme               | Strategies                  | Performance estimation                           |
|------------------|---------------|-----------------|-------------------------------|-----------------------------|--|
| DAGMan           | Centralized   | Local           | Just in-time                  | Performance-driven          | N/A  |
| Pegasus          | Centralized   | Local/global    | User-directed/just in-time    | Performance-driven          | Historical data, analytical modeling             |
| Triana           | Decentralized | Local           | Just in-time                  | Performance-driven          | N/A  |
| ICENI            | Centralized   | Global          | Prediction-based              | Performance & market-driven | Historical data                                  |
| Taverna          | Centralized   | Local           | Just in-time                  | Performance-driven          | N/A  |
| GridAnt          | Centralized   | User-defined*   | User-directed                 | User-defined*               | N/A  |
| GrADS            | Centralized   | Local/global    | Prediction-based              | Performance-driven          | Historical data (empirical), analytical modeling |
| GridFlow         | Hierarchical  | Local           | Simulation-based              | Performance-driven          | Analytical modeling                              |
| Unicore          | Centralized   | User-defined*   | User-directed                 | User-defined*               | N/A  |
| Gridbus Workflow | Hierarchical  | Local           | User-directed/just in-time    | Market-driven               | Historical data (empirical)                      |
| Askalon          | Decentralized | Global          | Just in-time/prediction-based | Performance & market-driven | Analytical modeling, historical data             |
| Karajan          | Centralized   | User-defined*   | User-defined*                 | User-defined*               | N/A  |
| Kepler           | Centralized   | User-defined*   | User-defined*                 | User-defined*               | N/A  |

\***User-defined**—the architecture of the system has been explicitly designed for user extension.

and is used by the prediction engine to predict the performance of the application on different platforms. It is required that Pegasus send the request associated with information such as the component name, the semantic parameter names and their values, and the list of available resources. The ranking of the given resources is returned by Prophesy after the query is received.

For ease of use, Pegasus is able to generate a workflow from a metadata description of the desired data product with the aid of artificial intelligence planning techniques. Although, the workflow execution of Pegasus is based on static planning and its executable workflow is transformed into Condor jobs for execution management by Condor DAGMan, it has been recently extended to support just in-time scheduling [42] and pluggable task scheduling strategies.

### 3.3. Triana

Triana [122, 123] is a visual workflow-oriented data analysis environment developed at Cardiff University. In 2002, Triana was extended to implement a consumer Grid [122] by using a peer-to-peer approach. Recently, Triana has been redesigned and integrated with Grids via GridLab GAT (Grid Application Toolkit) interface [10]. GAT defines a high

level API for core Grid service access through JXTA [70], Web services [133], and OGSA [54, 126].

Triana provides a visual programming interface with functionality represented by units. Applications are written by dragging the required units onto the workplace and connecting them to construct a workflow. Apart from many implemented tool units, Triana also provides a custom user interface to allow users to build their own units. Several control units (e.g., loop) and logic units (e.g., if) are also provided for users to control the logic of workflow execution. Since control and logic units are implemented as a standard Triana unit, it is easy to introduce new flow patterns. Interconnected units can also be grouped into a group unit, which has the same properties as normal unit.

Triana clients such as Triana GUI can log into a Triana Controlling Service (TCS), remotely build and run a workflow and then visualize the result on their device (e.g., PC, PDA, etc.). Each TCS interacts with the Triana engine and every engine provides a service and is capable of executing complete or partial task-graphs locally, or by distributing the code to other servers based on the specified distribution policy for the supplied task-graph. The distribution policy is based on the concept of group units and two distribution policies have been implemented, namely parallel and peer-to-peer. Both policies distribute

Table 4. Information retrieval, fault-tolerance and data movement taxonomy mapping.

| Project name                | Information retrieval  | Fault-tolerance  | Data movement                        |
|-----------------------------|--|--|--------------------------------------|
| DAGMan                      | Resource information is retrieved by Condor <i>Matchmaker</i> that manages resource and task info advertisement and notification.  | Task level<br>• Migration<br>• Retrying<br>Workflow Level<br>• Rescue workflow                                   | User-directed                        |
| Pegasus                     | Resource information retrieved through Globus MDS and RLS. Application component information is retrieved from the GriPhyN Transformation Catalog.   | Based on DAGMan  | Mediated                             |
| Triana<br>ICENI             | Based on GAT protocol Application component information is retrieved by the component metadata service and performance repository service.   | Based on GAT manger<br>Based on middleware   | Peer-to-peer<br>Mediated             |
| Taverna                     | Service information is retrieved through DAML-S web service ontology, domain ontology information service, and UDDI.   | Task level<br>• Retry<br>• Alternate resource  | Centralized                          |
| GridAnt                     | Resource information is retrieved through Globus MDS.  | User-defined*  | User-directed                        |
| GrADS                       | Resource information is retrieved through Globus MDS and GrADS information service (GIS). Dynamic information is retrieved by NWS. Autopilot is used for provide performance contract information. | Task level in rescheduling work in GrADS, but not in workflows.  | Peer-to-peer                         |
| GridFlow                    | Resource information is retrieved through Titan  | Task level<br>• Alternate resource   | Peer-to-peer                         |
| Unicore<br>Gridbus workflow | Unicore information service<br>Resource information is retrieved through the Grid Market Directory   | Based on Unicore middleware<br>Task level<br>• Alternate resource  | Mediated<br>Centralized              |
| Askalon                     | Static information<br>• Infrastructure-related<br>• Configuration-related<br>• QoS-related<br>Dynamic information<br>• Resource-related<br>• Execution-related                                     | Task level<br>• Retry<br>• Alternate resource<br>Workflow level<br>• Rescue workflow                             | Centralized User-directed            |
| Karajan                     | User-defined*  | Task level<br>• Retry<br>• Alternate resource<br>Workflow level<br>• User-defined exception handling             | User-directed                        |
| Kepler                      | User-defined*  | Task level<br>• Alternative resource<br>Workflow level<br>• User-defined exception handling<br>• Workflow rescue | Centralized Mediated<br>Peer-to-Peer |

\***User-defined**—the architecture of the system has been explicitly designed for user extension.

every unit in the group to separated hosts, however while the peer-to-peer mechanism relies on intermediate data being passed between hosts, there is no such host-based communication with the parallel policy. Since a distributed task-graph is not fixed to a specific set of resources, it can be dynamically allocated to available services in the most effective way.

### 3.4. *Workflow Management in ICENI*

The ICENI (Imperial College e-Science Network Infrastructure) [93, 94] developed at London e-Science Centre provides component-based Grid middleware. Within ICENI, users construct an abstract workflow, which is a collection of components, and then submit this to ICENI environment for execution.

Each ICENI component is described in terms of meaning, control flow and implementation. The workflow components are primarily composed based on a spatial view, in which all units are represented concurrently, with details of how they relate and interact with each other. Then a temporal view is derived from the spatial view by the system. In the temporal view, workflow information is attached to each component that consists of a graph in which the directed arcs contain the partnership according to the temporal dependence. Within ICENI, the workflow model is similar to that of the YAWL (Yet Another Workflow Language) [4], although simplified in certain respects. The workflow language includes all basic workflow structure such as sequence, parallelism, choice and iteration.

The scheduling service [93, 142, 143] within ICENI is responsible for concretizing the abstract workflow. The scheduling task includes matching component meaning with component implementation and mapping these qualified components onto a suitable subset of the available resources. Several scheduling algorithms used to determine resource mapping have been implemented. They include random, best of  $n$  random, simulated annealing and game theory. Most schedulers implemented within ICENI aim to provide approximate optimal solutions to map the abstract workflow to a combination of component implementations and resources in terms of execution time and cost. The schedulers take into account all components in applications rather than standalone components. The scheduling framework also allows third-party scheduling algorithms to be plugged in.

