

Chapter 13

Scientific Process Automation and Workflow Management*

Bertram Ludäscher¹, Ilkay Altintas², Shawn Bowers¹, Julian Cummings³,
Terence Critchlow⁴, Ewa Deelman⁵, David De Roure⁶, Juliana Freire¹⁰,
Carole Goble⁷, Matthew Jones⁸, Scott Klasky⁹, Timothy McPhillips¹,
Norbert Podhorszki⁹, Claudio Silva¹⁰, Ian Taylor¹¹, Mladen Vouk¹²

¹University of California, Davis

²San Diego Supercomputer Center

³California Institute of Technology, Pasadena

⁴Pacific Northwest National Laboratory

⁵USC Information Science Institute, Marina del Rey

⁶University of Southampton, United Kingdom

⁷The University of Manchester, United Kingdom

⁸University of California, Santa Barbara

⁹Oak Ridge National Laboratory

¹⁰University of Utah

¹¹Cardiff University, United Kingdom

¹²North Carolina State University

Abstract. We introduce and describe *scientific workflows*, *i.e.*, executable descriptions of automatable scientific processes such as computational science simulations and data analyses. Scientific workflows are often expressed in terms of tasks and their (dataflow) dependencies. This chapter first provides an overview of the characteristic features of scientific workflows and outlines their life cycle. A detailed case study highlights workflow challenges and solutions in simulation management. We then provide a brief overview of how some concrete systems support the various phases of the workflow life cycle, *i.e.*, design, resource management, execution, and provenance management. We conclude with a discussion on community-based workflow sharing.

Contents

13 Scientific Process Automation and Workflow Management*	0
13.1 Introduction	1
13.2 Features of Scientific Workflows	2
13.2.1 The Scientific Workflow Life Cycle	2
13.2.2 Types of Scientific Workflows	4
13.2.3 Models of Computation	4
13.2.4 Benefits of Scientific Workflows	5
13.3 Case Study: Fusion Simulation Management	5
13.3.1 Overview of the Simulation Monitoring Workflow	6
13.3.2 Issues in Simulation Management	8
13.4 Grid Workflows and the Scientific Workflow Life Cycle	10
13.4.1 Workflow Design and Composition	10
13.4.2 Mapping Workflows to Resources	12
13.4.3 Workflow Execution	14
13.5 Workflow Provenance and Execution Monitoring	15
13.5.1 Example Implementation of a Provenance Framework	17
13.6 Workflow Sharing and myExperiment	18
13.6.1 Workflow Reuse	19
13.6.2 Social Sharing	20
13.6.3 Realizing myExperiment	20
13.7 Conclusions and Future Work	21

13.1 Introduction

Scientific discoveries in the natural sciences are increasingly data-driven and computationally intensive, providing unprecedented data analysis and scientific simulation opportunities. To accelerate scientific discovery through advanced computing and information technology, various research programs have been launched in recent years, *e.g.*, the SciDAC program by the Department of Energy [Sci] and the Cyberinfrastructure initiative by the National Science Foundation [nsf], both in the United States. In the UK, the term *e-Science* [Esc] was coined to describe computationally and data-intensive science, and a large e-Science research program was started there in 2000. With the new opportunities for scientists also come new challenges, *e.g.*, managing the enormous amounts of data generated [And08] and the increasingly sophisticated but also more complex computing environments provided by cluster computers and distributed Grid environments. Scientific workflows aim to address many of these challenges.

In general terms, a *scientific workflow* is a formal description of a process for accomplishing a scientific objective, usually expressed in terms of tasks and their dependencies [LBM09, TDGS07, DGST09]. Scientific workflows can be used during several different phases of a larger science process, *i.e.*, the cycle of hypothesis formation, experiment design, execution, and data analysis [GDE⁺07, LWMB09]. Scientific workflows can include steps for the acquisition, integration, reduction, analysis, visualization, and publication (*e.g.*, in

a shared database) of scientific data. Similar to more conventional business workflows [Wes07], scientific workflows are composed of individual tasks that are organized at workflow design time and whose execution is orchestrated at runtime according to dataflow and task dependencies as specified by the workflow designer. Workflows are often designed visually, *e.g.*, using block diagrams, or textually using a domain-specific language. From a scientist’s perspective, scientific workflows constitute knowledge artifacts or “recipes” that provide a means to automate, document, and make repeatable a scientific process.

The primary task of a *scientific workflow system* is to automate the execution of scientific workflows. Scientific workflow systems may additionally support users in the design, composition, and verification of scientific workflows. They also may include support for monitoring the execution of workflows in real-time; recording the processing history of data; planning resource allocation in distributed execution environments; discovering existing workflows and workflow components; recording the lineage of data and evolution of workflows; and generally managing scientific data. Thus, a scientific workflow system primarily serves as a workflow *execution engine*, but may also include features of *problem-solving environments* (PSE) [RB96].

Wainer et al. describe some of the differences between business (or “office automation”) workflows and scientific workflows, stating “*whereas office work is about goals, scientific work is about data*” [WWVM96]. Business workflows are mainly concerned with the modeling of business rules, policies, and case management, and therefore are often control- and activity-oriented. In contrast, to support the work of computational scientists, scientific workflows are mainly concerned with capturing scientific data analysis or simulation processes and the associated management of data and computational resources. While scientific workflow technology and research can inherit and adopt techniques from the field of business workflows, there are several, sometimes subtle differences [LWMB09], ranging from the modeling paradigms used, to the underlying computation models employed to execute workflows. For example, scientific workflows are usually dataflow-oriented “analysis pipelines” that often exhibit pipeline parallelism over data streams in addition to supporting the data parallelism and task parallelism common in business workflows.¹ In some cases (*e.g.*, in seismic or geospatial data processing [RG08]), scientific workflows execute as digital signal processing (DSP) pipelines. In contrast, traditional workflows often deal with case management (*e.g.*, insurance claims, mortgage applications *etc.*), tend to be more control-intensive, and lend themselves to very different models of computation.

In Section 13.2 we introduce basic concepts and describe key characteristics of scientific workflows. In Section 13.3 we provide a detailed case study from a fusion simulation project where scientific workflows are used to manage complex scientific simulations. Section 13.4 describes scientific workflow systems currently in use and in development. Section 13.5 introduces and discusses basic notions of data and workflow provenance in the scientific workflow context, and describes how workflow systems monitor execution and manage provenance. Finally, Section 13.6 describes approaches for enabling workflow reuse, sharing, and collaboration.

13.2 Features of Scientific Workflows

13.2.1 The Scientific Workflow Life Cycle

The various phases and steps associated with developing, deploying, and executing scientific workflows comprise the scientific workflow life cycle. The following phases are largely supported by existing workflow systems using a wide variety of approaches and techniques (cf. Section 13.4).

Workflow Design and Composition. Development of scientific workflows usually starts with gathering requirements from scientists. A specification of the desired workflow functionality is then developed, and

¹In the parallel computing literature, *task parallelism* refers to distributing tasks (processes) across different parallel computing nodes, while *data parallelism* involves distributing data across multiple nodes. *Pipeline parallelism* is a more specific condition that arises whenever multiple processes arranged in a linear sequence execute simultaneously.

an actual workflow is assembled based on this specification. Workflow development differs from general programming in many ways. Most notably, it usually amounts to the composition and configuration of a special-purpose workflow from pre-existing, more general-purpose components, subworkflows, and services. Workflow development thus more closely resembles script- and shell-based programming than conventional application development. During workflow composition, the user (a scientist or workflow developer) either creates a new workflow by modifying an existing one (cf. *workflow evolution*, Section 13.5), or else composes a new workflow from scratch using components and subworkflows obtained from a repository. In contrast to the business workflow world, where standards have been developed over the years (e.g., most recently WS-BPEL 2.0 [JE07]), scientific workflow systems tend to use their own, internal languages and exchange formats (e.g., SCUFL [Tav], GPEL [WHH05], and MOML [BLL⁺08] among others). Reasons for this diversity include the wide range of computation models used in scientific workflows (Section 13.2.3), and the initial focus of development efforts on scientist-oriented functionality rather than standardization.

Workflow Resource Planning. Once the workflow description is constructed, scientific workflow systems often provide various functions prior to execution. These functions may include workflow validation (e.g., type checking), resource allocation, scheduling, optimization, parameter binding, and data staging. *Workflow mapping* is sometimes used to refer to optimization and scheduling decisions made during this phase. In particular, during workflow design and composition, the target resources to be used for execution are typically not chosen. Workflow mapping then refers to the process of generating an *executable workflow* based on a resource-independent *abstract workflow* description [DBG⁺03]. In some cases, the user performs the mapping directly by selecting appropriate resources (e.g., in Figure 13.2). In other cases, the workflow system automatically performs the mapping. In the latter case, users are allowed to construct workflows at a level of abstraction above that of the target execution environment.

Workflow Execution. Once a workflow is mapped and data has been staged (selected and made available to the workflow system), the workflow can be executed. During execution, a workflow system may record provenance information (data and process history, see Chapter 12 and Section 13.5) as well as provide real-time monitoring and failover functions. Depending on the system, provenance information generally involves the recording of the steps that were invoked during workflow execution, the data consumed and produced by each step, a set of data dependencies stating which data was used to derive other data, the parameter settings used for each step, and so on. If a workflow can change while executing (e.g., due to changing resource availability), the evolution of such a dynamic workflow may be recorded as well in order to support subsequent execution analysis.

Workflow Execution Analysis. After workflow execution, scientists often need to inspect and interpret workflow results. This involves evaluation of data products (*does this result make sense?*) examination of workflow execution traces (*is this how the result should have arisen?*), workflow debugging (*what went wrong here?*), and performance analysis (*why did this take so long?*).

Workflow and Result Sharing. Data and workflow products can be published and shared. As workflows and data products are committed to a shared repository, new iterations of the workflow life cycle can begin.

User Roles. Users of scientific workflow systems can play a number of different roles within the above phases: A *workflow designer* is usually a scientist who develops a new experimental or analytical protocol (or a new variant of an existing method). As mentioned above, a workflow design is often elicited through some form of requirements analysis, and the design and associated requirements can be used by a *workflow engineer* to implement the associated abstract or executable workflow description. A *workflow operator* is a user who executes workflows on the desired inputs. An operator may launch a workflow directly via a scientific workflow system, or indirectly through another application (e.g., within a web portal), monitor the execution (e.g., via a workflow dashboard), and subsequently validate results based on stored provenance

information. The above user roles are not necessarily disjoint, *e.g.*, a single person may assume the roles of designer, engineer, and operator. Indeed, scientific workflow systems aim at making workflow design, execution, and result analysis all easier in comparison to traditional script-based approaches to scientific process automation.

13.2.2 Types of Scientific Workflows

Scientific workflows can be used to model and automate scientific processes from many different science domains (*e.g.*, particle physics, bioinformatics, ecology, and cosmology, to name a few). Not surprisingly, such workflows can exhibit very different characteristics. For example, workflows might be *exploratory* in nature, starting from ad-hoc designs and then requiring frequent changes to the workflow design, parameter settings, *etc.* to determine which methods and components are most suitable for the particular datasets under investigation. Such exploratory workflow design is common when developing new analysis methods. Conversely, some applications require the development of *production workflows* to be executed on a regular basis with new datasets or simulation parameters (*e.g.*, environmental monitoring and analysis workflows or the fusion simulation workflow in Section 13.3).

Another important distinction has to do with what the workflow components (called *actors* or *tasks*) represent and model. In *science-oriented* workflows, actors model a scientific method or process. In such workflows individual workflow steps generally are meaningful to the scientist, *i.e.* more or less directly correspond to high-level steps of the scientific method being automated. Contrasting with science-oriented workflows are *resource-oriented* workflows. Actors and workflow steps in the latter model require data and resource handling tasks rather than the science. In such cases, the actual analytical or simulation operations might be “hidden” from the workflow system, and instead the workflow directly handles the “plumbing” tasks such as data movement, data replication, and job management (submit, pause, resume, abort, *etc.*) The simulation management workflow in Section 13.3 is an example of such a resource-oriented “plumbing workflow.”

13.2.3 Models of Computation

Consider a workflow graph W consisting of *actors* (tasks, workflow steps) and *connections* (directed edges) between them.² With W we can associate a set of *parameters* \bar{p} , input datasets \bar{x} , and output datasets \bar{y} . A *model of computation* (MoC) M prescribes how to execute the parameterized workflow $W_{\bar{p}}$ on \bar{x} to obtain \bar{y} . Therefore, we can view a MoC as a mapping $M: \mathcal{W} \times \bar{P} \times \bar{X} \rightarrow \bar{Y}$ which for any workflow $W \in \mathcal{W}$, parameter settings $\bar{p} \in \bar{P}$, and inputs $\bar{x} \in \bar{X}$, uniquely determines the workflow outputs $\bar{y} \in \bar{Y}$. We denote this by $\bar{y} = M(W_{\bar{p}}(\bar{x}))$. While most current scientific workflow systems employ a single MoC, the Kepler system [Kep], due to its heritage from Ptolemy [BLL⁺08], supports more than one such MoC: for each MoC M , there is a corresponding *director* of the same name which implements M .

