Automation of Network-Based Scientific Workflows

M. A. Vouk[1], I. Altintas[2], R. Barreto[3], J. Blondin[4], Z.Cheng[1], T. Critchlow[5], A. Khan[6], S. Klasky[3], J. Ligon[1], B. Ludaescher[7], P. A. Mouallem[1], S. Parker[6], N. Podhorszki[7], A. Shoshani[8], C. Silva[6]

[1]Department of Computer Science, North Carolina State University, Box 8206, Raleigh, NC 27695, USA, {vouk, zcheng, jtligon, pmouall}@ncsu.edu
[2]San Diego Supercomputing Center, University of California, La Jolla, CA 92093, USA, altintas@sdsc.edu
[3]Oak Ridge National Laboratory, PO BOX 2008, MS6008, Oak Ridge, TN 37831, USA, {barreto, klasky }@ornl.gov
[4]Department of Physics, North Carolina State University, Box 8202, Raleigh, NC 27695, USA, John_Blondin@ncsu.edu
[5]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94550, USA, critchlow1@llnl.gov
[6]Department of Computer Science, University of Utah, Salt Lake City, UT 84112, USA, {ayla, sparker,csilva }@cs.utah.edu
[7]Department of Computer Science, University of California Davis, Davis, CA 95616, USA {ludaesch, pnorbert}@ucdavis.edu
[8]Computing Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA, shoshani@lbl.gov

Abstract. Comprehensive, end-to-end, data and workflow management solutions are needed to handle the increasing complexity of processes and data volumes associated with modern distributed scientific problem solving, such as ultra-scale simulations and high-throughput experiments. The key to the solution is an integrated network-based framework that is functional, dependable, fault-tolerant, and supports data and process provenance. Such a framework needs to make development and use of application workflows dramatically easier so that scientists' efforts can shift away from data management and utility software development to scientific research and discovery. An integrated view of these activities is provided by the notion of scientific workflows - a series of structured activities and computations that arise in scientific problem-solving. An information technology framework that supports scientific workflows is the Ptolemy II based environment called Kepler. This paper discusses the issues associated with practical automation of scientific processes and workflows and illustrates this with workflows developed using the Kepler framework and tools.

1 Introduction

Scientific research is exploratory in nature. Scientists carry out experiments, often in a trial and error manner, and they modify the steps of the tasks performed as exploration proceeds. As technology advances, more and more scientists are relying on computing systems to aide them in this process. In fact, some of the heaviest users of computing are in the sciences, and often it is no longer possible for scientists to carry out their day-to-day activities without heavy use of computing. This holds in the fields and problem areas as diverse as computational medicine, biology, chemistry, genetics, environment, fusion and combustion.

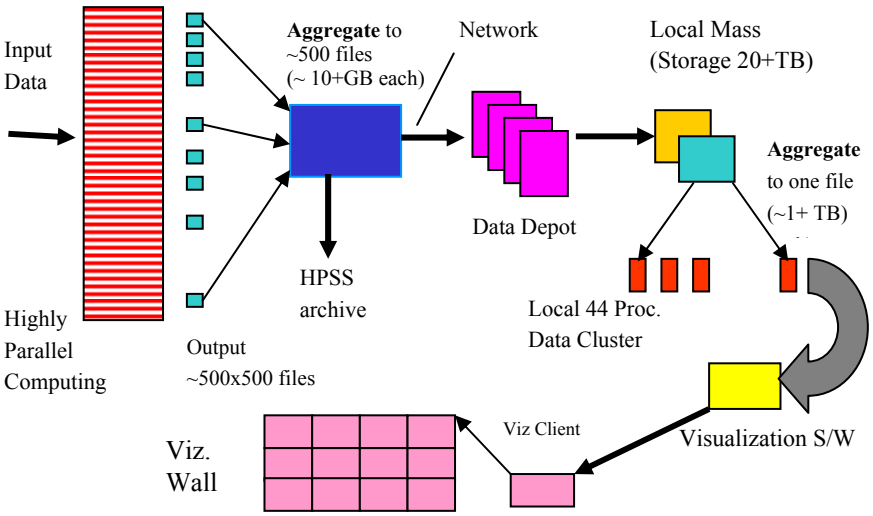

Fig. 1. Illustration of an astrophysics simulation workflow. Computations are done at a remote supercomputer, and the resulting data sets are transferred to NC State University via a high-speed internet link. This is followed by local "slicing and dicing" of the data, and their analysis and visualization.