ICENI has developed a performance repository system [91] which is able to monitor running applications and obtain and store performance data for the components within the applications. This data is stored within a repository with meta-data about the resource the component was executed on, the implementation of the component used, and the number of other components concurrently running on the same resource. This data can be used by schedulers for future runs of applications to estimate the execution times of each component within the workflow.

Two scheduling schemes [93] are considered within ICENI, namely lazy scheduling and advanced reservation. The metadata of the component implementation indicates which scheme the component can benefit from. Non-reservation component is scheduled to a resource just before it is required, while reservation component has been allocated to a resource and has made a reservation in advance. The schedulers can interrogate the performance repository to predict execution in order to produce accurate reservation. The reservation negotiation protocol is based on WS-Agreement [60].

### 3.5. *Taverna in <sup>my</sup>Grid*

Taverna [100] is the workflow management system of the <sup>my</sup>Grid [118] project, which aims to exploit Grid technology to develop high-level middleware for supporting personalized in silico experiments in biology. Taverna is a collaboration between several European universities, institutes and industries. The purpose of Taverna is used to assist scientists with the development and execution of bioinformatics workflows on the Grid. Taverna provides data models, enactor task extensions, and graphical user interfaces. FreeFluo [55] is also integrated into Taverna as a workflow enactment engine to transfer intermediate data and invoke services.

In Taverna, data models can be represented in either a graphical format or in an XML based language called Simple Conceptual Unified Flow Language (SCUFL). The data model consists of inputs, outputs, processors, data flow and control flow. In addition to specifying execution order, the control flow can also be triggered by state transitions during the execution of parent processors. Compared to other workflow languages, such as the Business Process Execution Language for Web Services (BPEL4WS)

[14], SCUFL allows implicit iteration over incoming data sets based on a specified strategy. At the execution level, the workflow enactor also provides a multithreading mechanism to speed up the iteration process. Users are allowed to set the *Thread* property to specify how many concurrent instances will send parallel requests to the iteration processor. It is especially suitable for services that are capable of handling significant simultaneous processing, for example, a service that is backed by a cluster. It also can reduce service waiting time since workflow engine can send the next input data at the same time as the service is working on the current input.

Taverna also provides a user-friendly multi-window environment for users to manipulate workflows, validate and select available resources, and then execute and monitor these workflows. The enactment status panel [121] of Taverna shows the current progress of a workflow invocation. It also allows the users to browse the intermediate and final results. Through the enactment panel, users can handle storage of those results on local or remote data stores in a variety of formats.

Fault tolerance [121] in the workflow management of <sup>my</sup>Grid is achieved by setting configuration for each processor in the workflow, for example, the number of retries, time delay and alternate processors. It also allows users to specify the critical level for faults on each processor. If the processor is set as *Critical*, after all retries and alternates have failed, entire workflow execution will be terminated, otherwise, the workflow will continue but children nodes of the failed processor will never be invoked.

<sup>my</sup>Grid follows service-oriented Grid architecture and supports several different types of services within the workflow management system, including WSDL-based [138] single operation web services, soaplab bio-services [111] and local services such as programs coded as java classes. In addition, information services such as UDDI (the Universal Description, Discovery and Integration) [127] and ontology directory [139] are adopted for service discovery.

### 3.6. GridAnt

The GridAnt [13, 75] is an extensible client-side workflow management system developed by Argonne National Laboratory. It has been designed for Grid end-users as a convenient tool to express

and control the execution sequence without having any expertise in sophisticated workflow systems. GridAnt focuses on distributed process management rather than the aggregation of services which is the concern of most other Grid-enabled workflow frameworks.

GridAnt consists of four major components, namely workflow engine, run-time environment, workflow vocabulary and workflow monitoring. The workflow engine is the central controller that handles task dependencies, failure recoveries, performance analysis, and process synchronization. GridAnt workflow engine extends Ant [15], an existing commodity tool for controlling build process in Java, by adding additional components to support workflow orchestration and composition. GridAnt also provides an environment for inter-task communication, so that individual GridAnt tasks can read and write intermediate data by using a globally accessible whiteboard-style communication model. Several important constructs such as constants, arithmetic expressions, global variables, array references, and literals are supported by the run-time environment. GridAnt extends Ant's vocabulary in the Grid domain with the addition of the tags such as *grid-copy*, *Grid-authenticate* and *Grid-query*. These new tags are used by users to pre-define the Grid tasks and construct complex workflows at compile time. It uses a control construct provided by Ant container for expressing parallel and sequential tasks. Furthermore, users are allowed to monitor the progress of the execution by means of graphical visualization tool.

In addition to mapping complex client-side workflows, GridAnt can be used for testing the functionality of different Grid services. It has been developed to support version 2 and version 3 of the Globus toolkit [59] by using the Java CoG kit [74]. It has been applied for Position-Resolved Diffraction [13], which is a new experimental technique for the study of nanoscale structures as part of the Argonne National Laboratory's advanced analytical electron microscope.

### 3.7. Workflow Management in GrADS

The Grid Application Development Software (GrADS) project [18] aims to provide programming tools and execution environments for ordinary scientific users to develop, execute, and tune applications on the Grid. GrADS is a collaboration between

several American Universities. GrADS supports application development either by assembling domain-specific components from a high-level toolkit or by creating a module by relatively low-level (e.g., MPI [95]) code [32].

GrADS provides application-level scheduling to map workflow application tasks to a set of resources. New Grid scheduling and rescheduling methods [32, 38] are introduced in GrADS. These scheduling methods are guided by an objective function to minimize the overall job completion time (*make-span*) of the workflow application. The scheduler obtains resource information by using services such as MDS [109] and NWS [136] and locates necessary software on the scheduled node by query GrADS Information Service (GIS). The workflow scheduler ranks each qualified resource for each application component. A rank value is calculated by using “a weighted sum of the expected execution time on the resource and the expected cost of data movement for the component.” After ranking, a performance matrix is constructed and used by the scheduling heuristics to obtain a mapping of components onto resources. Three heuristics have been applied in GrADS; those are Min–Min, Max–Min, and Suffrage heuristics [87].

GrADS has built up an architecture-independent model of the workflow component from individual component models. It employs analytical models that are constructed semi-automatically from empirical models (historical data/sample execution data), in order to estimate the performance of a workflow component on a single Grid node. It uses hardware performance counters to collect operation counts from several executions of the workflow components with different, small-size input problems, and then it performs a least-squares fit to the data to construct computational models. In addition, GrADS reuses distance data on small inputs to predict the fraction of cache hits and misses on the given data and cache configuration by its memory–hierarchy performance models.

GrADS utilizes Autopilot [107] to monitor performance of the agreement between the application demands and resource capabilities. Once the contract is violated, the rescheduler [32] of the GrADS takes corrective actions. It has been implemented using two rescheduling approaches for MPI applications, the stop/restart approach and process swapping. In the former approach, an executing application component

is suspended and migrated to a new resource if better resources are found for improving the execution performance [129]. As a migration event can involve large data transfers, expensive startup costs and significant application code modifications, process swapping provides a lightweight, but less flexible, alternative approach. In process swapping more machines than will actually be used for the computation are launched for an MPI application component, and slower machines in the active set are swapped with faster machines in the inactive set periodically, according to the performance of machines.

### 3.8. GridFlow

GridFlow [25] is a Grid workflow management system developed at the University of Warwick. This work is built on the top of an agent-based resource management system for Grid computing (ARMS) [24]. Rather than focusing on workflow specification and the communication protocol, GridFlow is more concerned about service-level scheduling and workflow management.

There are three layers of Grid resource management within the GridFlow system: the Grid resource, the local Grid and the global Grid. A Grid resource is simply just a particular Grid resource; local Grid consists of multiple Grid resources that belong to one organization; and a global Grid consists of all local Grids. Global Grid also provides a portal for compose the workflow.

A workflow in GridFlow is represented as a flow of several different activities, each activity represented by a sub-workflow. Each sub-workflow is a flow of closely related tasks that is to be executed in a local Grid. A portal has been developed by GridFlow as graphical user interface for users to compose workflow elements.