For example, consider the PN (Process Network) model of computation. Using the PN director in Kepler, a workflow W executes as a dataflow process network [Kah74, LP95]. In PN each actor executes as a separate, data-driven process (or thread) which is *continuously* running. Actor connections in PN correspond to unidirectional channels (modeled as unbounded queues) over which ordered token streams are sent, and actors in PN block (wait) only when there are not enough tokens available on the actor’s input ports. Process networks naturally support pipeline parallelism as well as task and data parallelism.

In SDF (Synchronous Data-Flow), each actor has fixed token consumption and production rates. In Kepler this allows the SDF director to construct an *actor firing schedule* prior to executing the workflow [LM08]. This also allows the SDF director readily to execute workflows in a single thread, firing actors one at a time based on the schedule.

Workflows employing the PN and SDF directors in Kepler may include cycles in the workflow graph. We use the term DAG to refer to a model of computation that restricts the workflow graph W to a directed, *acyclic* graph of task dependencies. In DAG each actor node in W is executed only once, and each actor A

²Here we ignore a number of details, *e.g.*, actor *ports*, subworkflows “hidden” within so-called composite actors, *etc.*

in W is executed only after all actors A' preceding A (denoted $A' \prec_W A$) in W have *finished* their execution. Note that we make no assumption about whether W is executed sequentially or task-parallel; we only require that any DAG-compatible schedule for W satisfy the partial order \prec_W induced by W . A DAG director can obtain all legal schedules for W (i.e., the relation \prec_W) via a topological sort of W . Finally, note that the DAG model can easily support task and data parallelism, but not pipeline parallelism.

Another model of computation, extending PN, is COMAD (Collection-Oriented Modeling and Design) [MBL06, BMWL07]. In this MoC, actors operate on streams of nested data collections (similar to XML data), and can be configured (via XPath-like *scope expressions* and *signatures*) to “pick up” and operate only on relevant parts of the input stream, injecting results back into the output stream for further downstream processing. This MoC can simplify workflow design and reuse when compared with DAG, SDF, and PN workflows [MBZL09].

13.2.4 Benefits of Scientific Workflows

Scientific workflows are designed to help scientists perform effective computational experiments by providing an environment that simplifies (*in silico*) experimental design, implementation, and documentation. The increasing use of scientific workflow environments and systems is due to a number of advantages these systems can offer over alternative approaches.

- Scientific workflows *automate* repetitive tasks, allowing scientists to focus on the science driving the experiment instead of data and process management. For example, automation of parameter studies—where the same process is performed hundreds to thousands of times with different parameter sets—can often be more easily and efficiently achieved than with conventional programming approaches.
- Scientific workflows explicitly *document* the scientific process being performed, which can lead to better communication, collaboration (e.g., sharing of workflows among scientists), and reproducibility of results.
- Scientific workflow systems can be used to *monitor* workflow execution and *record* the provenance of workflow results. Provenance, in particular, provides a form of documentation that can be used to validate and interpret results produced by (often complex) scientific processes.
- Scientific workflow systems often can *optimize* and then more *efficiently execute* scientific processes, e.g., by exposing and exploiting various forms of parallelism inherent in data-driven scientific processes, as well as by employing other techniques for efficient resource management.
- Workflow environments encourage the *reuse* of knowledge artifacts (actors, workflows, etc.) developed when automating a scientific process, both within and across disciplines.

13.3 Case Study: Fusion Simulation Management

We now present a detailed case study to make the previously discussed notions more concrete. We chose a simulation management workflow as our example because it exhibits a number of challenging issues typically not found in other types of scientific workflows. In our terminology, the workflow is a *resource-oriented, production* workflow. The main scientific computations (the fusion simulation) are performed on a remote supercomputer cluster, while the management workflow can be executed on the scientist’s desktop. The overall computation managed by the workflow is both *data-intensive* and *compute-intensive*; involves pipeline parallelism over a stream of data or reference tokens;³ and is responsible for job management, file transfers, and data archiving. Such workflows have been called “plumbing” workflows due to their focus on explicitly dealing with underlying resources (which the end-user scientist prefers not to deal with).

³a *reference token* is a “logical pointer” to a data object, e.g., a file name

From the scientist’s point of view, the primary task is to observe and analyze the simulation results as soon as possible. To understand this challenge, we first describe briefly the physics problems studied in the simulations.

The Center for Plasma Edge Simulation (CPES) is a DOE SciDAC project [cpe] requiring close collaboration between physicists, applied mathematicians, and computer scientists. Together, these researchers have developed a complete plasma fusion simulation, called XGC1 [KCA⁺06] that runs in a high-performance computing environment. The computational scientists use this simulation to study the behavior of hot plasma in a tokamak-type fusion reactor. The central issue under study is as follows. Within a fusion reactor, if the hot edge of the plasma is allowed to contact the reactor wall in an uncontrolled way, it can sputter the wall material into the plasma, degrade or extinguish the fusion burn, and shorten the wall lifetime to an unacceptable level. Anomalous tokamak plasma transport is thought to be associated with small-scale plasma turbulence. When the heating power to the core plasma is above a certain threshold, a thin plasma layer forms, making the plasma almost free of turbulence; in addition, the central plasma temperature and density rise under these conditions with the added benefit that the fusion probability increases dramatically. However, this layer also triggers large magnetohydrodynamic-type instabilities, which simultaneously destroy the layer (lowering the fusion power in the core) and dump the plasma energy to the material walls. It is currently not fully understood how this layer builds up or how the following crash occurs. The success of next-generation burning plasma experiments is heavily dependent upon solving this problem. Thus, understanding these physical processes is an important area in fusion plasma research (see [CPP⁺08] for more details).

Due to the complexity of running the simulation (*e.g.*, staging input data, monitoring execution status, and managing result data) as well as to the rapid changes and evolution of the simulation code itself, automating the execution of the simulation via a workflow is crucial for both XGC1 developers and scientists wanting to evaluate XGC1 results. Here, we can distinguish three layers of activity: (1) At the highest level, physicists are interested in understanding (and ultimately taming) nuclear fusion in tokamak-type reactors; (2) in addition to performing the actual experiments, sophisticated, large-scale simulations on supercomputers are used to gain insights into the process (this is a goal of the CPES project); and finally (3) a simulation management workflow is used to deal with the challenging issues in running the simulations, and automating the necessary steps as much as possible.

13.3.1 Overview of the Simulation Monitoring Workflow

The XGC1 simulation outputs one-dimensional diagnostic variables in three NetCDF files. It also outputs three-dimensional data written in a custom binary format for efficient I/O performance. The latter data subsequently is converted to standard formats, such as HDF5, for archiving and analysis. Simple plots are generated from the 1-D diagnostic variables, while 2-D visualizations (*i.e.*, cross-section slices of the tokamak) are produced from the converted 3-D data. The NetCDF, HDF5, and all images produced during the simulation are archived.

Figure 13.1 shows a graphical representation of a CPES simulation monitoring workflow implemented using the Kepler scientific workflow system [Kep]. After initial preparation steps (*e.g.*, checking if a simulation restart is requested; logging in to all involved machines; creating directories at the processing sites, *etc.*), two independent, concurrently executing pipelines are started for monitoring the XGC1 simulation. The term “monitoring” is used to indicate that for each output step, plots and images are generated in real-time, and that these can be visualized on a dashboard, enabling the scientist to observe whether the simulation is progressing correctly. The simulation itself executes on a dedicated supercomputer (*primary cluster*) at Oak Ridge National Laboratory (ORNL), while a *secondary cluster* computer at ORNL is used for on-the-fly analysis of the simulation run on the primary cluster.

The first pipeline (shown in the center of Figure 13.1) performs the **NetCDF file processing** portion of the monitoring workflow. This pipeline starts by checking the availability of NetCDF files. As each such file grows (they are extended after every diagnostic period), the workflow performs *split* (taking the most recent data entry), *transfer*, and *merge* operations on recent data to mirror XGC1’s output on the secondary

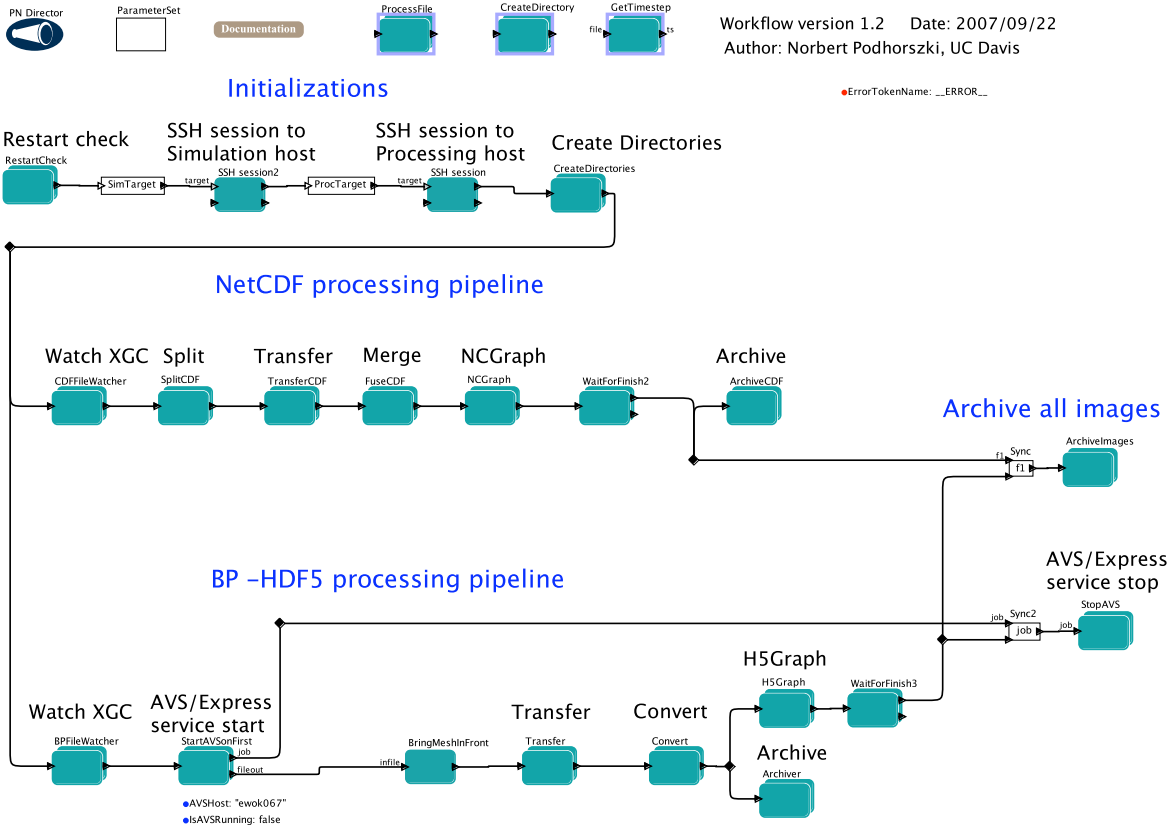


Figure 13.1: CPES Fusion Simulation Monitoring Workflow (in Kepler)

analysis cluster efficiently. Finally, images are generated using *xmgrace*⁴ for all variables in the output for each diagnostic time step and placed into a remote directory where the scientist can browse them via the web-based dashboard application [CPP⁺08] (cf. Section 13.5). The split and merge operations are executed on the login nodes of the primary simulation machine and on the secondary analysis cluster, respectively. To make the plots, however, a job has to be submitted on the secondary cluster for each file in each step. Although one such job is small—lasting only for a couple of seconds—there is almost always one running; this would typically overload the login node of the primary cluster.

The second pipeline (bottom of Figure 13.1) performs the **BP-HDF5 processing**. This pipeline’s role is similar to the NetCDF pipeline, but with the following differences. For each step, XGC1 creates new BP files (a custom **binary packed format**); hence, there are no split and merge steps when transferring them to the secondary processing site. The BP files are converted to HDF5 using an external code, and then images are created for all 2-D slices of the 3-D data stored in those files using an AVS/Express⁵ dataflow network. For this purpose, the pipeline starts AVS/Express as a remote job on the secondary cluster and then makes image-creation requests to it as a (private) service.