We use the term ***scientific workflow*** to describe a series of structured activities and computations (we call them workflow components or *actors*[1]) that arise in scientific research and problem-solving process [11]. A scientist may divide the overall task into smaller sub-tasks, each of which can be considered to be an individual step in an experiment or a simulation. At each step, the results can be

[1] The term "actor" is the one used in the Kepler [2] workflow support system based on Ptolemy II framework [12] to describe process components interconnected by data flows and orchestrated by a "director" or a workflow control process. In general, a process oriented network can be described using generalized activity networks [13]. Activity oriented networks have nodes interconnected by data flows and their graph-based depictions are sometimes called actigrams, while data-oriented networks have data nodes interconnected by data transforming activity links and their graph-based depictions are sometimes called datagrams (not to be confused with internet protocol datagrams).

generated, managed, analyzed, stored, or otherwise processed, and then used as an input to the next step in the process. Such reuse of data can be done repeatedly until the overall task is completed to scientist's satisfaction. We use the term "workflow" to describe the chaining of smaller tasks to achieve the desired results using data from different source in combination with different transformation, analysis and visualization services, [1, 11]. Today, many – often all – of the steps involve support from or interaction with information technology. Scientific workflow includes actions performed (by actors), decisions made (control-flow), information transferred (data-flow), exception and interrupt handling (e.g., event-flows) and the underlying coordination and scheduling required to execute a workflow (orchestration). In its simplest case, a workflow is a linear sequence of tasks, each one implemented by an actor.

An example of a workflow is: a) transfer of executable simulation application code and computational and storage configuration information to a cluster or a high-performance computer, b) running of this application, and c) transferring of the results to a remote machines for further analysis and visualization. Figure 1 illustrates such a workflow.

Comprehensive end-to-end data and workflow management solutions are needed to handle the increasing complexity of processes and data volumes associated with modern distributed scientific problem solving, such as ultra-scale simulations and high-throughput experiments. The key to the solution is an integrated network-based framework that is functional, dependable, fault-tolerant, and supports data and process provenance. Such a framework needs to make application workflows dramatically easier to develop and use [36] so that scientists' efforts can shift away from data management and application development to scientific research and discovery. A Ptolemy II based environment called Kepler [2] is one such framework.

This paper discusses the issues associated with practical automation of scientific processes and workflows and illustrates this through workflows developed using the Kepler framework and tools.

2 Workflows

Workflow technologies have a long history in the databases and information systems communities [1]. Scientific community has developed a number of problem-solving environments, most of them as integrated solutions [24 and references there in]. However, more recently component-based solution support systems have become more popular [e.g., 14, 25, 26, 29, 30]. Scientific workflows merge advances in all these areas to automate support for sophisticated scientific information technology assisted exploration and problem-solving [e.g., 2 – 11, 46, 55, 61].