The workflow management within GridFlow is conducted by a hierarchical scheduling system including global Grid workflow manager and local Grid sub-workflow scheduling. Global Grid workflow manager receives requests from the GridFlow portal with the workflow description in the format of XML, and then simulates workflow execution to find a near-optimal schedule. After the users accept the simulated result, GridFlow schedules the workflow onto different local Grids through ARMS. Within ARMS, each agent represents a local Grid at a global

level of Grid resource management, and conducts local Grid sub-workflow scheduling. In contrast to the global Grid workflow management, the local Grid schedulers handle conflicts since scheduled sub-workflows may belong to different workflows.

ARMS has integrated Titan [116], which utilizes performance data obtained from PACE [98], a toolset for resource performance and usage analysis, with iterative heuristic algorithms to minimize the make-span and idle time of a Grid resource. PACE can exact control flow, and use an analytical model approach based on queuing theory, to predict application performance on a given set of resources such as time, scalability and system resource usage. Titan also provides Grid resource information.

### 3.9. Workflow Management in Unicore Plus

Unicore plus [128] provides seamless and secure access to distributed resources of the German high performance computing centers. Unicore plus is a follow-on project of Unicore (Uniform Interface to Computing Resources) [11], started in 1997 to improve uniform interfaces to distributed High Performance Computing and data resources using the mechanisms of the World Wide Web. Unicore plus provides a programming environment for users to design and execute job flow.

Within Unicore, one job or job group that can be executed on any Unicore site may contain other jobs and/or job groups. The original Unicore job model supports jobs that are constructed as a set of directed acyclic graphs with temporal dependencies. Since Unicore version 4, advanced flow controls have been added, which include conditional execution (e.g., if-then-else), repeated execution (e.g. do-*n*), conditional repeated execution (e.g., do-repeat), and conditional suspend action (e.g., hold-job). In addition, three types of run-time conditions are implemented for supporting conditional checking; these are based on the return code of a previous executed task, existence or properties of a file and whether a given time and date have passed.

Unicore plus provides graphical tools that allow users to create a job flow and convert it into an Abstract Job Object (AJO) which is a serialized java object. The AJO is submitted from a user client to a Unicore server. The server translates the job specification into a number of batch jobs and dispatches them

to the target resource. The server also makes sure that a successor is executed if its predecessors are finished and all necessary data is available at the executing site.

Unicore allows users to specify jobs and different parts of job group onto multiple resources. The output of individual jobs may be needed by its successors. Therefore, a temporary Unicore space is created for each job group for transferring data sets. Unicore also allows users to explicitly specify the transfer function as a task through GUI; it is also able to perform the necessary data movement function without user intervention.

### 3.10. Workflow Management in Gridbus

The Gridbus Toolkit [23] developed by the University of Melbourne provides Grid technologies for service-oriented utility computing. Its architecture is driven by the requirements of Grid economy [22]. A Grid economy mechanism has been proposed as a technique for efficient management of distributed resources. It helps in efficient allocation of resources to different users and applications based on their QoS requirements in addition to regulation of the supply and demand for Grid resources.

The workflow management in Gridbus [144] provides a simple XML-based workflow language for users to define their tasks and dependencies. The workflow description language of Gridbus is aimed towards enabling the expression of parameter sweep tasks [8] and users' QoS requirements [146].

The workflow engine of Gridbus provides a hierarchical scheduling architecture to adapt to heterogeneous and dynamic Grid environments. Within the workflow execution engine, the schedules of the workflow tasks are driven by the events by using the tuple-space model [56]. An event-driven mechanism with subscription-notification approach makes the workflow execution loosely coupled and flexible. The system also supports just in-time scheduling, allowing scheduling decision to be made at the time of task execution. The scheduler can also reschedule failed tasks to an alternative resource. In addition, Grid Market Directory (GMD) [145] is utilized by the workflow schedulers for run-time resource discovery.

In contrast to other workflow management systems, the Gridbus workflow system emphasizes on the use of market-based principles and algorithms for

resource allocation and scheduling applications in global Grid environments. It has been targeted to support applications in both scientific and business domains such as natural language processing and molecular modeling for drug discovery.

### 3.11. *Askalon*

Askalon [49] is a Grid application development and computing environment developed by the University of Innsbruck, Austria. The main objective of Askalon is to simplify the development and optimization of mostly Grid workflow applications that can harness the power of Grid computing.

Askalon comes with two separate composition systems, AGWL (Abstract Grid Workflow Language) [47] and Teuta [48], that support the development of Grid workflow applications. AGWL is an XML-based language. It provides a rich set of constructs to express sequence, parallelism, choice, and iteration workflow structure. In addition, programmers can specify high-level constraints and properties defined over functional and non-functional parameters for tasks and their dependencies which can be useful for a runtime system to optimize the workflow execution. Teuta supports the graphical specification of Grid workflow applications based on the UML activity diagram which is a graphical interface to AGWL.

Askalon provides a new hybrid approach for scheduling workflow applications on the Grid through dynamic monitoring and steering combined with a static optimization. Static scheduling maps entire workflows onto the Grid using genetic algorithms. A problem-independent objective function design allows to plug-in a variety of optimization metrics such as the execution time, efficiency, economical cost, or any user-defined QoS parameter. A dynamic scheduling algorithm takes into consideration the dynamic nature of the Grid resources such as machine crashes or external CPU and network load. Performance contracts are defined for every task and monitor whether tasks execute properly or whether they should be migrated. Askalon develops a fault tolerant execution engine that supports reliable workflow execution in the presence of resource failures through checkpointing and migration techniques.

In order to provide automatic workflow orchestration, Askalon Grid Resource Management (Grid-ARM) provides a distributed GT4-based registry to

map generic or domain specific tasks to their implementations. Askalon also includes automatic search for performance problems and faults in Grid infrastructures and applications. The monitoring and performance analysis component provides static information of Grid infrastructure and dynamic information of computational resources, networks, and applications. Dynamic information of workflow-based applications is provided for the entire workflow as well as for invoked applications called within tasks. The performance of workflow components is estimated based on a training phase which measures the actual execution time of tasks for different loads and problem sizes on a variety of Grid sites. The performance estimation of the workflow is conducted based on a combination of historical data obtained from a training phase and analytical modeling.

### 3.12. *Karajan*

Karajan [76, 77], developed by Argonne National Laboratory, aims to provide an integrated approach of exposing workflow to the Grid community. It is an extensible workflow framework and can be easily utilized by third parties to provide workflow solutions for a variety of users. It is derived from GridAnt and provides additional capabilities such as scalability, workflow structure and error handling.

Karajan is part of Java CoG Kit. Java CoG Kit is based on modular design and provides mechanisms for fast application development and easy integration of the variety of Grid middleware. It provides a number of programming abstractions for job executions and file transfers. The concept of *Grid providers* is introduced to facilitate different middleware to be used as part of an instantiation of Grid abstractions. As a result, it is easy to integrate Karajan to any middleware. To date, it has been integrated into various versions of Globus, Condor, runtime exec, ssh, and some data transfer techniques such as WebDAV [31] and scp. Karajan leverages lower-level programming abstractions in Java CoG Kit to access the Grid, and at the same time it provides programming interfaces for higher level applications such as workflow schedulers and application portlets to develop users' strategies.

In addition to sequence and parallelism, Karajan supports choices and loops of workflow structures. It also provides a user friendly XML-based workflow

language. Elements used for the description of workflow tasks are user-definable. Thus, the user can define names and parameters along with annotations and descriptions for a new element. A number of standard operators including mathematical and Boolean operators are defined for integration within execution control statements. It also provides advanced data structures such as list, range, and map (or hash tables) for repetitive tasks (e.g., parameter studies) as part of the workflow.