This workflow uses a set of fine-grain *job-control* steps provided by Kepler for calling AVS/Express. The workflow waits until the AVS/Express job is started on its execution host, performs the other tasks while the job is running, and stops the job at the end of the processing. The individual steps in Figure 13.1 are workflows themselves (*i.e.*, subworkflows or *composite actors* in Kepler terminology), implementing special tasks. One such subworkflow is the archival step in the HDF5 pipeline, which assembles files into large chunks and stores them in a remote mass-storage system (see [PLK07] for details of this subworkflow).

⁴A tool for **g**raphing, **a**dvanced **c**omputation and **e**xploration of data; see <http://plasma-gate.weizmann.ac.il/Grace/>

⁵http://www.avs.com/software/soft_t/avsxps.html

13.3.2 Issues in Simulation Management

In a *compute-intensive* workflow, jobs typically are executed remotely from the workflow execution engine, and thus the output of one job often must be transferred to another host where subsequent jobs are executed. For efficiency reasons, files are directly transmitted between remote sites, while the workflow engine only “sees” reference tokens to the remote files. The XGC1 case study falls into this category of workflows. It also belongs to a category of *data-intensive* workflows in which the data produced during a supercomputing simulation must be processed on the fly and as quickly as possible. This scenario is typical of most scientific simulations that use supercomputers and produce ever larger amounts of data as the size and speed of the supercomputer clusters continues to increase.

Runtime Decision Support. The typical tasks that a computational scientist performs during and after a simulation run are often tedious to perform manually without automation support. For instance, to maintain high utilization of supercomputing resources, it is essential to be able to detect and halt a divergent simulation. Thus, in most scientific simulations, the status of the computation must be regularly checked to ensure that it is not diverging given for the initial input parameters. However, it can be difficult to check the status of an executing simulation because typically the user has to log in to the primary supercomputer cluster (since applications typically write data to local disks) at regular intervals to analyze *diagnostic values* that reveal errors in the input or simulation code. Moreover, simulations typically write out other (more involved) diagnostic data such as physical variables or derivatives of these variables which must be plotted and analyzed. Although such plots give deeper insight into the current state of the simulation, even more information may be needed for monitoring and runtime decision support, *e.g.*, the ability to visually analyze parts of the dataset written out by the simulation. The latter operation usually cannot be done on the supercomputer’s login node, however, which is one of the reasons for transferring data to another, secondary computer, such as the scientist’s desktop computer or a dedicated visualization computer. Although not described in detail here, the CPES project has automated these various tasks via a separate workflow that greatly reduces the amount of manual work required of users by automatically routing diagnostic information and data, and by displaying the appropriate plots and visualizations on a web-based dashboard.

Data Archiving. Another important task is the archiving of output data. At present, it is sufficient to archive data after the simulation run. In the near future, however, it is anticipated that the largest simulations will create more data in a single run than can fit onto the disk system of the supercomputers. Therefore, files must be transferred to a remote mass storage system on the fly and then removed from the local disk to make space for more data coming from the simulation. There also is a requirement to create “archival chunks” of an intermediate size; for performance reasons, neither individual files nor the complete simulation output (as a single file) can be sent to the archive system. Thus, the automated solution puts files into appropriately sized chunks while taking care of other requirements, *e.g.*, ensuring that all data for one time step goes into the same chunk.⁶ Finally, recording the *data provenance* of all generated data becomes increasingly important as the size and complexity of the output grows. For example, from an automatically generated diagnostic image, a scientist must be able to easily find the output of the simulation corresponding to the visualization. Tools can greatly help with transferring the relevant data to the scientist’s host machine (which could be at a remote site) provided that the above simulation management workflow records the necessary data lineage of all operations.

Pipeline Parallel Processing. An important feature of the Kepler environment is its support for the data-flow process network [Kah74, LP95] model of computation, implemented via the Process Network (PN) director [BLL⁺08]. Using the PN director, all actors are running continuously in separate threads, waiting for input to be processed immediately. Each pipeline in the above workflow is therefore processing

⁶An additional problem arises when data is generated faster than it can be archived. In this case, an additional workflow step can be inserted which uses an auxiliary disk to queue the data, decoupling the slow archival from the fast data generation.

a stream of data items in pipeline-parallel mode. For example, since XGC1 outputs diagnostic data into three NetCDF files at each timestep, plots can be created for one file, while a second file is being used in a merge operation, and while a third file is being transferred. In a typical production run scenario, XGC1 outputs a new timestep every 30 seconds. The time to get one file through the processing pipeline includes the time for recognizing its presence, the transfer time, and the execution time of the plot generation job on the processing cluster. If the workflow performed only one of these steps at a time (*e.g.*, as prescribed by the SDF director), the simulation would generate files faster than they could be processed. Due to the size of the 3-D data in the HDF5 pipeline and the longer transfer time of those files, the situation is similar in this pipeline as well. Finally, the archiving process must obviously work in parallel with the rest of the workflow, since it is a slow process in itself. If the task and pipeline parallelism exhibited by the above workflow is not enough to keep up with the flow of data, one can replicate individual actors on different compute nodes to process multiple data items at the same time. Although the above workflow does not need to do this currently, a more complex production workflow is in use for coupling other codes with the XGC1 predecessor code (such as those described in Section 5), XGC0 [CPP⁺08], where a parameter study has to be executed for each timestep of the simulation, and that study is executed in this parallel mode.

Robustness of Workflows. There are two different but related aspects of robustness that can occur in compute-intensive workflows: what happens if the overall workflow execution fails and stops (*e.g.*, at the workflow engine level), and what happens if an individual task in the workflow fails? For a workflow responsible for starting and monitoring jobs, both eventualities mean that there is a set of successfully executed jobs (whose results should be salvaged) and a set of not yet executed jobs that cannot be started because they either depend on a failed task, or because the workflow (engine) is no longer running and thus cannot start them. After restarting the workflow, a previously failed task can resume.

The tasks comprising the simulation monitoring workflow represent operations carried out on individual data items (files) as the simulation produces data at each timestep. If some operation during a particular timestep fails (*e.g.*, transfer to another host fails, mass storage is down at archiving time, or a statistic cannot be created—all common failures outside the control of the workflow engine), this should not prevent the workflow from invoking the complete pipeline of operations over the data produced during the next timestep. However, because a downstream actor may be affected by the result of an upstream actor, actors should be prepared for such failures. Two possible solutions are to (a) discard from the token stream that token corresponding to the failed operation, or (b) introduce special “failure tokens” to mark jobs that did not succeed. If we discard the token for the failed operation, downstream actors do not receive a bad task request, and therefore no change to the actor is required to handle them. However, the absence of tokens changes the balance between the consumption and production rates of the actors, and this can lead to difficulties in complex workflow design, *e.g.*, if we need to split and merge pipelines. If we replace the token with a failure token, and downstream actors are programmed to simply ignore such failure tokens, the workflow structure remains simpler.⁷

Resuming Workflow Execution Following a Fault. Pipelined (*e.g.* PN) workflows are harder to restart than DAG workflows because the current state of the workflow is not as easy to describe and restore; all actors in the workflow graph may be concurrently executing. While the progress of executing a conventional DAG workflow (Section 13.2.3) can be seen as a single “wavefront” progressing from the beginning of the workflow DAG towards the end, in a pipeline-parallel workflow each task can be invoked repeatedly. If the workflow system does not support full restoration of the workflow and actor state (a nearly impossible task when dealing with workflow components outside the control of the engine), the workflow itself has to include some sort of light-weight checkpoint and restart capability.

In the CPES workflow, the solution is to have the remote execution actor—used for executing all of the actual data processing operations along the pipeline—record all successful operations [PLK07]. When

⁷The first design of CPES workflows was based on approach (a), while for the above reason, an improved design employed the second approach (b). The COMAD model of computation (see Section 13.2.3 and [MBZL09]) natively supports mechanisms to tag data, which is an elegant way to achieve variant (b); it can be used to skip over or even bypass data around actors [ZBML09], or perform other forms of exception handling based on tags.

restarted, *e.g.*, after failure, this actor checks the current tasks to be performed against the set of successful tasks and skips over any that were already executed successfully. In this way, the next actor in the pipeline can immediately start working on it (or skip over it as well). Thus, although the workflow restarts from the very beginning, pushing the initial input tokens back into the pipeline, the actors “fast-forward” to the most recent state prior to the workflow failure, by skipping the tokens corresponding to previously successful tasks. The time spent by the workflow engine in this fast-forward restoration process is negligible compared to the time of actually executing the remote operations.

13.4 Grid Workflows and the Scientific Workflow Life Cycle

The term *Grid workflow* applies to workflows that employ distributed (often wide-area) computational resources (often referred to as “the Grid”). Like other scientific workflows, grid workflows can be seen as high-level specifications of sets of tasks and the dependencies between them that must be satisfied in order to accomplish a specific goal. The specific goal of grid-enabled workflow systems is to reduce the programming effort required of scientists orchestrating a computational science experiment in a wide-area, distributed system. The vast majority of scientists do not use grid systems in their day-to-day practices, largely because of usability barriers. Workflow systems are beginning to address these usability barriers and to make grid computing far more accessible to general science users. In this section, we focus on the first three stages of the life cycle summarized in Section 13.2.1—scientific workflow composition, mapping of workflows onto resources, and workflow execution—and describe how several popular Grid-enabled workflow systems support these different stages. We present a cross-sectional view of the types of Grid workflows that are currently being deployed and compare features provided by Kepler [Kep, LAB⁺06], Pegasus [Peg, DBG⁺04], Taverna [Tav, OAF⁺04, OGA⁺06], Triana [Tri, TSWR03], and Wings [GRD⁺07].⁸

13.4.1 Workflow Design and Composition

Most e-Science workflow systems provide a graphical tool for composing workflows. For example, Kepler and Triana have sophisticated graphical composition tools for building workflow graphs using a graph or block diagram metaphor, where nodes in the workflow graph represent tasks and edges dataflow dependencies or task precedence. The intent of these graphical composition tools is to simplify for scientists the task of describing workflows. Other, *task-based* systems such as Pegasus focus on the mapping and execution capabilities and leave the higher-level composition tasks to other tools.

Task-level workflow systems focus on resource-level functionality and fault-tolerance, while service-level systems generally provide interfaces to certain classes of services for management and composition. One important factor to the adoption of workflow systems by scientists is the availability of workflow tools and services that scientists can build on in order to create their applications. Such service availability forms part of the composition process since it represents the available tools that can be composed within a system.

Pegasus takes a workflow description in a form of a Directed Acyclic Graph in XML format (DAX). The DAX can be generated using a Java API, any scripting language, or using semantic technologies such as Wings [GRD⁺07]. In some scientific applications, users prefer an interface that simply supports metadata queries while hiding the details of how the underlying systems work. In astronomy, for example, users want simply to retrieve images of an area of the sky of interest to them. In such cases Pegasus is usually integrated into a portal environment, and the user is presented with a web form for entering desired metadata attributes. Behind the portal a workflow instance is then generated automatically based on the user’s input, given to Pegasus for mapping, and then passed to DAGMan [Tea02] for execution. Examples of this approach can be seen in the Montage project (an astronomy application) [BGL⁺03, KBB⁺04], the Telescience portal (a neuroscience application) [LSK⁺06], and the Earthworks portal (an earthquake science application) [MPMF⁺06]. In all of these applications, Pegasus and DAGMan are being used to run workflows on national infrastructure such as the TeraGrid.

⁸For a high-level overview and attempt at a classification of current systems see [YB05] and <http://www.extreme.indiana.edu/swf-survey/>; these include references to other scientific workflow systems, such as Askalon, Karajan, and many others.

Kepler provides a graphical user interface for composing and editing workflows using a hierarchical representation of the workflow graph (see the example in Figure 13.1). Dataflow is indicated by channels represented as edges among the nodes of the graph, and each node represents either an atomic task or a composite task (containing a sub-workflow). The user interface provides a semantic-search system across hundreds of different scientific computing components available in the Kepler library. These components cover a wide variety of scientific data processing and modeling activities, such as geospatial data processing, signal processing, statistical algorithms, and data transformations. The semantic search feature [BBJ⁺05, BL05] assists the user in locating components that are relevant to their analysis and modeling tasks. It is also useful when searching the remote Kepler library, allowing users to find components that have been shared by other Kepler users and to share their own components and workflows with others. Kepler workflows can be executed directly from the workflow-composition GUI or saved in an XML representation (MoML) and later passed to Kepler for execution in the absence of the GUI. This feature allows Kepler to be embedded in web portals and other applications.