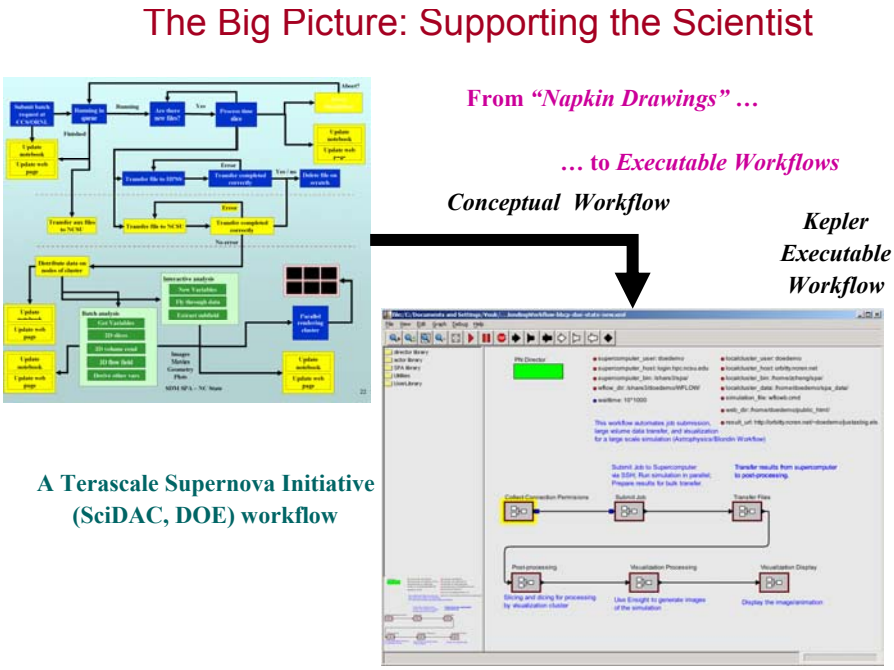

Fig. 2. From a "napkin drawing" to an executable Kepler-based workflow.

Scientific workflows, as we understand them, are crucial to the success of major initiatives in high-performance computing. As parallel computing expands, their standards encourage scientists to construct complex distributed solutions that span the networks, and through web-based interfaces and virtualization invite incorporation into still more complex systems that may include interactions with economic and business flows. Workflows provide the necessary abstractions that enable effective usage of computational resources, and development of robust problem-solving environments that marshal high-performance computing resources.

Workflows have many synergies with web and network-based services. In fact, (web) service based workflows are quickly becoming a requirement of a wide range of new service-oriented applications. Many domain experts, particularly in life sciences, do not wish to construct workflows by coding them beyond what is necessary to do research in their domain, e.g., to develop appropriate algorithms. They would like to considerably reduce the overhead currently required by some information technology solutions. That overhead can be as much a 50% of the activity. Therefore, workflow automation and higher-level specification fits naturally into the trends towards increased domain specialization as application developers move to become (web) services providers, and computer scientists seek reusable libraries and tools, rather than custom made applications.

Of course, workflows such as the one shown in Figure 1 have much more depth and structure than shown in the Figure 1 diagram. Often they can be naturally mapped onto graph representations, e.g., [13, 30]. Typically, a scientist would like to go from a conceptual "napkin drawing" of a workflow to an executable version of it with as little overhead from the information technology tools and solutions as

possible (Figure 2). Sometimes the best way to manage complexity of such structures is to nest the graphs (e.g., Figure 3). A graph can then be translated into executable form either manually or automatically. However, the process can be a reverse one – the code and some process scripts for an, in part, manually assisted workflow already exists, and the workflow technology is used to integrate these elements. In either case, it is beneficial to keep a high level graph representation of a workflow so that end-users can better understand and modify application logic.