A number of fault handling methods are supported in Karajan. *Error handling* allows users to integrate strategies for errors and exceptions into the workflow. Checkpointing enables users to store intermediate states of the workflow execution for later roll back when a problem occurs.

### 3.13. Kepler

Kepler [12, 85] is one of the popular workflow systems with advanced features for composing scientific applications. It is derived from Ptolemy II system [82] and currently under development across a number of scientific data management projects. In addition to a user-friendly graphical user-interface and an extendable open source platform, Kepler also inherits the actor-oriented feature from Ptolemy II. It models a workflow system as a composition of independent components (actors) that communicate through well-defined interfaces. An actor is an encapsulation of parameterized operations performed on input to produce output data. An execution model of a workflow, which can be defined in a *director* object, imposes an execution order and communication mechanisms on the usable actors of the workflow. This modular design approach allows different execution models or machineries to be implemented and easily plugged into workflows without changing any of the components of workflows.

Kepler has been extended to support seamless access to remote resources and services. A web service HARVESTER component can retrieve all service description files in a web page or service repository to create instantiations of web services actors in the user's local actor library. Each web services actor can be instantiated for any particular operation specified in its service description. A number of fault-tolerant methods have been developed to make workflows with web services more

reliable. Instead of associating a service operation with a fixed URL, a list of services is allowed to provide the alternative invocation during service failure. It is also able to produce partial results even when the entire workflow fails. Advanced failure handling can also be supported through extensions of exception-catching actors. In addition, Kepler has defined a set of Grid actors for access authentication, file copy, job execution, job monitoring, execution reporting, storage access, data discovery, and service discovery.

## 4. Summary and Discussion

We have presented a taxonomy for Grid workflow management systems. The taxonomy focuses on workflow design, workflow scheduling, fault management and data movement. We also surveyed some workflow management systems for Grid computing and classify them into different categories using the taxonomy. This paper thus helps to understand key workflow management approaches and identify possible future enhancements.

Many Grid workflow-enabled systems have developed graph-based editing environments. They allow users to compose the workflow by dragging and dropping components on a composition panel. A workflow abstract specification or concrete specification is then generated by these visual tools and passed to the workflow enactment engine. These processes are transparent to users for better usability. Currently, only Pegasus supports automatic workflow composition. In order to support the automatic composition, catalogs with rich information about application components and services need to be addressed. Besides GriPhyN Chimera system and UDDI (Universal Description, Discovery and Integration) directory service for web services discovery, many efforts from semantic Web such as DAML+OIL ontology [67] can be used for providing accurate description and flexible discovery of application components and services.

Most of the Grid workflow projects discussed in this paper have their own graphical workflow modeling and language. Obviously, the lack of standardized syntax and semantic description for workflow modeling and language results in many replicated works. More effort is thus needed towards workflow

modeling standardization. Even though there are some proposed workflow languages for web services such as BPEL4WS, they are still not sufficient due to lack of implementation, levels of abstraction and limited supported services [9].

Quality of Service (QoS) issues have not been addressed very well in most Grid workflow management systems due to their focus on the use of system centric policies in resource allocation. However, when workflow management systems are used in commercial or production environments, supporting QoS at both specification and execution level becomes increasingly critical. At the specification level, workflow languages need to allow users to express their QoS requirements. At the execution level, the workflow scheduling must be able to map the workflow onto Grid resources to meet users' QoS requirements. Therefore, the role of market-driven strategies will become increasingly important, currently being ignored in most Grid workflow management systems. Trust-based scheduling is another approach to improve QoS in open distributed systems such as Grid and peer-to-peer; however, it has not been addressed very well in the context of workflow management.

It is impossible to make an optimal scheduler without knowledge of estimated time of task execution. Several performance information services are utilized in Grid workflow projects to predict performance prediction. One example is PACE employed in GridFlow project. It uses analytical model to predict application performance, but the current implementation is only adapted to MPI program. Prophesy used by Pegasus uses historical performance database to gain insight into the relationship between applications and resources in order to predict the performance of the applications on a given set of resources. Similarly, ICENI developed a performance repository system which is able to collect performance data for application components. GrADS have developed two analytical models for their GrADS programs.

Given the dynamic nature of Grid environments, fault tolerance should be fully supported by Grid workflow management systems. However, most fault handling techniques have not been developed or implemented in many Grid workflow systems, especially at the workflow execution level. It is hard for a workflow management system to survive in real

Grid environments without robust fault handling techniques.

### Acknowledgements

We would like to acknowledge all developers of the workflow management systems described in the paper. We thank Chee Shin Yeo, Hussein Gibbins, Anthony Sulistio, Srikumar Venugopal, Tianchi Ma, Sushant Goel, Krishna Nadiminti, and Baden Hughes (Melbourne University, Australia), Rob Gray (Monash University, Australia), Wolfram Schiffmann (FernUniversitaet in Hagen, Germany), Ivona Brandic (University of Vienna, Austria), Soonwook Hwang (National Institute of Informatics, Japan), Ewa Deelman (University of Southern California, USA), Chris Mattmann (NASA Jet Propulsion Laboratory, USA), Henan Zhao (University of Manchester, UK), Bertram Ludaescher (University of California, Davis), Thomas Fahringer (University of Innsbruck, Austria), Gregor von Laszewski (Argonne National Laboratory, USA), Ken Kennedy, Anirban Mandal, and Chuck Koelbel (Rice University, USA) for their comments on this paper. We thank anonymous reviewers for their constructive comments. This work is partially supported through the Australian Research Council (ARC) Discovery Project grant and Storage Technology Corporation sponsorship of Grid Fellowship.

### References

1. W.M.P. van der Aalst, K.M. van Hee and G.J. Houben, "Modelling and Analysing Workflow using a Petri-net Based Approach", in *2nd Workshop on Computer-supported Cooperative Work, Petri Nets Related Formalisms*, pp. 31–50, 1994. <http://citeseer.ist.psu.edu/vanderaalst94modelling.html> [December 2004].
2. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski and A.P. Barros, "Workflow Patterns", Technical Report, Eindhoven University of Technology, 2000.
3. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski and A.P. Barros, "Advanced Workflow Patterns", in *CoopIS 2000, Lecture Notes in Computer Science (LNCS) 1901*, Springer, Berlin, Heidelberg, New York, pp. 18–29, 2000.
4. W.M.P. van der Aalst and A.H.M. ter Hofstede, "YAWL: Yet Another Workflow Language", Technical Report, Queensland University of Technology, Brisbane, 2002.

5. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA, 2002.
6. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski and A.P. Barros, "Workflow Patterns", URL: <http://tmitwww.tn.tue.nl/research/patterns/> [December 2004].
7. J.H. Abawajy, "Fault-Tolerant Scheduling Policy for Grid Computing Systems", in *18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, Santa Fe, New Mexico, IEEE Computer Society (CS), Los Alamitos, CA, USA, pp. 238–244, April 26–30, 2004.
8. D. Abramson, J. Giddy and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?" in *14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, Cancun, Mexico, IEEE CS, Los Alamitos, CA, USA, May 1–5, 2000.
9. M. Addis, J. Ferris, M. Greenwood, P. Li, D. Marvin, T. Oimn and A. Wipat, "Experiences with e-Science Workflow Specification and Enactment in Bioinformatics", in *UK e-Science All Hands Meeting 2003*, IOP Publishing Ltd., Bristol, UK, pp. 459–467, 2003.
10. G. Allen, K. Davis, K.N. Dolkas, N.D. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf and I. Taylor, "Enabling Applications on the Grid – A GridLab Overview", in *International Journal of High Performance Computing Applications (JHPCA)*, Special Issue on Grid Computing: Infrastructure and Applications, SAGE Publications Inc., London, UK, August 2003.
11. J. Almond and D. Snelling, "Unicore: Secure and Uniform Access to Distributed Resources via the World Wide Web", White Paper, October 1998, <http://www.fz-juelich.de/zam/RD/coop/unicore/whitepaper.ps> [December 2004].
12. I. Altintas, A. Birnbaum, K. Baldrige, W. Sudholt, M. Miller, C. Amoreira, Y. Potier and B. Ludaescher, "A Framework for the Design and Reuse of Grid Workflows", in *International Workshop on Scientific Applications on Grid Computing (SAG'04)*, LNCS 3458, Springer, Berlin, Heidelberg, New York, 2005.
13. K. Amin and G. von Laszewski, "GridAnt: A Grid Workflow System", Manual, February 2003, <http://www-unix.globus.org/cog/projects/gridant/gridant-manual.pdf> [December 2004].
14. T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic and S. Weerawarana, "Business Process Execution Language for Web Services Version 1.1", 05 May 2003, <http://www-128.ibm.com/developerworks/library/ws-bpel/> [Feb 2005].
15. The Apache Ant Project. <http://ant.apache.org/> [December 2004].
16. D.A. Bacigalupo, S.A. Jarvis, L. He and G.R. Nudd, "An Investigation into the Application of Different Performance Techniques to E-Commerce Applications", in *Workshop on Performance Modelling, Evaluation and Optimization of Parallel and Distributed Systems*, 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Santa Fe, New Mexico, IEEE CS, Los Alamitos, CA, USA, April 26–30, 2004.
17. R. Bastos, D. Dubugras and A. Ruiz, "Extending UML Activity Diagram for Workflow Modeling in Production Systems", in *35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, Big Island, Hawaii, IEEE CS, Los Alamitos, CA, USA, January 07–10, 2002.
18. F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon and R. Wolski, "The GrADS Project: Software Support for High-Level Grid Application Development", *International Journal of High Performance Computing Applications (JHPCA)*, Vol. 15, No. 4, pp. 327–344, SAGE Publications Inc., London, UK, Winter 2001.
19. I. Brandic, S. Benkner, G. Engelbrecht and R. Schmidt, *Towards Quality of Service Support for Grid Workflows*, First European Grid Conference (EGC 2005), Amsterdam, The Netherlands, Feb 2005.
20. T.D. Braun, H.J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys and B. Yao, "A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems", in *17th Symposium on Reliable Distributed Systems*, West Lafayette, IN, IEEE CS, Los Alamitos, CA, pp. 330–335, October 1998.
21. R. Buyya, D., Abramson and J. Giddy, "Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid", *HPC Asia 2000*, Beijing, China, IEEE CS, Los Alamitos, CA, USA, pp. 283–289, May 14–17, 2000.
22. R. Buyya, D. Abramson and J. Giddy, "A Case for Economy Grid Architecture for Service-Oriented Grid Computing", in *10th IEEE International Heterogeneous Computing Workshop (HCW 2001)*, San Francisco, CA, USA, IEEE CS, Los Alamitos, CA, USA, April 2001.
23. R. Buyya and S. Venugopal, "The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report", in *1st IEEE International Workshop on Grid Economics and Business Models*, GECON 2004, Seoul, Korea, IEEE CS, Los Alamitos, CA, USA, pp. 19–36, April 23, 2004.
24. J. Cao, S.A. Jarvis, S. Saini, D.J. Kerbyson and G.R. Nudd, "ARMS: An Agent-based Resource Management System for Grid Computing", *Scientific Programming*, Special Issue on Grid Computing, Vol. 10, No. 2, pp. 135–148, IOS, Amsterdam, Netherlands, 2002.
25. J. Cao, S.A. Jarvis, S. Saini, G.R. Nudd, "GridFlow: Workflow Management for Grid Computing", in *3rd International Symposium on Cluster Computing and the Grid (CCGrid)*, Tokyo, Japan, IEEE CS, Los Alamitos, May 12–15, 2003.
26. J. Cardoso, "Stochastic Workflow Reduction Algorithm", Technical Report, LSDIS Lab, Department of Computer Science University of Georgia, 2002.

27. J. Cardoso and A. Sheth, "Semantic E-Workflow Composition", *Journal of Intelligent Information Systems*, Vol. 21, No. 3, pp. 191–225, Kluwer, Netherlands, 2003.
28. J. Cardoso, J. Miller, A. Sheth and J. Arnold, "Modeling Quality of Service for Workflows and Web Service Processes", *Web Semantics Journal: Science, Services and Agents on the World Wide Web*, Vol. 1, No. 3, pp. 281–308, Elsevier Inc., Massachusetts, USA, 2004.
29. T.L. Casavant and J.G. Kuhl, "A Taxonomy of Scheduling in General-purpose Distributed Computing Systems", *IEEE Transactions on Software Engineering*, Vol. 14, No. 2, pp. 141–154, IEEE CS, Los Alamitos, Feb. 1988.
30. A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Lamnitchi, C. Kesselman, P. Kunst, M. Ripeanu, B. Schwartzkopf, H. Stockinger, K. Stockinger and B. Tierney, "Giggle: A Framework for Constructing Scalable Replica Location Services", in *Supercomputing (SC2002)*, Baltimore, USA, IEEE Computer Society, Washington, DC, USA, November 16–22, 2002.
31. G. Clemm, J.F. Reschke, E. Sedlar and J. Whitehead, "Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol", *The Internet Society*, May 2004.
32. K. Cooper, A. Dasgupta, K. Kennedy, C. Koelbel, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, F. Berman, H. Casanova, A. Chien, H. Dail, X. Liu, A. Olugbile, O. Sievert, H. Xia, L. Johnsson, B. Liu, M. Patel, D. Reed, W. Deng, C. Mendes, Z. Shi, A. YarKhan and J. Dongarra, "New Grid Scheduling and Rescheduling Methods in the GrADS Project", *NSF Next Generation Software Workshop*, International Parallel and Distributed Processing Symposium, Santa Fe, IEEE CS, Los Alamitos, CA, USA, April 2004.
33. D. Crichton, J.S. Hughes and S. Kelly, "A Science Data System Architecture for Information Retrieval", in *Clustering and Information Retrieval*, Kluwer, December 2003.
34. K. Czajkowski, S. Fitzgerald, I. Foster and C. Kesselman, "Grid Information Services for Distributed Resource Sharing", in *10th IEEE International Symposium on High Performance Distributed Computing*, San Francisco, CA, USA: IEEE CS, Los Alamitos, CA, USA, 7–9 August 2001.
35. D. Hollinsworth. The Workflow Reference Model, Workflow Management Coalition, TC00-1003, 1994.
36. DAGMan Application. [http://www.cs.wisc.edu/condor/manual/v6.4/2\\_11DAGman\\_Applications.html](http://www.cs.wisc.edu/condor/manual/v6.4/2_11DAGman_Applications.html) [December 2004].
37. H.J. Dail, "A Modular Framework for Adaptive Scheduling in Grid Application Development Environments", Master's Thesis, UCSD Technical Report CS2002-0698, University of California at San Diego, March 2002.
38. H. Dail, H. Casanova and F. Berman, "A Decoupled Scheduling Approach for the GrADS Program Development Environment", *Journal of Parallel Distributed Computing*, Vol. 63, No. 5, pp. 505–524, Elsevier Inc., MA, USA, 2003.
39. E. Deelman, C. Kesselman and G. Mehta, "Transformation Catalog Design for GriPhyN", Technical Report GriPhyN-2001-17, 2001.
40. E. Deelman, J. Blythe, Y. Gil and C. Kesselman, "Workflow Management in GriPhyN", *The Grid Resource Management*, Kluwer, Netherlands, 2003.
41. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta and K. Vahi, "Mapping Abstract Complex Workflows onto Grid Environments", *Journal of Grid Computing*, Vol. 1, pp. 25–39, Kluwer, Netherlands, 2003.
42. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.H. Su, K. Vahi and M. Livny, "Pegasus: Mapping Scientific Workflow onto the Grid", in *Across Grids Conference 2004*, Nicosia, Cyprus, 2004.
43. P.A. Dinda, "Online Prediction of the Running Time of Tasks", *Cluster Computing*, Vol. 5, No. 3, pp. 225–236, Kluwer, Netherlands, 2002.
44. dom4j. <http://www.dom4j.org> [December 2004].
45. M. Dumas and A.H.M. ter Hofstede, "UML Activity Diagrams as a Workflow Specification Language", in *UML'2001 Conference*, Toronto, Ontario, Canada, Lecture Notes in Computer Science (LNCS), Springer, Berlin, Heidelberg, New York, October 1–5, 2001.
46. R. Eshuis and R. Wieringa, "Comparing Petri Net and Activity Diagram Variants for Workflow Modelling – A Quest for Reactive Petri Nets", *Advances in Petri Nets: Petri Net Technology for Communication Based Systems*, Lecture Notes in Computer Science (LNCS), Vol. 2472, pp. 321–351, Springer, Berlin, Heidelberg, New York, 2003.
47. T. Fahringer, S. Pllana and A. Villazon, "AGWL: Abstract Grid Workflow Language", in *International Conference on Computational Science, Programming Paradigms for Grids and Meta-computing Systems*, Krakow, Poland, Springer, Berlin, Heidelberg, New York, June 2004.
48. T. Fahringer, S. Pllana and J. Testori, "Teuta: Tool Support for Performance Modeling of Distributed and Parallel Applications", in *International Conference on Computational Science, Tools for Program Development and Analysis in Computational Science*, Krakow, Poland, Springer, Berlin, Heidelberg, New York, Heidelberg, June 2004.
49. T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C.S. Jr. and H.L. Truong, "ASKALON: A Tool Set for Cluster and Grid Computing", *Concurrency and Computation: Practice and Experience*, Vol. 17, pp. 143–169, Wiley, 2005.
50. D. Fernández-Baca, "Allocating Modules to Processors in a Distributed System", *IEEE Transactions on Software Engineering*, Vol. 15, No. 11, pp. 1427–1436, November 1989.
51. I. Foster and C. Kesselman (eds.). *The Grid: Blueprint for a Future Computing Infrastructure*, Morgan Kaufmann, USA, 1999.
52. I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of Supercomputing Applications*, Vol. 15, No. 3, 2001.
53. I. Foster, J. Vöckler, M. Wilde, Y. Zhao, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation", in *14th International Conference on Scientific and Statistical Database Manage-*