Taverna provides a GUI-based desktop application that uses semantic annotations associated with services, and employs the use of semantics-enabled helper functions, and uses reasoning techniques to infer service annotations [BEP⁺08]. Over 800 services are described using ontologies [GWG⁺07] expertly annotated by a full time curator used by clients such as Find-O-Matic, its discovery tool, Feta [LAWG05] which is only available as a plug-in from the Taverna Workflow Workbench. The BioCatalogue⁹ project [GSH⁺08] incorporates the experiences of the Taverna Registry and myExperiment (Section 13.6) to build and manage a richly described catalogue of web services in the Life Sciences. The catalogue's services have descriptive content capturing functional capabilities curated by experts and by the community through social collaboration; operational content such as quality of service and popularity is automatically curated by monitoring and use analysis. The BioCatalogue is a free standing component with its own RESTful APIs that can be embedded within and accessed from third party applications. Developers can incorporate new services through simple actions and can load a pre-existing workflow as a service definition within the service palette, which can then be used as a service instance within the current workflow. Taverna also supports the configuration of the appearance of the graphical representation of workflows, so that a workflow can be suppressed to give higher level views, e.g., to remove details such as data translation (or other "shim") services.¹⁰

One of the most powerful aspects of Triana is its graphical user interface. It has evolved in its Java form for over 10 years and contains a number of powerful editing capabilities, wizards for on-the-fly creation of tools and GUI builders for creating user interfaces. Triana editing capabilities include: multi-level grouping for simplifying workflows, cut/copy/paste/undo, ability to edit input/output nodes (to make copies of data and add parameter dependencies, remote controls or plug-ins), zoom functions, various cabling types, optional inputs, type checking and so on. Since Triana came from the gravitational-wave field, the system contains a wide ranging palette of tools (around 400) for the analysis and manipulation of one-dimensional data, which are mostly written in Java (with some in C). Recently, other extensive toolkits have been added for audio analysis, image processing, text editing, for creating retinopathy workflows (*i.e.*, for diabetic retinopathy studies) and even data mining based on configurable web services to aid in the composition process. See [Tay06] for a further discussion and description of such applications.

Wings [GRD⁺07] uses rich semantic descriptions of components and workflow templates expressed in terms of domain ontologies and constraints. Wings has a workflow template editor to compose components and their dataflow. The editor assists the user by enforcing the constraints specified for the workflow components. It also assists the user with data selection, to ensure the data sets selected conform to the requirements of the workflow template. With this information, Wings generates a workflow instance that specifies the computations (but not where they will take place) and the new data products. For all the new data products, it generates metadata attributes by propagating metadata from the input data through the descriptions and constraints specified for each of the components.

⁹<http://biocatalogue.org>

¹⁰Shims align or mediate data that is syntactically or semantically closely related, but not directly compatible [HSL⁺04].

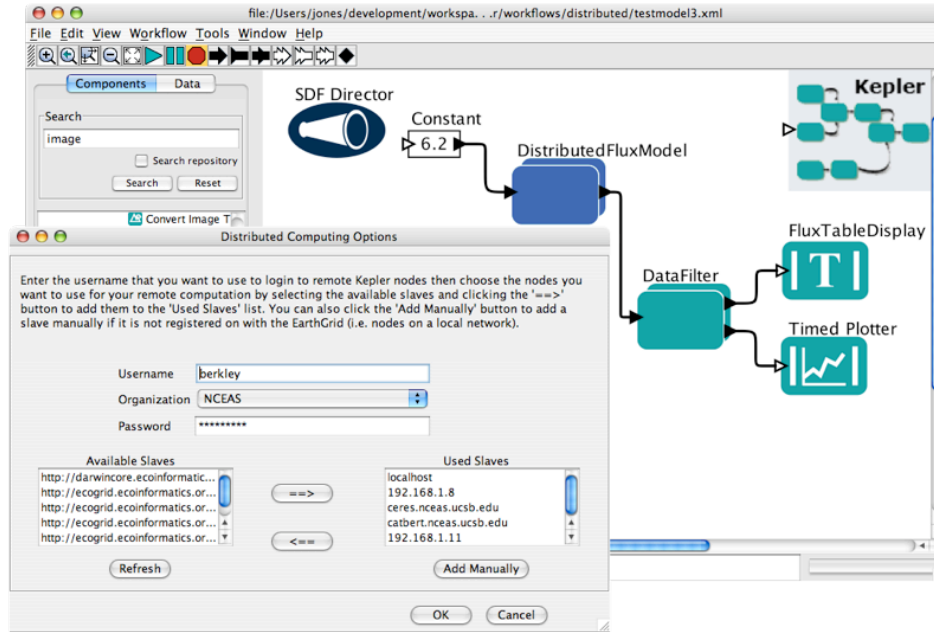


Figure 13.2: Kepler supports execution of workflows on remote peer nodes and remote clusters. Users indicate which portions of a workflow should be remotely executed by grouping them in a distributed composite component (shown in blue in the workflow). The user selects from a list of available remote nodes for execution (see dialog), and Kepler calculates a schedule and stages each data token before execution on one of the set of selected remote nodes.

13.4.2 Mapping Workflows to Resources

It is often the case that at the time the workflow is being designed the target resources are yet to be chosen. *Workflow mapping* refers to the process of generating an executable workflow based on a resource-independent workflow description sometimes called an abstract workflow. In some cases the user performs the mapping directly by selecting the appropriate resources. In other cases, the workflow system performs the mapping.

Depending on the underlying execution model of standalone applications, or individual services, different approaches are taken to the mapping process. In the case of service-based workflows, mapping consists of finding and binding to services appropriate for the execution of a high-level functionality. Service-based workflows also can consider quality of service requirements when performing the mapping. In the case of workflows composed of stand-alone applications, the mapping not only involves finding the necessary resources to execute the computations and performing various optimizations, but may also include modifying the original workflow.

Some systems such as Taverna rely on the user to make the choice of resources or services. In the case of Taverna, the user can provide a set of services which match a particular workflow component, so if errors occur, an alternate service can be automatically invoked. The newer versions of Taverna will include late service binding capabilities.

Kepler, on the other hand, allows the user to specify resource bindings through its distributed computation configuration system. The user designs the workflow in a manner that indicates which components are compute-intensive and should be distributed across remote computational resources. The user then is presented with a dialog listing available compute resources, which can include both other Kepler peers and remote Kepler slaves running on computing clusters (see example in Figure 13.2). The user selects which set should be used for the execution, and the Kepler execution engine then determines a schedule for data

transfer and execution of jobs based on the execution model used in the abstract workflow model. In addition, Kepler can be used to configure and submit jobs to a variety of other grid-based computing systems, including Griddles [KA05], Nimrod [ASGH95], and other systems.

Triana is able to interface to a variety of execution environments using the GAT (Grid Application Toolkit) [TSWR03] for task-based workflows and the GAP (Grid Application Prototype) for service-based workflows. In the case of a service-based workflow, a user can provide the information about the services to invoke (or locate them via a repository). Alternatively, a user can create a workflow and then map part of the workflow to distributed services through the use of one of the internal scripts, *e.g.*, parallel or pipeline. In this mode, Triana distributes workflows by using (or deploying on-the-fly) distributed Triana services that can accept a Triana taskgraph as input. In the case of a task-based workflow, the user can designate portions of the workflow as compute-intensive and Triana will send the tasks to the available resources for execution. It can, for example, use the GAT interface to the Gridlab GRMS broker [GRM05] to perform the resource selection at runtime. Workflows can also be specified using a number of built-in scripts that can be used to map from a simple workflow specification (*e.g.*, specifying a loop) to multiple distributed resources in order to simplify the orchestration process for distributed rendering. Such scripts can map sub-workflows onto available resources by using any of the service-oriented bindings available, *e.g.*, WSRF (Web Services Resource Framework), Web and P2P (peer-to-peer) services using built-in deployment services for each binding.

Workflows specified in DAGMan can be a mixture of concrete and abstract tasks. When DAGMan is interfaced to a Condor task execution system [Tea02], the abstract tasks can be matched dynamically to Condor resources. The matching is done by the Condor matchmaker, which matches the requirements of an abstract task specified in a Condor classAd¹¹ with the resource preferences published in their classAds. We also note that Pegasus uses DAGMan as an execution engine (Figure 13.3). Currently, Pegasus and DAGMan are being integrated into a system, Pegasus-WMS, which provides the user with an end-to-end workflow solution.

Pegasus performs a mapping of the entire workflow, portions of the workflow, or individual tasks onto the available resources. In the simplest case Pegasus chooses the sources of input data (assuming that it is replicated in the environment) and the locations where the tasks are to be executed. Pegasus provides an interface to a user-defined scheduler and includes a number of scheduling algorithms. As with many scheduling algorithms, the quality of the schedule depends on the quality of the information both of the execution time of the tasks and data access as well as the information about the resources. In addition to the basic mapping algorithm, Pegasus can perform the following optimizations: tasks clustering, data reuse, data cleanup, and partitioning. Before the workflow mapping, the original workflow can be partitioned into any number of sub-workflows. Each sub-workflow is then mapped by Pegasus. The order of the mapping is dictated by the dependencies between the sub-workflows. In some cases the sub-workflows can be mapped and executed in parallel. The granularity of the partitioning is dictated by how fast the target execution resources are changing. In a dynamic environment, partitions with small numbers of tasks are preferable, so that only a small number of tasks are bound to resources at any one time. On the other hand, in a dedicated execution environment, the entire workflow can be mapped at once. Pegasus can also reuse intermediate data products if they are available and thus possibly reduce the amount of computation that needs to be performed. Pegasus also adds data cleanup nodes to the workflow, which remove the data at the execution sites when they are no longer needed. This often results in a reduce workflow data footprint. Finally, Pegasus can also perform task clustering, treating a set of tasks as one for the purpose of scheduling to a remote location. The execution of the cluster at the remote site can be sequential or parallel (if applicable). Task clustering can be beneficial for fine computational granularity workflows. Pegasus has also been used in conjunction with resource provisioning techniques to improve the overall workflow performance [SSV⁺08].

¹¹Classified Advertisement

13.4.3 Workflow Execution

In this section, we contrast approaches to workflow execution in Pegasus, Triana, and Kepler. Pegasus can map workflows onto a variety of target resources such as those managed by PBS (Portable Batch System) [DBG⁺04], LSF (Load Sharing Facility), Condor [WNB07], and individual machines. Authentication to remote resources is done via GSI (Grid Security Infrastructure) [SPG05]. During workflow execution, Pegasus captures provenance information about the executed tasks. Provenance includes a variety of information including the hosts where tasks executed, task run times, environment variables, *etc.* Pegasus uses the DAGMan workflow engine for execution (Figure 13.3). DAGMan interfaces in turn to a local Condor queue managed by a scheduler daemon. DAGMan uses the scheduler’s API and logs to submit, query, and manipulate jobs, and does not directly interact with jobs. DAGMan can also use Condor’s grid abilities (Condor-G) to submit jobs to many other batch and grid systems. DAGMan reads the logs of the underlying batch system to follow the status of submitted jobs rather than invoking interactive tools or service APIs. By relying on file-based I/O, DAGMan’s implementation can be simpler, more scalable and reliable across many platforms, and therefore more robust. For example, if DAGMan has crashed while the underlying batch system continues to run jobs, DAGMan can recover its state upon restart (by reading logs provided by the batch system) without losing information about the executing workflow. DAGMan workflow management includes not only job submission and monitoring but also job preparation, cleanup, throttling, retry, and other actions necessary to ensure successful workflow execution. DAGMan attempts to overcome or work around as many execution errors as possible, and in the face of errors it cannot overcome, it provides a *Rescue DAG*¹², and allows the user to resolve the problem manually and then resume the workflow from the point where it last left off. This can be thought of as a “checkpointing” of the workflow, just as some batch systems provide checkpointing of jobs.

Triana supports job level execution through GAT integration, which can make use of job execution components such as GRMS [GRM05], GRAM [GRA08] or Condor [Tea02] for the actual job submission. It also supports service-level execution through the GAP bindings to Web, WSRF and P2P services. During execution, Triana will identify failures for components and provide feedback to the user if a component fails. Triana does not contain fail-safe mechanisms within the system for, *e.g.* retrying a service, however.

As discussed in Section 13.2.3, execution of a Kepler workflow is managed through an independent component called a *director* which is in charge of workflow scheduling and execution¹³. A director in Kepler encapsulates a *model of computation* (MoC) and a scheduling algorithm, which allows the same workflow to be executed in different ways depending on which workflow director/MoC is used. Kepler ships with some common MoCs, such as SDF, PN, and DDF (see Section 13.2.3 for more details).