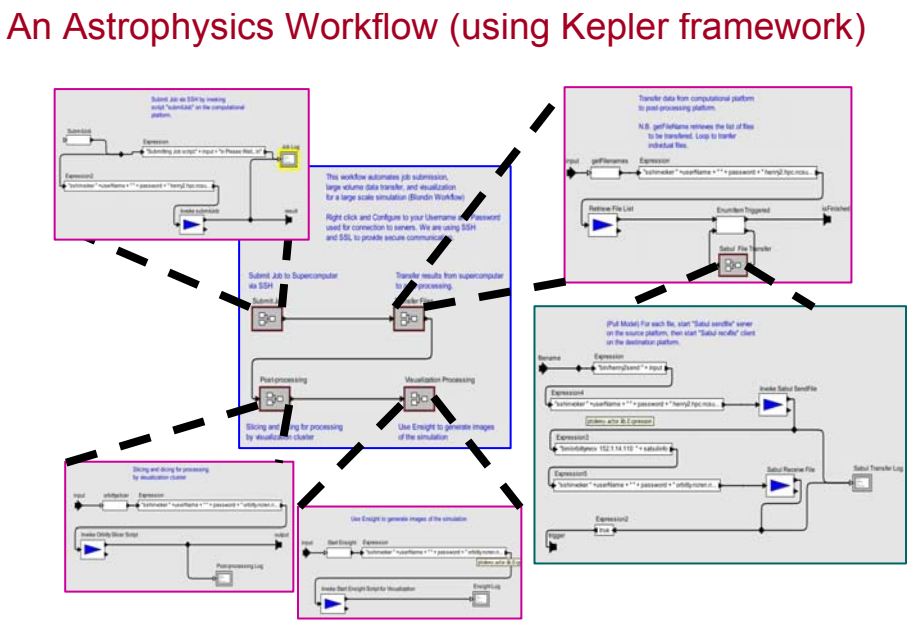

Fig. 3. Nesting can help manage complexity of workflows.

Scientific workflows can exhibit and exploit data-, task-, and pipeline-parallelism. In science and engineering workflow processes, tasks and computations are often large-scale, complex, and structured with intricate dependencies [7, 13, 14]. Information technology assisted scientific workflows have several common characteristics:

- **Composition**. Scientific workflows require invocation, interconnection and integration of multiple data collection, simulation, application or **analysis** elements, i.e., methods, approaches, tools and processes. While these elements are often invoked in a routine manner, there may also be changes in the workflow as scientists interactively explore new options. Developing an executable workflow requires resolving mismatches between what an element expects and what the previous step in the process generated.
- **Diversity**. Scientific workflows require significant **heterogeneous**, computational, storage and networking **resources**. Many large-scale

scientific workflows will execute for hours, often days, perhaps weeks and months, and may require user intervention at multiple times. If the workflow, or one of the associated computations or activities runs into trouble, **fault-tolerant behavior**, e.g., via human intervention or perhaps automated failover or recovery techniques must be attempted because returning to the initial starting point is usually not acceptable.

- **Verification and validation** of processes as well as intermediate and final results is essential in the domain of scientific problem solving. This ensures integrity of the data, processes and results, that the activity as a whole remains on track, and that resources are not wasted. Often real-time or near-real-time status tracking and preservation of state capabilities are required. One of the most difficult (and currently not yet fully solved) issues is **semantic** validity of workflows. Semantic mismatches between workflow components, tools and data must be handled in order to maintain confidence in the results. For example, some of the tools may be designed for performing simulations under different circumstances or assumptions, and this must be accommodated to prevent spurious results.
- **Evolution**. Because of their evolutionary and exploratory nature, **frequent changes** are often an integral part of a scientific workflow lifecycle. Therefore, is critical to record **provenance** information (e.g., the lineage of data and processes) in a way that is consistent, persistent, and easily retrievable and auditable. Related to this is the ability to **steer** the workflows and the associated computational tasks through use of run-time **dashboards**, analytics and process feedback loops.

3 Overhead

In the 21st century, a key differentiating characteristic of a successful information technology (IT) is its ability to become true and valuable contributor to cyberinfrastructure. Cyberinfrastructure [36] makes IT systems, applications and services dramatically easier to develop, deploy and use. This expands the scope of applications and services possible within budget and organizational constraints. It also increases efficiency, quality, and reliability by capturing commonalities and by facilitating efficient sharing of resources and services. Ultimately, cyberinfrastructure shifts the effort away from IT (overhead) concentrating it on the basic end-user mission and business.