- ment (SSDBM), Edinburgh, Scotland, UK: IEEE CS, Los Alamitos, CA, USA, July 24–26, 2002.
54. I. Foster, C. Kesselman, J.M. Nick and S. Tuecke, “The Physiology of the Grid”, Technical Report, Globus Project, <http://www.globus.org/research/papers/ogsa.pdf> [December 2004].
  55. Freefluo Overview. <http://freefluo.sourceforge.net/> [December 2004].
  56. D. Gelernter, “Generative Communication in Linda”, *ACM Computing Surveys*, Vol. 7, No. 1, pp. 80–112, 1985.
  57. A. Galstyan, K. Czajkowski and K. Lerman, “Resource Allocation in the Grid Using Reinforcement Learning”, in *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS’03)*, New York City, NY, USA, IEEE CS, Los Alamitos, CA, USA, July 19–23, 2004.
  58. A. Geppert, M. Kradolfer and D. Tombros, “Market-based Workflow Management”, *International Journal of Cooperative Information Systems*, World Scientific, New Jersey, USA, 1998.
  59. Globus Project. <http://www.globus.org> [December 2004].
  60. Grid Resource Allocation Agreement Protocol. <https://forge.gridforum.org/projects/graap-wg> [December 2004].
  61. GriPhyN. <http://www.griphyn.org> [December 2004].
  62. Z. Guan, F. Hernandez, P. Bangalore, J. Gray, A. Skjellum, V. Velusamy and Y. Liu, “Grid-Flow: A Grid-Enabled Scientific Workflow System with a Petri Net-based Interface”, Technical Report, <http://http://www.cis.uab.edu/gray/Pubs/grid-flow.pdf> [December 2004].
  63. V. Hamscher, U. Schwiegelshohn, A. Streit and R. Yahyapour, “Evaluation of Job-Scheduling Strategies for Grid Computing”, in *1st IEEE/ACM International Workshop on Grid Computing (Grid 2000)*, Berlin, Lecture Notes in Computer Science (LNCS), Springer, Berlin, Heidelberg, New York, pp. 191–202, 2000.
  64. F. Hernández, P. Bangalore, J. Gray and K. Reilly, “A Graphical Modeling Environment for the Generation of Workflows for the Globus Toolkit”, in *Workshop on Component Models and Systems for Grid Applications, 18th Annual ACM International Conference on Supercomputing (ICS 2004)*, Saint-Malo, France, ACM, New York, NY, USA, June 2004.
  65. A. Hoheisel. User Tools and Languages for Graph-based Grid Workflows. *Grid Workflow Workshop*, GGF10, Berlin, March 9, 2004.
  66. S. Hwang and C. Kesselman, “Grid Workflow: A Flexible Failure Handling Framework for the Grid”, in *12th IEEE International Symposium on High Performance Distributed Computing (HPDC’03)*, Seattle, Washington, USA, IEEE CS, Los Alamitos, CA, USA, June 22–24, 2003.
  67. I. Horrocks, “DAML+OIL: A Reason-able Web Ontology Language”, in *International Conference on Extending Database Technology (EDBT 2002)*, Lecture Notes in Computer Science (LNCS), pp. 11–28, Vol. 1091, Springer, Berlin, Heidelberg, New York, pp. 2–13, March 24–28, 2002.
  68. S. Jang, X. Wu, V. Taylor, G. Mehta, K. Vahi and E. Deelman, “Using Performance Prediction to Allocate Grid Resources”, Technical Report 2004-25, GriPhyN Project, USA.
  69. JDOM. <http://www.jdom.org> [December 2004].
  70. JXTA Project. <http://www.jxta.org> [Feb 2005].
  71. P. Kacsuk, G. Dózsa, J. Kovács, R. Lovas, N. Podhorszki, Z. Balaton and G. Gombás, “P-GRADE: a Grid Programming Environment”, *Journal of Grid Computing*, Vol. 1, No. 2, pp. 171–197, Kluwer, Netherlands, 2003.
  72. B. Kao and H. Garcia-Molina, “Deadline Assignment in a Distributed Soft Real-Time System”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 12, pp. 1268–1274, IEEE CS, Los Alamitos, CA, USA, 1997.
  73. K. Krauter, R. Buyya and M. Maheswaran, “A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing”, *Software: Practice and Experience*, Vol. 32, No. 2, pp. 135–164, Wiley, NJ, USA, February 2002.
  74. 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, Wiley, Chichester, UK, 2001.
  75. G. von Laszewski, K. Amin, M. Hategan, N.J. Zaluzec, S. Hampton and A. Rossi, “GridAnt: A Client-Controllable Grid Workflow System”, in *37th Annual Hawaii International Conference on System Sciences (HICSS’04)*, Big Island, Hawaii: IEEE CS, Los Alamitos, CA, USA, January 5–8, 2004.
  76. G. Von Laszewski, “Java CoG Kit Workflow Concepts for Scientific Experiments”, Technical Report, Argonne National Laboratory, Argonne, IL, USA, 2005.
  77. G. von Laszewski and M. Hategan, “Java CoG Kit Karajan/ GridAnt Workflow Guide”, Technical Report, Argonne National Laboratory, Argonne, IL, USA, 2005.
  78. A. Lerina, C. Aniello, G. Pierpaolo and V.M. Luisa, “FlowManager: A Workflow Management System Based on Petri Nets”, in *26th Annual International Computer Software and Applications Conference*, Oxford, England, IEEE CS, Los Alamitos, CA, USA, pp. 1054–1059, August 2002.
  79. F. Leymann. Web Services Flow Language (WSFL 1.0), May 2001, <http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf> [December 2004].
  80. D.C. Li and N. Ishii, “Scheduling Task Graphs onto Heterogeneous Multiprocessors”, *TENCON’94*, IEEE Region 10’s Ninth Annual International Conference, Theme: Frontiers of Computer Technology, IEEE CS, Los Alamitos, CA, USA, 1994.
  81. M. Litzkow, M. Livny and M. Mutka, “Condor – A Hunter of Idle Workstations”, in *8th International Conference of Distributed Computing Systems (ICDCS)*, IEEE CS, Los Alamitos, CA, USA, pp. 104–111, June 1988.
  82. X. Liu, J. Liu, J. Eker and E.A. Lee, “Heterogeneous Modeling and Design of Control Systems”, in Tariq Samad and Gary Balas (eds.), *Software-Enabled Control: Information Technology for Dynamical Systems*, Wiley-IEEE, April 2003.