When comparing different workflow execution strategies and approaches, it seems that a “one size fits all” solution is hard, if not impossible, to achieve.¹⁴ In Pegasus, for example, workflow execution is primarily via Condor/DAGMan, a very mature and reliable platform (with some built-in fault tolerance) for job-oriented scientific workflows that can be expressed as acyclic task dependency graphs. However, there are applications such as scientific workflows over remote data streams (see, *e.g.*, [RG08]) which require other models of computation, *e.g.*, to express loops and streaming pipeline parallelism. Kepler inherits from Ptolemy [BLL⁺08] a number of such advanced models of computation that can even be combined in different ways [GBA⁺07]. Kepler also adds new models, *e.g.*, a data-oriented model of computation called COMAD that results in workflows which are easier to build and understand [BMW07, MBZL09]. Triana, on the other hand, is a service-oriented system, supporting a wide variety of different Grid and service-oriented execution environments. In the end, user requirements and application constraints have to be taken into account when deciding which execution model or system to choose.

¹²http://www.cs.wisc.edu/condor/manual/v7.0/2_10DAGMan_Applications.html

¹³or orchestration and choreography in web service parlance

¹⁴The different approaches do not necessarily exclude each other however: *e.g.*, [MDM⁺07] reports on experiences in combining a Kepler frontend with a Pegasus backend, combining features of both systems (but also limiting each system’s more general capabilities). While end users typically avoid “system mashups”, some interesting insights into the different approaches and capabilities can still be gained by such interoperability experiments.

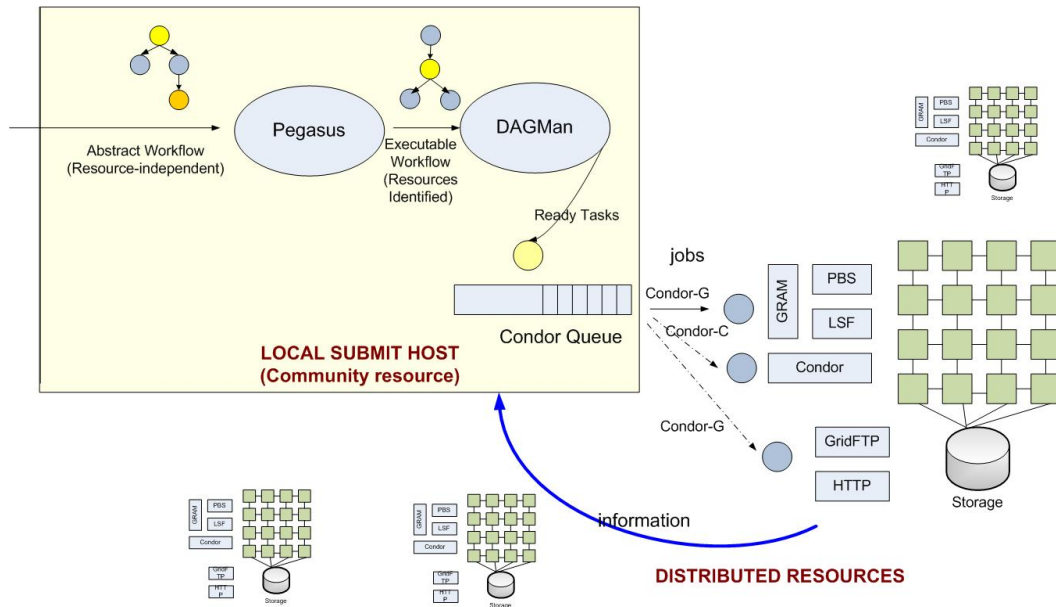


Figure 13.3: An overview of the Pegasus/DAGMan workflow management system. The Pegasus Mapper takes a resource-independent workflow description and maps it onto the available cyberinfrastructure resources. The resulting executable workflow is managed by DAGMan which uses Condor/Condor-G to send individual workflow tasks from the submit host (a user or community resource) to the cyberinfrastructure: a local machine, a campus cluster, or a Grid.

13.5 Workflow Provenance and Execution Monitoring

The absence of detailed provenance information presents difficulties for scientists wishing to share and reproduce results, validate the results of others, and re-use the knowledge involved in analyzing and generating scientific data. In addition, the lack of provenance information may also limit the longevity of data. For example, without sufficient information describing how data products were generated, the value and use of this data may be greatly diminished. Thus, many current scientific workflow systems provide mechanisms for recording the provenance of workflows and their associated data products. This provenance information can be used to answer a number of basic questions posed by scientists related to data such as: Who created this data product and when? What were the processes used in creating this data product? Which data products were derived from this data product? And were these two data products derived from the same raw (input) data?

The software infrastructure required to accurately and efficiently answer these questions is far from trivial [DF08, FKSS08], especially in light of the need to make provenance-related software tools usable by domain scientists, who do not necessarily have programming expertise. For instance, while it may be possible to use session logs generated by software tools to capture provenance information [LPA⁺08], these logs may not be represented in a format that can be easily queried, and further, they may require sophisticated programming techniques to uncover the information needed to answer the above questions related to data lineage.

Similar to a workflow specification, the provenance of a workflow run is often represented using graph structures in which nodes represent processes and data products, and edges capture the flow of data between processes [DF08, BML⁺06]. Some approaches support additional graph representations, *e.g.*, by recording explicit process and data dependencies [MBZG08, BMR⁺08]. A complete description of provenance models and capture mechanisms is beyond the scope of this chapter (see, *e.g.*, [SPG05, DF08]). Instead, we concentrate on describing one particular scheme of storing provenance information that combines

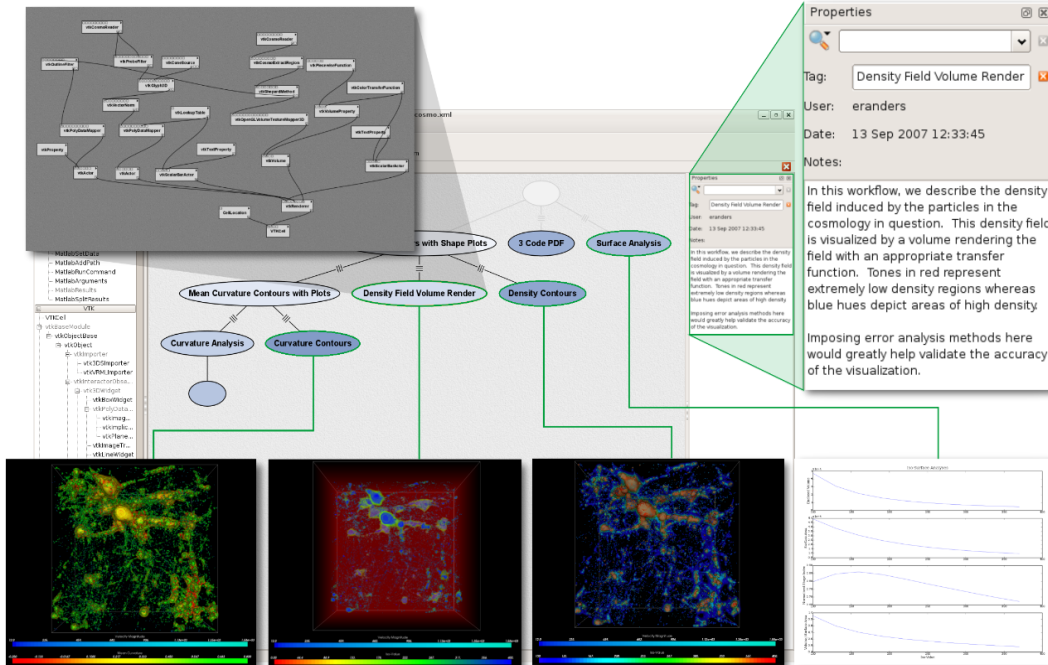


Figure 13.4: In VisTrails, workflow evolution provenance is displayed as a history tree, each node representing a workflow that generates a visualization. This tree allows a user to return to previous workflow versions and to be reminded of the actions that led to a particular result. Additional meta-data stored with each version includes free-text notes and user who created it.

features from Kepler [Kep], VisTrails [Vis], and Pegasus [SSV⁺08]. We first describe the different types of provenance information considered by these approaches, and then discuss the current implementation of the provenance framework used by the SDM Center [SAC⁺07].

Types of Provenance Information. Provenance information related to scientific workflow systems is sometimes divided into three distinct types, or layers [SKS⁺08a]: workflow *description*, workflow *evolution*, and workflow *execution*. The workflow description layer consists of the specifications of individual workflows. The workflow evolution layer captures the relationships among a series of workflow specifications that are created in the course of defining an exploratory analysis. Finally, the execution layer stores runtime information about the execution of a workflow. This information may include, *e.g.*, the day and time the workflow was run, the execution time of each workflow step, the data provided to and generated by each step, a description of the workflow deployment environment, and so on. There are many ways to store information in each layer. For example, in VisTrails, a “change-based” model is used to represent both the evolution and workflow layers [FSC⁺06], run-time information is captured by the workflow execution engine and stored in a relational database. The three layers are related by the overall provenance storage infrastructure.

The separation of provenance information into distinct layers can lead to a more normalized representation that avoids storing portions of each layer redundantly. For instance, this is in contrast to provenance approaches that store information about the workflow specification within the execution log, where a module name, the module parameters, and the parameter values are saved for each invocation of a given module.

Separating provenance information into distinct layers can also help provenance frameworks become more extensible, *e.g.*, by allowing layers to be replaced with new representation approaches or by allowing entirely new layers to be added [SKS⁺08b].

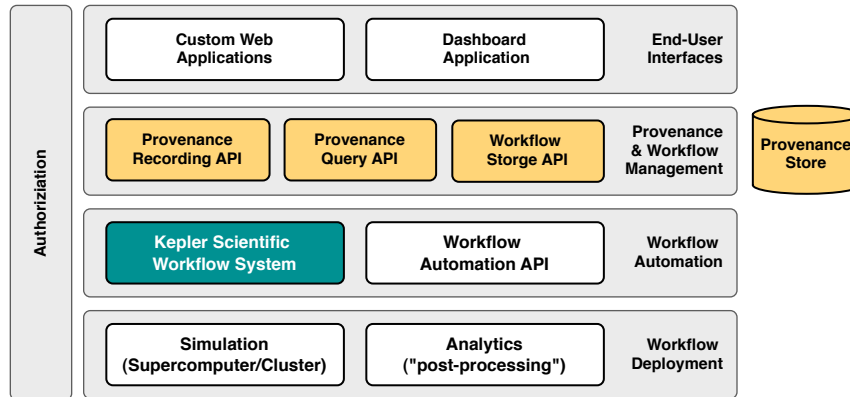


Figure 13.5: Overview of the SDM provenance framework

The VisTrails workflow evolution approach captures changes to workflow specifications and displays these changes using a history tree called a *visualization trail*, or *vistrail* for short [FSC⁺06]. As a workflow developer makes modifications to a workflow, the VisTrails system records each change. Instead of storing a set of related workflows, the change-based model stores the operations, or actions, that are applied to the workflows (*e.g.*, the addition or deletion of a module, the addition or deletion of a connection between modules, and the modification of a parameter value). This representation (similar, *e.g.*, to source-code control systems such as Subversion) uses substantially less space than the alternative of explicitly storing each version of a workflow. In addition, VisTrails provides an intuitive interface that can help users to both understand and interact with the version history of a workflow design [FSC⁺06]. This tree-based view (see Figure 13.4) allows a user to return to a previous version, undo changes, compare different workflows, and determine the actions that led to a particular result.

In addition, query languages and user interfaces that can allow users to explore the provenance of workflow runs are also important [BMR⁺08, BMWL07, BEKM06, FSC⁺06, SVK⁺07]. For example, the ability to query both the specification and provenance of computational tasks enables users to better understand the tasks and their results. In this way, users can identify workflows that are suitable for and can be re-used for a given task; identify workflow instances that have been found to contain anomalies; and compare and understand the differences between workflows [FSC⁺06, SVK⁺07, BMWL07]. Many existing workflow systems support query and visualization of provenance information associated with the workflow definition and execution layers (*e.g.*, see [Mor08]).

13.5.1 Example Implementation of a Provenance Framework

Figure 13.5 shows the high-level architecture for the provenance framework employed within the SDM Center. The architecture has been implemented with the goal of supporting scientists as they run large-scale simulations [BCK⁺07, ACC⁺07]. At the heart of this framework is the *provenance store*, which includes one or more databases providing physical storage, as well as various application programming interfaces (APIs) to access and manage provenance information.