83. R. Lovas, G. Dózsa, P. Kacsuk, N. Podhorszki and D. Drótos, "Workflow Support for Complex Grid Applications: Integrated and Portal Solutions", in *2nd European Across Grids Conference*, Nicosia, Cyprus, 2004.
84. B. Ludäscher, I. Altintas and A. Gupta, "Compiling Abstract Scientific Workflows into Web Service Workflows", in *15th International Conference on Scientific and Statistical Database Management*, Cambridge, MA, USA, IEEE CS, Los Alamitos, CA, USA, pp. 241–244, July 09–11, 2003.
85. B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E.A. Lee, J. Tao and Y. Zhao, "Scientific Workflow Management and the KEPLER System", *Concurrency and Computation: Practice & Experience*, 2005 (in press), Published online in Wiley InterScience ([www.interscience.wiley.com](http://www.interscience.wiley.com)). DOI:10.1002/cpe. 994.
86. A. Luther, R. Buyya, R. Ranjan and S. Venugopal, "Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework", in Laurence Yang and Minyi Guo (eds.), *High Performance Computing: Paradigm and Infrastructure*, ISBN: 0-471-65471-X, Wiley, NJ, USA, June 2005.
87. M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen and R. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems", in *8th Heterogeneous Computing Workshop (HCW'99)*, Juan, Puerto Rico, IEEE Computer Society, Los Alamitos, April 12, 1999.
88. A. Mani and A. Nagarajan, "Understanding Quality of Service for Web Services", <http://www-106.ibm.com/developerworks/library/ws-quality.html> [December 2004].
89. D.C. Marinescu, "A Grid Workflow Management Architecture", GGF White Paper, 2002.
90. G. Mateescu, "Quality of Service on the Grid via Metascheduling with Resource Co-scheduling and Co-reservation", *International Journal of High Performance Computing Applications*, Vol. 17, No. 3, pp. 209–218, SAGE Publications Inc, London, UK, August 2003.
91. A. Mayer, S. McGough, N. Furmento, W. Lee, S. Newhouse and J. Darlington, "ICENI Dataflow and Workflow: Composition and Scheduling in Space and Time", in *UK e-Science All Hands Meeting*, Nottingham, UK, IOP, Bristol, UK, pp. 627–634, September 2003.
92. A. Mayer, S. McGough, N. Furmento, W. Lee, M. Gulamali, S. Newhouse and J. Darlington, "Workflow Expression: Comparison of Spatial and Temporal Approaches", in *Workflow in Grid Systems Workshop*, GGF-10, Berlin, March 9, 2004.
93. S. McGough, L. Young, A. Afzal, S. Newhouse and J. Darlington, "Workflow Enactment in ICENI", in *UK e-Science All Hands Meeting*, Nottingham, UK, IOP, Bristol, UK, pp. 894–900, Sep. 2004.
94. S. McGough, L. Young, A. Afzal, S. Newhouse and J. Darlington, "Performance Architecture within ICENI", in *UK e-Science All Hands Meeting*, Nottingham, UK, IOP, Bristol, UK, pp. 906–911, Sep. 2004.
95. Message Passing Interface Forum, <http://www.mpi-forum.org/> [Feb 2005].
96. R.A. Moreno, "Job Scheduling and Resource Management Techniques in Dynamic Grid Environment", in *1st European Across Grids Conference*, Spain, Lecture Notes in Computer Science (LNCS), Springer, Berlin, Heidelberg, New York, February 2003.
97. T. Murata, "Temporal Uncertainty and Fuzzy-Timing High-Level Petri Nets", in *Application and Theory of Petri Nets*, Lecture Notes in Computer Science (LNCS), Vol. 1091, pp. 11–28, Springer, Berlin, Heidelberg, New York, 1996.
98. G.R. Nudd, D.J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper and D.V. Wilcox, "PACE-A Toolset for the Performance Prediction of Parallel and Distributed Systems", *International Journal of High Performance Computing Applications (JHPCA)*, Special Issues on Performance Modelling-Part I, Vol. 14, No. 3, pp. 228–251, SAGE, London, UK, 2000.
99. Object Management Group. Unified Modeling Language (UML), <http://www.uml.org/> [Feb 2005].
100. T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver and K. Glover, M.R. Pocock, A. Wipat and P. Li, "Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows", *Bioinformatics*, Vol. 20, No. 17, pp. 3045–3054, Oxford University Press, London, UK, 2004.
101. T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, A. Wipat and P. Li. Taverna, "Lessons in Creating a Workflow Environment for the Life Sciences", *GGF10*, Berlin, 2004.
102. OMG. Unified Modeling Language Version 1.3., July 1999.
103. C. Patel, K. Supekar and Y. Lee, "A QoS Oriented Framework for Adaptive Management of Web Service based Workflows", *Lecture Notes in Computer Science*, Vol. 2736, pp. 826–835, Springer, Berlin, Heidelberg, New York, 2003.
104. C.A. Petri, "Kommunikation mit Automaten", PhD Thesis, Institut für instrumentelle Mathematik, Bonn, 1962.
105. S. Pillana, T. Fahringer, J. Testori, S. Benkner and I. Brandic, "Towards an UML Based Graphical Representation of Grid Workflow Applications", in *2nd European AcrossGrids Conference (AxGrids 2004)*, Nicosia, Cyprus, LNCS, Springer, Berlin, Heidelberg, New York, January 28–30, 2004.
106. R. Prodan and T. Fahringer, "Dynamic Scheduling of Scientific Workflow Applications on the Grid: A Case Study", in *20th Annual ACM Symposium on Applied Computing (SAC 2005)*, New Mexico, USA, ACM, New York, NY, USA, March 2005.
107. R.L. Ribler, H. Simitci and D.A. Reed, "The Autopilot Performance-directed Adaptive Control System", *Future Generation Computer Systems*, Vol. 18, No. 1, pp. 175–187, Elsevier Inc., MA, USA, 2001.
108. H.G. Rotthor, "Taxonomy of Dynamic Task Scheduling Schemes in Distributed Computing Systems", *IEE Proceedings of Computers and Digital Techniques*, Vol. 141, No. 1, pp. 1–10, London, UK, January 1994.

109. S. Fitzgerald, I. Foster, C. Kesselman, G. Von Laszewski, W. Smith and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations", in *6th IEEE Symposium on High-Performance Distributed Computing*, Portland, OR, IEEE CS, Los Alamitos, pp. 365–375, August 1997.
110. R. Sakellariou and H. Zhao, "A Low-Cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems", *Scientific Programming*, Vol. 12, No. 4, pp. 253–262, IOS, Netherlands, December 2004.
111. M. Senger, P. Rice and T. Oinn, "Soaplab – A Unified Sesame Door to Analysis Tools", in *UK e-Science All Hands Meeting*, pp. 509–513, September 2003.
112. A. Slominski, D. Gannon and G. Fox, "Introduction to Workflows and Use of Workflows in Grids and Grid Portals", *GGF 9*, Chicago, USA, 7 Oct, 2004.
113. W. Smith, I. Foster and V. Taylor, "Predicting Application Run Times Using Historical Information", in *Workshop on Job Scheduling Strategies for Parallel Processing*, 12th International Parallel Processing Symposium & 9th Symposium on Parallel and Distributed Processing (IPPS/SPDP '98), IEEE CS, Los Alamitos, CA, USA, 1998.
114. S.S. Song and K. Hwang, "Security Binding for Trusted Job Outsourcing in Open Computational Grids", *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, submitted May 2004, revised Dec. 2004.
115. S.S. Song, Y.K. Kwok and K. Hwang, "Trusted Job Scheduling in Open computational Grids: Security-Driven heuristics and A Fast Genetic Algorithm", in *19th IEEE International Parallel & Distributed Processing Symposium (IPDPS-2005)*, Denver, CO, USA, IEEE Computer Society, Los Alamitos, CA, USA, April 4–8, 2005.
116. D.P. Spooner, J. Cao, J.D. Turner, H.N. Lin Chio Keung, S.A. Jarvis and G.R. Nudd, "Localized Workload Management Using Performance Prediction and QoS Contracts", in *18th Annual UK Performance Engineering Workshop*, Glasgow, UK, pp. 69–80, 2002.
117. D.P. Spooner, J. Cao, S.A. Jarvis, L. He and G.R. Nudd, "Performance-aware Workflow Management for Grid Computing", *The Computer Journal*, Oxford University Press, London, UK, 2004.
118. R.D. Stevens, A.J. Robinson and C.A. Goble, "myGrid: Personalized Bioinformatics on the Information Grid", *Bioinformatics*, Vol. 19, Suppl. 1, pp. i302–i304, Oxford University Press, London, UK, 2003.
119. A. Sulistio and R. Buyya, "A Grid Simulation Infrastructure Supporting Advance Reservation", in *16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*, MIT Cambridge, Boston, USA, ACTA, CA, USA, November 9–11, 2004.
120. T. Tannenbaum, D. Wright, K. Miller and M. Livny, "Condor – A Distributed Job Scheduler", *Beowulf Cluster Computing with Linux*, MIT, MA, USA, 2002.
121. Taverna User Manual. <http://taverna.sourceforge.net/manual/docs.word.html> [December 2004].
122. I. Taylor, R. Philp, M. Shields, O. Rana and B. Schutz, "The Consumer Grid", in *Global Grid Forum (2002)*, Toronto, Ontario, Canada, February 17–20, 2002.
123. I. Taylor, M. Shields and I. Wang, "Resource Management of Triana P2P Services", *Grid Resource Management*, Kluwer, Netherlands, June 2003.
124. D. Thain, T. Tannenbaum and M. Livny, "Condor and the Grid", *Grid Computing: Making the Global Infrastructure a Reality*, Wiley, New Jersey, USA, 2003.
125. S. Thatte. XLANG-Web Services for Business Process Design, Microsoft Corporation, 2001, [http://www.gotdotnet.com/team/xml\\_wsspecs/xlang-c/default.htm](http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm) [Feb 2005].
126. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt and D. Snelling, "Open Grid Services Infrastructure (OGSI) Version 1.0", *Global Grid Forum Draft Recommendation*, June 27, 2003.
127. UDDI Technical White Paper, September 2000, <http://www.uddi.org> [December 2004].
128. Unicore Forum. Unicore Plus Final Report: Uniform Interface to Computing Resource. 2003, <http://www.unicore.org/documents/UNICOREPlus-Final-Report.pdf> [December 2004].
129. S. Vadhiyar and J. Dongarra, "A Performance Oriented Migration Framework for the Grid", in *IEEE Computing Clusters and the Grid (CCGrid)*, Tokyo, Japan, IEEE CS, Los Alamitos, May 12–15, 2003.
130. S. Venugopal, R. Buyya and L. Winton, "A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids", in *2nd International Workshop on Middleware for Grid Computing*, Middleware 2004, Toronto, Ontario-Canada, ACM, New York, NY, USA, October 18, 2004.
131. H.M.W. Verbeek, A. Hirschall and W.M.P. van der Aalst, "XRL/Flower: Supporting Inter-Organizational Workflows Using XML/Petri-nets Technology", in *Workshop on Web Services, e-Business, and the Semantic Web (WES): Foundations, Models, Architecture, Engineering and Applications*, The Fourteenth International Conference on Advanced Information Systems Engineering (CAiSE 2002), Toronto, Ontario, Canada, Lecture Notes in Computer Science (LNCS), Springer, Berlin, Heidelberg, New York, pp. 535–552, May 27–28, 2002.
132. W3C. Extensible Markup Language (XML) 1.0 (Third Edition), <http://www.w3.org/TR/REC-xml/> [Feb 2005].
133. W3C. Web Services, 2002, <http://www.w3.org/2002/ws/> [Feb 2005].
134. W3C. XML Schema, <http://www.w3.org/XML/Schema> [Feb 2005].
135. W3C. XML Pipeline Definition Language Version 1.0, <http://www.w3.org/TR/2002/NOTE-xml-pipeline-20020228/> [Feb 2005].
136. R. Wolski, N.T. Spring and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", *Future Gen-*

- eration Computer Systems, Vol. 15, No. 5–6, pp. 757–768, 1999.
137. Workflow Management Coalition, <http://www.wfmc.org/> [December 2004].
  138. World Wide Web Consortium. Web Services Description Language (WSDL) Version 1.2, <http://www.w3.org/TR/wsdl12> [December 2004].
  139. C. Wroe, R. Stevens, C. Goble, A. Roberts and M. Greenwood, “A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data”, *International Journal of Cooperative Information Systems*, Vol. 12, No. 2, pp. 197–224, World Scientific Publishing Co., NJ, USA, 2003.
  140. X.F. Wu, V. Taylor and R. Stevens, “Design and Implementation of Prophesy Automatic Instrumentation and Data Entry System”, in *13th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS2001)*, Anaheim, CA, IASTED, Philadelphia, PA, USA, August 2001.
  141. R. Yahyapour, P. Wieder, A. Pugliese, D. Talia and J. Hahm, “Grid Scheduling Use Cases”, White Paper, Global Grid Forum, 19 July, 2004.
  142. L. Young and J. Darlington, “Scheduling Componentized Applications on a Computational Grid”, MPhil/PhD Transfer Report, Imperial College London, University of London, UK, 2004.
  143. L. Young, S. McGough, S. Newhouse and J. Darlington, “Scheduling Architecture and Algorithms within the ICENI Grid Middleware”, in *UK e-Science All Hands Meeting*, IOP, Bristol, UK, Nottingham, UK, pp. 5–12, Sep. 2003.
  144. J. Yu and R. Buyya, “A Novel Architecture for Realizing Grid Workflow using Tuple Spaces”, in *5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, Pittsburgh, USA, IEEE CS, Los Alamitos, CA, USA, Nov. 8, 2004.
  145. J. Yu, S. Venugopal and R. Buyya, “A Market-Oriented Grid Directory Service for Publication and Discovery of Grid Service Providers and their Services”, Technical Report, GRIDS-TR-2003-0, Grid Computing and Distributed Systems (GRIDS) Laboratory, The University of Melbourne, Australia, January 2003.
  146. J. Yu, R. Buyya and C.K. Tham, “QoS-based Scheduling of Workflow Applications on Service Grids”, Technical Report, GRIDS-TR-2005-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, June 9, 2005.
  147. S.Y. Zhao and V. Lo, “Result Verification and Trust-based Scheduling in Open Peer-to-Peer Cycle Sharing Systems”, *Technical Report*, University of Oregon, USA, 2005.
  148. G. Zheng, T. Wilmarth, P. Jagadishprasad and L.V. Kalé, “Simulation-based Performance Prediction for Large Parallel Machines”, *International Journal of Parallel Programming*, Vol. 33, No. 2–3, pp. 183–207, Springer Academic Publishers, The Netherlands, 2005.