The provenance store within the SDM framework captures the following types of information:

- *Process monitoring* information, which includes data transfer rates, file sizes moved, time taken for actor execution and check-pointing, memory usage, process states (initiated, executing, waiting, terminated, aborted), *etc.* This information is useful, *e.g.*, to benchmark workflow execution and detect bottlenecks.
- *Data provenance and lineage* information, which links an actor's data output to (i) the specific actor invocation that created the data, (ii) the relevant data inputs, and (iii) the parameters at the time of

invocation. Data provenance allows a scientist to interpret and “debug” analysis results, *e.g.*, by stepping through time in the processing history, thus tracing back (intermediate) results to the inputs that created them. Other uses are increased workflow robustness (cf. Section 13.3.2) and improved efficiency upon re-running only the affected parts after modifying some workflow parameters or inputs.

- *Workflow evolution*, which captures changes over time to the workflow description (including parameter changes). It is particularly important to track workflow evolution as part of exploratory workflow design, when there are many cycles of workflow modifications and workflow runs.
- *System environment* information, which captures data about the system that executes the workflow, and its environment, *e.g.*, the machines, operating systems, compiler versions, job queues, *etc.* that were used. It is important to capture such information since, in practice, results will depend on the system environment that a workflow executes in.

Kepler has been extended to record and store various forms of provenance information [ABJF06, BMR⁺08, CA08]. Depending on the settings of the Kepler provenance recorder, data may be recorded for all actors in the workflow, or some subset, *e.g.*, only top-level composites. The recording API also supports recording of information from components external to Kepler (*e.g.*, from Python or shell scripts that are invoked by an actor).

A provenance query API provides a (read-only) mechanism to retrieve provenance information from the provenance store, *e.g.*, a call-back mechanism to notify applications (such as a web-based workflow monitoring dashboard) during workflow execution. In addition to providing current workflow status, authorized users and applications can query the provenance store about past executions via an SQL interface, thus supporting provenance analytics.

While provenance information is typically used “post-mortem”, *i.e.*, after a workflow is run to interpret, validate, or debug results, it can also support runtime execution monitoring [KBB⁺05, BCK⁺07]. The SDM dashboard application supports run-time execution monitoring using the architecture of Figure 13.5. The dashboard is illustrated in Figure 13.6, which shows the on-the-fly visualizations generated by the monitoring workflow described in Section 13.3. Dashboards generally display condensed information about the status of workflow processes, data, the execution environment, *etc.* In addition to providing a high-level overview, such dashboards may also offer a way to navigate into details of runtime progress and provenance trace information, and to show trends in the output data or execution performance.

Other scientific workflow systems have similar capabilities to those described above. For example, Pegasus has been integrated with the PASOA provenance system [KDG⁺08]. Within Triana, provenance information can include the components executed, their parameters, and the data sets that pass through during execution. A data provenance system for Taverna is described in [MBZG08].

13.6 Workflow Sharing and myExperiment

Understanding the whole lifecycle of workflow design, prototyping, production, management, publication and discovery is fundamental to developing systems that support the scientist’s work and not just the workflow’s execution. Supporting that lifecycle can be the factor that means a workflow approach is adopted or not. Workflow descriptions are not simply digital data objects like many other assets of e-Science, but rather they capture pieces of the scientific process: they are valuable knowledge assets in their own right, capturing valuable know-how that is otherwise often tacit. We can conceive of packages of workflows for certain topics, and of workflow “pattern books”, *i.e.*, new structures above the level of the individual workflow. Workflows themselves can be the subject of peer review, and can support reproducibility in the scholarly knowledge cycle. We can view them as commodities, as valuable first-class assets in their own right, to be pooled and shared, traded and reused, within communities and across communities. This perspective of the interacting data, services, workflows and their metadata within a scientific environment is a *workflow ecosystem*. Understanding and enabling this ecosystem is the key to unlocking the broader scientific potential of workflow systems.

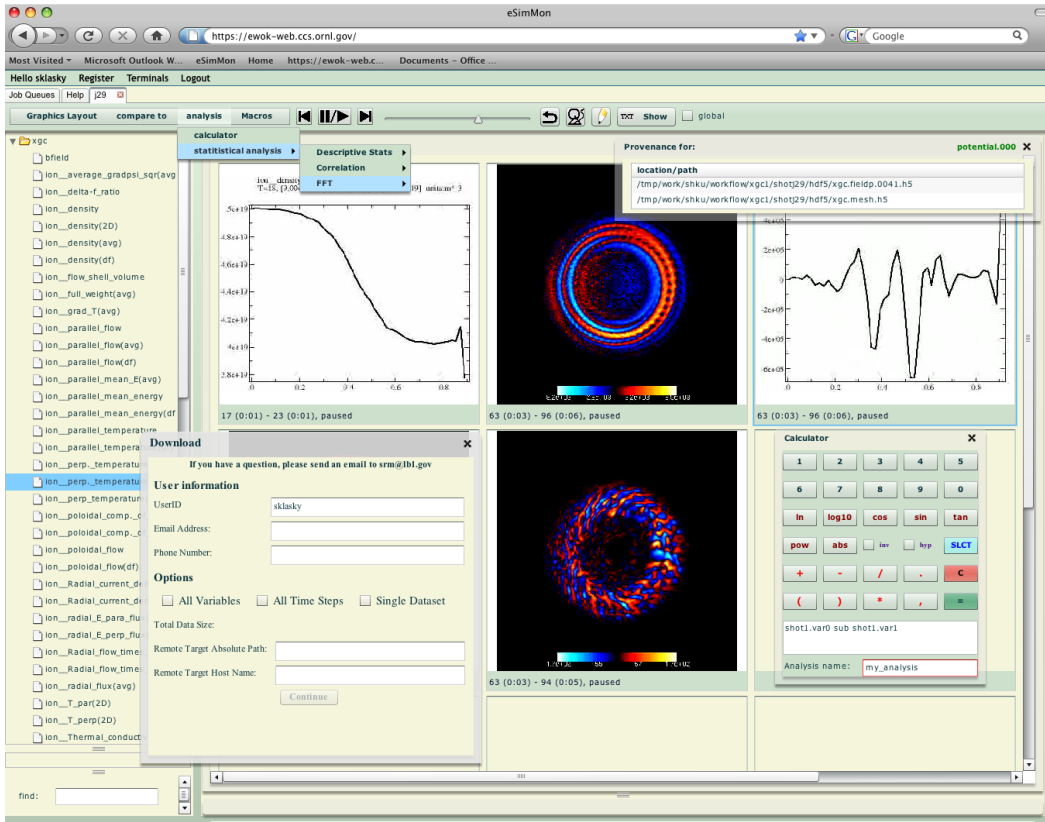


Figure 13.6: SDM dashboard with on-the-fly visualization

13.6.1 Workflow Reuse

Workflow reuse is effective at multiple levels: the scientist reuses a workflow with different parameters and data, and may modify the workflow, as part of the routine of their daily scientific work; workflows can be shared with other scientists conducting similar work, so they provide a means of codifying, sharing and thus spreading the workflow designer's practice; and workflows, workflow fragments and workflow patterns can be reused to support science outside their initial application.

The latter point illustrates the tremendous potential for new scientific advance. An example of this is a workflow used to help identify genes involved in tolerance to Trypanosomiasis in east African cattle [FHW+07]. The workflow was initially successful because it enabled data to be processed systematically without a need for manual triage. This same workflow was then reused over a new dataset to identify the biological pathways implicated in the ability for mice to expel the *Trichuris muris* parasite (a parasite model of the human parasite *Trichuris trichuria*). This reuse was made easier by the explicit, high-level nature of the workflow that describes the analytical protocol.

Workflows bring challenges too. Realistic workflows require skill to produce and therefore they can be difficult and expensive to develop. Consequently, workflow developers need development assistance, and prefer not to start from scratch. This is another incentive for reusing workflows. Unfortunately it is easy for the reuse of a workflow to be confined to the project in which it was conceived. In the Trypanosomiasis example, the barrier to reuse was how the knowledge about the workflow could be spread to the scientists with the potential need. In this case it was word of mouth within one institution; this barrier needs to be overcome.

Workflow management systems already provide basic sharing mechanisms, through repository stores for workflows developed as part of projects or communities. For example, the Kepler actor repository is an

LDAP-based directory for the remote storage, query and retrieval of actors (processes) and other workflow components. Similarly, the Southern California Earthquake Center (SCEC) uses component and workflow libraries annotated with ontologies [MCD⁺05]. These follow the tradition of cataloguing scripting libraries and codes. InforSense’s online Customer Hub and workflow library¹⁵ allow users to share best practices and leverage community knowledge potentially across projects.

13.6.2 Social Sharing

The myExperiment project¹⁶ is taking a more social approach, recognizing the use of workflows by a community of scientists [RGS09]. This acknowledges that the lifecycle of the workflows is coupled with the process of science—that the human system of workflow use is coupled to the digital system of workflows. More workflows imply more users and more enactments, which in turn provide scientists with more samples to assist in selecting workflows, to identify best practices, and to learn and build a reputation by sharing workflows within the community.

From the scientist’s perspective there are many factors guiding reuse of a workflow, including: descriptions of its function and purpose; documentation about the services with which it has been used, with example input and output data, and design explanations; provenance, including its version history and origins; reputation and use within the community; ownership and permissions constraints; quality, whether it is reviewed and still works; and dependencies on other workflows, components and data types. Workflows also enable us to record the provenance of the data resulting from their enactment, and logs of service invocations from workflow runs can inform later decisions about service use. By binding workflows with this kind of information, a basis is provided for workflows to be trusted, interpreted unambiguously, and reused accurately.

The community perspective brings ‘network effects’. By mining the sharing behavior between users within a community we can provide recommendations for use. By using the structure and interactions between users and workflow tools we can identify what is considered to be of greater value to users. Provenance information helps track down workflows through their use in content syndication and aggregation. By sharing or publishing a workflow, with the appropriate attribution, a scientist can allow their work to be reused with the concomitant spread of their scientific reputation but even if scientists do not contribute workflows directly, their usage of workflows within the community still adds value to the body of knowledge about those workflows.

13.6.3 Realizing myExperiment

The rise of harnessing the collective intelligence of the Web has dramatically reminded us that it is people who generate and share knowledge and resources, and people who create network effects in communities. Blogs and wikis, shared tagging services, instant messaging, social networks and semantic descriptions of data relationships are flourishing. Within the scientific community we have many examples, such as OpenWetWare, Connotea, PLoS on Facebook, *etc.*¹⁷

myExperiment is a virtual research environment to support scientists using workflows by adopting a “Web 2.0 approach”. The myExperiment software provides services and a user interface (a social web site) to address the requirements of the social sharing of workflows. It aims to be: a gossip shop to share and discuss workflows and their related scientific objects, regardless of the workflow system; a bazaar for sharing, reusing and repurposing workflows; a gateway to other established environments, for example: depositing into data repositories and journals; and a platform to launch workflows, whatever their system.

In comparison with existing workflow repositories, myExperiment goes the next step: it aims to cross project, community and product boundaries; it emphasizes social networking around the workflows; it gateways to other environments; and it forms the foundation of a personal or laboratory workbench. It also transcends individual workflow systems, envisaging a multiworkflow environment in which scientists will

¹⁵http://www.inforsense.com/pdfs/InforSense_WorkflowLibrary_DataSheet.pdf

¹⁶<http://www.myexperiment.org/>

¹⁷see corresponding .org web sites and facebook.com

use whatever workflow is appropriate for their applications—finding workflows and experiments that they can run across multiple systems.

The design of the myExperiment software is completely user-centric. In order to bootstrap the system, both in terms of content and community, the initial user community comprised users of one particular scientific workflow management system—the Taverna workbench. Developed by the myGrid project [SRG03], Taverna is used extensively across a range of Life Science problems: gene and protein annotation; proteomics, phylogeny and phenotypical studies; microarray data analysis and medical image analysis; high throughput screening of chemical compounds and clinical statistical analysis. Importantly, Taverna has been designed to operate in the open wild world of bioinformatics. Rather than large scale, closed collaborations which own resources, Taverna is used to enable individual scientists to access the many open resources available on the Web. Consequently it has a distributed and decoupled community of users who obtain immediate benefit from sharing workflows through myExperiment.

Released in November 2007, myExperiment was supporting 500 users within 10 weeks and now provides a unique public collection of several hundred workflows from multiple workflow systems.

myExperiment has been designed and built following the mores of Web 2.0 and a set of principles for designing software for adoption by scientists which were established through the Taverna development [RG09].

It is a web based application built on the Ruby on Rails platform, and is not just a single site, like Facebook, YouTube etc, but rather a software package that can be installed independently and separately in a laboratory, supporting the exchange of content between other Web applications and different installations of myExperiment. It reuses other services as far as possible, and it provides simple APIs so that others can make use of it—to make it easy to bring myExperiment functionality into the scientists’ existing environment rather than obliging them to come to myExperiment.

Although initially focused on sharing workflows, myExperiment deals not in workflows or scripts *per se* but in scientific objects—this allows sharing of documents, presentations, service descriptions, notes, ontologies, plans and so forth. More generally, myExperiment can be used to glue together heterogeneous collections like distributed experimental data or, for example, packages of workflows—these collections are described as packs or Research Objects. Hence, rather than a workflow repository, myExperiment can be seen as an aggregator and registry of scientific objects—as its name suggests, it deals in experiments.

13.7 Conclusions and Future Work

Scientific workflows are increasingly being adopted across many natural science and engineering disciplines, spanning all conceivable dimensions and scales, from particle physics and computational chemistry simulations, to bioinformatics analyses, medicine, environmental sciences, engineering, geology, phylogeny, all the way to astronomy and cosmology, to name a few. Not surprisingly, with these rather different domains come different requirements for scientific workflow systems. While some scientific workflows can be conveniently executed on a scientist’s laptop or desktop computer, others require significant computational resources, such as compute clusters, possibly distributed over a local or wide area network. In this chapter, we have given an overview of common features of scientific workflows, described the phases of the scientific workflow life-cycle, and provided some background on the different computational models (and other differences) of scientific workflow systems. A detailed case study from plasma fusion simulation was used to take a closer look at the challenges when managing simulation workflows. We also provided an overview of some of the different approaches taken for scientific workflow systems, focusing on workflow composition, resource mapping, and execution. Furthermore, we described approaches for runtime monitoring and provenance management in scientific workflows, and finally discussed workflow sharing and reuse using a “Web 2.0 approach”.

The area of scientific workflows is dynamic and growing, as evidenced by many workshops, conferences, and special issues of journals, devoted to the topic (*e.g.*, see [LG05, FG06, TDGS07, GDE⁺07]). Numerous challenges of scientific workflows remain and require future research and development. For example, findings from an NSF-sponsored workshop on scientific workflow challenges are reported in

[GDE⁺07], where they are grouped into *application requirements* (e.g., supporting collaborations, reproducibility of scientific analyses, and flexible system environments), *sharing workflow descriptions* (e.g., how to represent and share different levels of workflow abstractions), *dynamic workflows* (how to support the exploratory and dynamic nature of scientific analyses), and *system-level workflow management* (e.g., how to scale workflows and how to deal with infrastructure constraints). In [DC08], data management challenges of data-intensive workflows are presented, and organized according to the data life-cycle in a workflow. For example, during workflow creation, effective means are needed to discover data and software tools, and to capture workflow evolution [FSC⁺06]. Similarly, during workflow planning and execution, there are numerous challenges, e.g., how to efficiently and reliably transfer large amounts of data, or how to deal with distributed, heterogeneous system environments.

Traditionally, in computer science, a core theme is optimization of program runtime and memory usage by developing time- and space-efficient algorithms for the problems at hand. In many application areas, including scientific workflows, “human cycles” are a sometimes neglected resource, which can and should be optimized as well. For example, the use of data and workflow provenance information can be used for traditional purposes (such as optimizing system performance or improving fault-tolerance [CA08]), but also to enhance the scientist’s insights when trying to understand or debug scientific workflow results [DBE⁺07]. Similarly, approaches are needed to facilitate modeling and design of scientific workflows that are easy to use. For example, [MBZL09] lists the following *user-oriented requirements* and provides initial steps towards addressing them: *well-formedness* (facilitate the design of well-formed and valid workflows), *clarity* (facilitate the creation of self-explanatory workflows), *predictability* (make it easy to see what a workflow will do without running it), *recordability* (make it easy to see what a workflow actually did do when it ran) and *reportability* (make it easy to see if a workflow result makes sense scientifically). Clearly, the last two requirements are related to capturing and managing provenance information, a recurring theme in current scientific workflow research. Other research issues mentioned in [MBZL09] are *reusability* (make it easy to design new workflows from existing ones), and *data modeling* (provide first-class support for modeling scientific data), in addition to the already mentioned *optimization* issues (the system should take responsibility for optimizing performance).

Acknowledgements. Work on Kepler is partially supported by the NSF (under grants IIS-0630033, OCI-0722079, IIS-0612326, DBI-0533368) and the DOE (DE-FC02-ER25809 and DE-FC02-07-ER25811); work on VisTrails is partially supported by the NSF (under grants IIS-0844546, IIS-0751152, IIS-0746500, IIS-0513692, CCF-0401498, EIA-0323604, CNS-0514485, IIS-0534628, CNS-0528201, OISE-0405402, IIS-0905385), the DOE SciDAC2 program (VACET), and IBM Faculty Awards (2005, 2006, 2007, and 2008). Ewa Deelman’s work was funded by the NSF under Cooperative Agreement OCI-0438712 and grant #CCF-0725332. We would like to thank Pierre Moualem, Meiyappan Nagappan, and Ustun Yildiz for their support.

Bibliography

- [ABJF06] I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance Collection Support in the Kepler Scientific Workflow System. In L. Moreau and I. T. Foster, editors, *Intl. Provenance and Annotation Workshop (IPAW)*, volume 4145 of *Lecture Notes in Computer Science*, pp. 118–132. Springer, 2006.
- [ACC⁺07] I. Altintas, G. Chin, D. Crawl, T. Critchlow, D. Koop, J. Ligon, B. Ludäscher, P. Moullem, M. Nagappan, N. Podhorszki, C. Silva, and M. Vouk. Provenance in Kepler-based Scientific Workflow Systems. In *Microsoft eScience Workshop*, page 82, 2007. poster.
- [And08] C. Anderson. The End of Theory: The Data Deluge Makes the Scientific Method Obsolete. *WIRED Magazine*, June 2008.
- [ASGH95] D. Abramson, R. Sasic, J. Giddy, and B. Hall. Nimrod: A tool for performing parametrised simulations using distributed workstations. In *4th IEEE Symposium on High Performance Distributed Computing*, pp. 112–121, 1995.
- [BBJ⁺05] C. Berkley, S. Bowers, M. Jones, B. Ludäscher, M. Schildhauer, and J. Tao. Incorporating Semantics in Scientific Workflow Authoring. In J. Frew, editor, *17th Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, pp. 75–78, 2005.
- [BCK⁺07] R. Barreto, T. Critchlow, A. Khan, S. Klasky, L. Kora, J. Ligon, P. Moullem, M. Nagappan, N. Podhorszki, and M. Vouk. Managing and Monitoring Scientific Workflows through Dashboards. In *Microsoft eScience Workshop*, page 108, Chapell Hill, NC, 2007.
- [BEKM06] C. Beerli, A. Eyal, S. Kamenkovich, and T. Milo. Querying Business Processes. In *VLDB*, pp. 343–354, 2006.
- [BEP⁺08] K. Belhajjame, S. M. Embury, N. W. Paton, R. Stevens, and C. A. Goble. Automatic annotation of Web services based on workflow definitions. *ACM Trans. Web*, 2(2):1–34, 2008.
- [BGL⁺03] G. B. Berriman, J. C. Good, A. C. Laity, A. Bergou, J. Jacob, D. S. Katz, E. Deelman, C. Kesselman, G. Singh, M. Su, and R. Williams. Montage: A Grid-Enabled Image Mosaic Service for NVO. *Astronomical Data Analysis Software and Systems, ADASS*, 13, 2003.
- [BL05] S. Bowers and B. Ludäscher. Actor-Oriented Design of Scientific Workflows. In *24st Intl. Conference on Conceptual Modeling (ER)*, pp. 369–384, Klagenfurt, Austria, October 2005. Springer.
- [BLL⁺08] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. Heterogeneous Concurrent Modeling and Design in Java (Volume 3: Ptolemy II Domains). Technical Report No. UCB/EECS-2008-37, April 2008. <http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-37.pdf>.
- [BML⁺06] S. Bowers, T. M. McPhillips, B. Ludäscher, S. Cohen, and S. B. Davidson. A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows. In L. Moreau and I. T. Foster, editors, *Intl. Provenance and Annotation Workshop (IPAW)*, volume 4145 of *Lecture Notes in Computer Science*, pp. 133–147. Springer, 2006.

- [BMR⁺08] S. Bowers, T. McPhillips, S. Riddle, M. Anand, and B. Ludäscher. Kepler/pPOD: Scientific Workflow and Provenance Support for Assembling the Tree of Life. In *Intl. Provenance and Annotation Workshop (IPAW)*, 2008.
- [BMWL07] S. Bowers, T. McPhillips, M. Wu, and B. Ludäscher. Project Histories: Managing Data Provenance Across Collection-Oriented Scientific Workflow Runs. In *4th Intl. Workshop on Data Integration in the Life Sciences (DILS)*, University of Pennsylvania, June 2007.
- [CA08] D. Crawl and I. Altintas. A Provenance-Based Fault Tolerance Mechanism for Scientific Workflows. In *Intl. Provenance and Annotation Workshop (IPAW)*, 2008.
- [cpe] U.S. Department of Energy, The Center for Plasma Edge Simulation Project (CPES). <http://www.cims.nyu.edu/cpes/>.
- [CPP⁺08] J. Cummings, A. Pankin, N. Podhorszki, G. Park, S. Ku, R. Barreto, S. Klasky, C. Chang, H. Strauss, L. Sugiyama, P. Snyder, D. Pearlstein, B. Ludäscher, G. Bateman, and A. Kritz. Plasma Edge Kinetic-MHD Modeling in Tokamaks Using Kepler Workflow for Code Coupling, Data Management and Visualization. *Communications in Computational Physics*, 4(3):675–702, 2008.
- [DBE⁺07] S. B. Davidson, S. C. Boulakia, A. Eyal, B. Ludäscher, T. M. McPhillips, S. Bowers, M. K. Anand, and J. Freire. Provenance in Scientific Workflow Systems. *IEEE Data Eng. Bull.*, 30(4):44–50, 2007.
- [DBG⁺03] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing*, 1(1):25–39, 2003.
- [DBG⁺04] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny. Pegasus: Mapping Scientific Workflows onto the Grid. In *European Across Grids Conference*, pp. 11–20, 2004.
- [DC08] E. Deelman and A. Chervenak. Data Management Challenges of Data-Intensive Scientific Workflows. In *8th IEEE Intl. Symposium on Cluster Computing and the Grid (CCGRID)*, pp. 687–692. IEEE Computer Society, 2008.
- [DF08] S. B. Davidson and J. Freire. Provenance and Scientific Workflows: Challenges and Opportunities. In *SIGMOD Conference*, pp. 1345–1350, 2008.
- [DGST09] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.
- [Esc] Defining e-Science. <http://www.nesc.ac.uk/nesc/define.html>.
- [FG06] G. C. Fox and D. Gannon, editors. *Concurrency and Computation: Practice & Experience. Special Issue: Workflow in Grid Systems*, volume 18(10). John Wiley & Sons, 2006.
- [FHW⁺07] P. Fisher, C. Hedeler, K. Wolstencroft, H. Hulme, H. Noyes, S. Kemp, R. Stevens, and A. Brass. A systematic strategy for large-scale analysis of genotype phenotype correlations: identification of candidate genes involved in African trypanosomiasis. *Nucleic Acids Res*, 35(16):5625–5633, 2007.
- [FKSS08] J. Freire, D. Koop, E. Santos, and C. T. Silva. Provenance for Computational Tasks: A Survey. *Computing in Science and Engineering*, 10(3):11–21, 2008.

- [FSC⁺06] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing Rapidly-Evolving Scientific Workflows. In *Intl. Provenance and Annotation Workshop (IPAW)*, volume 4145 of *Lecture Notes in Computer Science*, pp. 10–18. Springer, 2006.
- [GBA⁺07] A. Goderis, C. Brooks, I. Altintas, E. A. Lee, and C. A. Goble. Composing Different Models of Computation in Kepler and Ptolemy II. In Y. Shi, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, editors, *International Conference on Computational Science (3)*, volume 4489 of *Lecture Notes in Computer Science*, pp. 182–190. Springer, 2007.
- [GDE⁺07] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the Challenges of Scientific Workflows. *Computer*, 40(12):24–32, 2007.
- [GRA08] Grid Resource Allocation and Management (GRAM), 2008. <http://www.globus.org/toolkit/docs/4.2/4.2.0/developer/globusrun-ws.html>.
- [GRD⁺07] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, and J. Kim. Wings for Pegasus: Creating Large-Scale Scientific Applications Using Semantic Representations of Computational Workflows. In *AAAI*, pp. 1767–1774, 2007.
- [GRM05] GridLab Resource Management System (GRMS), 2005. <http://www.gridlab.org/WorkPackages/wp-9/>.
- [GSH⁺08] C. Goble, R. Stevens, D. Hull, K. Wolstencroft, and R. Lopez. Data curation + process curation = data integration + science. *Briefings in Bioinformatics*, December 2008.
- [GWG⁺07] C. Goble, K. Wolstencroft, A. Goderis, D. Hull, J. Zhao, P. Alper, P. Lord, C. Wroe, K. Belhajjame, D. Turi, R. Stevens, and D. D. Roure. Knowledge discovery for in silico experiments with Taverna: Producing and consuming semantics on the Web of Science. In C. Baker and K.-H. Cheung, editors, *Semantic Web: Revolutionizing Knowledge Discovery in Life Sciences*. Springer Science+Business Media, LLC, New York, 2007.
- [HSL⁺04] D. Hull, R. Stevens, P. Lord, C. Wroe, and C. Goble. Treating shimantic web syndrome with ontologies. In *First Advanced Knowledge Technologies workshop on Semantic Web Services (AKT-SWS04)*, CEUR-WS.org, Volume 122, 2004.
- [JE07] D. Jordan and J. Evidemo. Web Services Business Process Execution Language, Version 2.0 (WS-BPEL 2.0), 11 April 2007.
- [KA05] J. Kommineni and D. Abramson. GriddLeS Enhancements and Building Virtual Applications for the GRID with Legacy Components. In *Advances in Grid Computing (European Grid Conference)*, pp. 961–971, 2005.
- [Kah74] G. Kahn. The Semantics of Simple Language for Parallel Programming. In *IFIP Congress*, pp. 471–475, 1974.
- [KBB⁺04] D. S. Katz, A. Bergou, G. B. Berriman, G. L. Block, J. Collier, D. W. Curkendall, J. Good, L. Husman, J. C. Jacob, A. C. Laity, P. Li, C. Miller, T. Prince, H. Siegel, and R. Williams. Accessing and Visualizing Scientific Spatiotemporal Data. In *16th Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, pp. 107–110. IEEE Computer Society, 2004.
- [KBB⁺05] S. A. Klasky, M. Beck, V. Bhat, E. Feibush, B. Ludäscher, M. Parashar, A. Shoshani, D. Silver, and M. Vouk. Data Management on the Fusion Computational Pipeline. *Journal of Physics: Conference Series*, 16:510–520, 2005.

- [KCA⁺06] S.-H. Ku, C. Chang, M. Adams, J. Cummings, F. Hinton, D. Keyes, S. Klasky, W. Lee, Z. Lin, and S. Parker. Gyrokinetic Particle Simulation of Neoclassical Transport in the Pedestal/Scrape-Off Region of a Tokamak Plasma. *Journal of Physics: Conference Series*, 46:87–91, 2006. Institute of Physics Publishing.
- [KDG⁺08] J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar. Provenance trails in the Wings-Pegasus system. *Concurrency and Computation: Practice & Experience*, 20(5):587–597, 2008.
- [Kep] The Kepler Project, <http://www.kepler-project.org>, 2009.
- [LAB⁺06] 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*, 18(10):1039–1065, 2006.
- [LAB⁺09] B. Ludäscher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D. D. Roure, J. Freire, C. Goble, M. Jones, S. Klasky, T. McPhillips, N. Podhorszki, C. Silva, I. Taylor, and M. Vouk. Scientific Process Automation and Workflow Management. In A. Shoshani and D. Rotem, editors, *Scientific Data Management: Challenges, Existing Technology, and Deployment*, Computational Science Series, chapter 13. Chapman & Hall/CRC, 2009.
- [LAWG05] P. Lord, P. Alper, C. Wroe, and C. Goble. Feta: A Light-Weight Architecture for User Oriented Semantic Service Discovery. In *2nd European Semantic Web Conference*, volume 3532 of LNCS, pp. 17–31, Crete, 2005. Springer.
- [LBM09] B. Ludäscher, S. Bowers, and T. McPhillips. Scientific Workflows. In T. Özsu and L. Liu, editors, *Encyclopedia of Database Systems*. Springer, 2009.
- [LG05] B. Ludäscher and C. Goble, editors. *ACM SIGMOD Record: Special Issue on Scientific Workflows*, volume 34(3), September 2005.
- [LM08] E. A. Lee and E. Matsikoudis. The Semantics of Dataflow with Firing. In G. Huet, G. Plotkin, J.-J. Lévy, and Y. Bertot, editors, *From Semantics to Computer Science: Essays in memory of Gilles Kahn*. Cambridge University Press, March 2008. preprint, <http://ptolemy.eecs.berkeley.edu/publications/papers/08/DataflowWithFiring/>.
- [LP95] E. A. Lee and T. M. Parks. Dataflow Process Networks. In *Proceedings of the IEEE*, pp. 773–799, 1995.
- [LPA⁺08] B. Ludäscher, N. Podhorszki, I. Altintas, S. Bowers, and T. McPhillips. From Computation Models to Models of Provenance: The RWS Approach. *Concurrency and Computation: Practice & Experience*, 20(5):507–518, 2008.
- [LSK⁺06] A. Lathers, M.-H. Su, A. Kulungowski, A. Lin, G. Mehta, S. Peltier, E. Deelman, and M. Ellisman. Enabling parallel scientific applications with workflow tools. In *Challenges of Large Applications in Distributed Environments, 2006 IEEE*, pp. 55–60, 2006.
- [LWMB09] B. Ludäscher, M. Weske, T. McPhillips, and S. Bowers. Scientific Workflows: Business as Usual? In U. Dayal, J. Eder, J. Koehler, and H. Reijers, editors, *7th Intl. Conf. on Business Process Management (BPM)*, LNCS 5701, Ulm, Germany, 2009.
- [MBL06] T. McPhillips, S. Bowers, and B. Ludäscher. Collection-Oriented Scientific Workflows for Integrating and Analyzing Biological Data. In *3rd Intl. Workshop on Data Integration in the Life Sciences (DILS)*, LNCS, European Bioinformatics Institute, Hinxton, UK, July 2006. Springer.
- [MBZG08] P. Missier, K. Belhajjame, J. Zhao, and C. Goble. Data lineage model for Taverna workflows with lightweight annotation requirements. In *Intl. Provenance and Annotation Workshop (IPAW)*, 2008.

- [MBZL09] T. McPhillips, S. Bowers, D. Zinn, and B. Ludäscher. Scientific Workflows for Mere Mortals. *Future Generation Computer Systems*, 25(5):541–551, 2009.
- [MCD⁺05] P. Maechling, H. Chalupsky, M. Dougherty, E. Deelman, Y. Gil, S. Gullapalli, V. Gupta, C. Kesselman, J. Kim, G. Mehta, B. Mendenhall, T. A. Russ, G. Singh, M. Spraragen, G. Staples, and K. Vahi. Simplifying Construction of Complex Workflows for Non-Expert Users of the Southern California Earthquake Center Community Modeling Environment. In Ludäscher and Goble [LG05].
- [MDM⁺07] N. Mandal, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Integrating existing scientific workflow systems: the Kepler/Pegasus example. In *2nd Workshop on Workflows in Support of Large-Scale Science (WORKS'07)*, pp. 21–28, New York, NY, USA, 2007. ACM.
- [Mor08] L. Moreau, *et al.* The First Provenance Challenge. *Concurrency and Computation: Practice and Experience – Special Issue on the First Provenance Challenge*, 20(5), 2008.
- [MPMF⁺06] J. Muench, H. P. Maechling, Francoeur, D. Okaya, and Y. Cui. SCEC Earthworks Science Gateway: Widening SCEC Community Access to the TeraGrid. In *The first annual TeraGrid Conference*, 2006.
- [nsf] National Science Foundation, Office of Cyberinfrastructure. <http://www.nsf.gov/dir/index.jsp?org=OCI>, 2009.
- [OAF⁺04] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17), 2004.
- [OGA⁺06] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. In Fox and Gannon [FG06].
- [Peg] The Pegasus Project, <http://pegasus.isi.edu>, 2009.
- [PLK07] N. Podhorszki, B. Ludäscher, and S. A. Klasky. Workflow Automation for Processing Plasma Fusion Simulation Data. In *2nd Workshop on Workflows in Support of Large-Scale Science (WORKS)*, pp. 35–44, 2007.
- [RB96] J. R. Rice and R. F. Boisvert. From Scientific Software Libraries to Problem-Solving Environments. *IEEE Computational Science and Engineering*, 3:44–53, 1996.
- [RG08] C. Rueda and M. Gertz. Real-Time Integration of Geospatial Raster and Point Data Streams. In B. Ludäscher and N. Mamoulis, editors, *20th Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, volume 5069 of LNCS, pp. 605–611, Hong Kong, China, July 2008. Springer.
- [RG09] D. D. Roure and C. Goble. Software Design for Empowering Scientists. *IEEE Software*, 26(1):88–95, January/February 2009.
- [RGS09] D. D. Roure, C. Goble, and R. Stevens. The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows. *Future Generation Computer Systems*, 25:561–567, 2009.
- [SAC⁺07] A. Shoshani, I. Altintas, A. Choudhary, T. Critchlow, C. Kamath, B. Ludäscher, J. Nieplocha, S. Parker, R. Ross, N. Samatova, and M. Vouk. SDM Center Technologies for Accelerating Scientific Discoveries. *Journal of Physics: Conference Series*, 78, 2007.
- [Sci] Scientific Discovery through Advanced Computing (SciDAC), Department of Energy. <http://www.scidac.gov/>, 2009.

- [SKS⁺08a] C. Scheidegger, D. Koop, E. Santos, H. Vo, S. Callahan, J. Freire, and C. Silva. Tackling the Provenance Challenge one layer at a time. *Concurrency and Computation: Practice & Experience*, 20(5):473–483, 2008.
- [SKS⁺08b] C. E. Scheidegger, D. Koop, E. Santos, H. T. Vo, S. P. Callahan, J. Freire, and C. T. Silva. Tackling the Provenance Challenge one layer at a time. *Concurrency and Computation: Practice and Experience*, 20(5):473–483, 2008.
- [SPG05] Y. Simmhan, B. Plale, and D. Gannon. A Survey of Data Provenance in e-Science. In Ludäscher and Goble [LG05].
- [SRG03] R. D. Stevens, A. J. Robinson, and C. A. Goble. myGrid: Personalised Bioinformatics on the Information Grid. In *11th Intl. Conf. on Intelligent Systems for Molecular Biology (ISMB)*, pp. 302–304, 2003.
- [SSV⁺08] G. Singh, M.-H. Su, K. Vahi, E. Deelman, G. B. Berriman, J. Good, D. S. Katz, and G. Mehta. Workflow task clustering for best effort systems with Pegasus. In *Mardi Gras Conference*, page 9, 2008.
- [SVK⁺07] C. E. Scheidegger, H. T. Vo, D. Koop, J. Freire, and C. T. Silva. Querying and Creating Visualizations by Analogy. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1560–1567, 2007.
- [Tav] The Taverna Project, <http://www.taverna.org.uk>, 2009.
- [Tay06] I. Taylor. Triana Generations. In *2nd Intl. Conf. on e-Science and Grid Technologies (e-Science)*, page 143. IEEE Computer Society, 2006.
- [TDGS07] I. Taylor, E. Deelman, D. Gannon, and M. Shields, editors. *Workflows for e-Science: Scientific Workflows for Grids*. Springer, 2007.
- [Tea02] The Condor Team. *Condor Version 7.0.4 Manual*. University of Wisconsin-Madison, 2002. <http://www.cs.wisc.edu/condor/manual/v7.0/>.
- [Tri] The Triana Project, <http://www.trianacode.org>, 2009.
- [TSWR03] I. J. Taylor, M. S. Shields, I. Wang, and O. F. Rana. Triana Applications within Grid Computing and Peer to Peer Environments. *Journal of Grid Computing*, 1(2):199–217, 2003.
- [Vis] The VisTrails Project, <http://www.vistrails.org>, 2009.
- [Wes07] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.
- [WHH05] Y. Wang, C. Hu, and J. Huai. A New Grid Workflow Description Language. In *IEEE International Conference on Services Computing, IEEE SCC*, pp. 257–260. IEEE Computer Society, 2005.
- [WNB07] R. Wolski, D. Nurmi, and J. Brevik. An Analysis of Availability Distributions in Condor. In *IPDPS*, pp. 1–6, 2007.
- [WWVM96] J. Wainer, M. Weske, G. Vossen, and C. B. Medeiros. Scientific Workflow Systems. In *Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State of the Art and Future Directions*, 1996.
- [YB05] J. Yu and R. Buyya. A Taxonomy of Scientific Workflow Systems for Grid Computing. In Ludäscher and Goble [LG05].
- [ZBML09] D. Zinn, S. Bowers, T. McPhillips, and B. Ludäscher. X-CSR: Dataflow Optimization for Distributed XML Process Pipelines. In *ICDE*, 2009. See also Technical Report CSE-2008-15.