Appropriate cyberinfrastructure is especially important for any business that in large part relies on IT to conduct its daily operations. Today, this is true of many financial, educational, research, government and retail organizations. From the perspective of an end-user IT must be enabling and appliance-like. End-users should be able to use the technology to improve their productivity and reduce technology-driven overhead, e.g., software installation or management. For example, unless IT is the primary business of an organization or an individual, less than 20% of its effort not directly connected to its primary business should have to do with IT issues, even though 80% of its business may be conducted using electronic means. In general, infrastructure installation and maintenance overhead must have the property of the economy-of-scale at all levels – hardware, software, provisioning, maintenance, etc.

A powerful cyberinfrastructure enabling concept is utility-computing through service-oriented architectures (SOA) [e.g., 50]. An SOA is an environment where end-users can request an IT service at the desired functional, quality and capacity level, and receive it either at the time requested or at a specified later time. A key enabler of SOA is component-based construction of services. Another key supporting technology is virtualization of IT resources and services. It is expected that in the next 10 years, service-based solutions will be a major vehicle for delivery of information and other IT assisted functions at both individual and organizational levels, e.g., software applications, web-based services, even personal and business "desktop" computing.

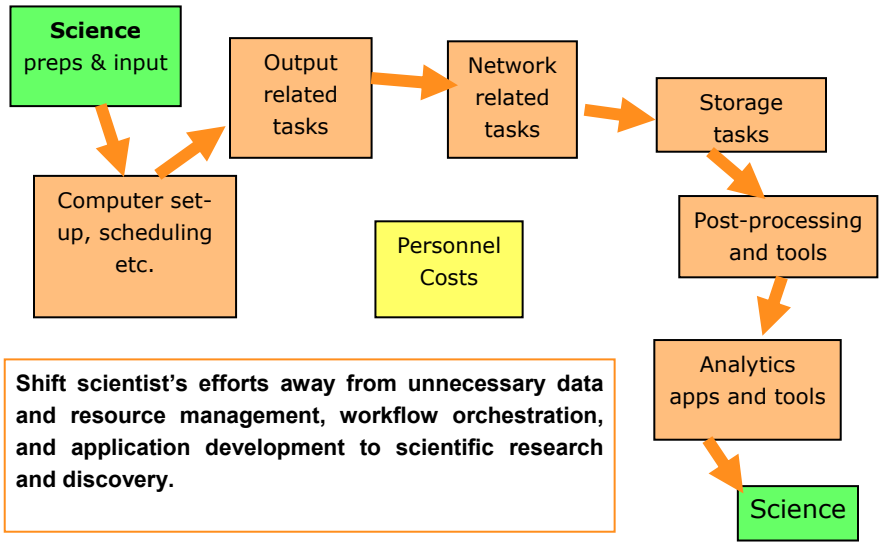

Fig. 4. A typical scientist is primarily interested in preparation of inputs and codes (green areas) related to his/her specific research domain and in doing "science," i.e., discovery. There is much less interest in tending computers, moving data or developing peripheral IT applications (orange areas).

Scientific computing is no different in this respect. Today a scientist involved with a large-scale scientific workflow, e.g., of the peta-scale class of problems, may spend a lot of time dealing with IT related activities they need, but often wish they did not have to do [37]. For example, a typical class of heavy-duty scientific simulation workflows may have abstraction steps shown in Figure 4. A typical scientist's primary interest is in preparation of inputs and codes related to her or his specific research domain and in doing domain specific scientific discovery. Unless IT is the research or development passion of the scientist, there is much less interest in tending computers, moving data or developing peripheral IT applications and support tools (e.g., visualization frameworks). Yet, as much as 50%, sometimes even more, of a scientist's time may be taken up by IT tasks that can be, but are not, automated and/or easy to use. Obviously, there is a need to improve on this. In fact, this has prompted a number of entities (including the US Department of Energy) to

sponsor research and development projects[2] aimed at making scientists more productive.

4 Component-based Construction

Component-based construction of solutions, of course, is not a new concept. It has been one of the "holy grails" of software engineering since its earliest days. Results have been mixed so far. However, the advent of reliable and readily available networked resources, and especially of service-oriented architecting, makes truly component-based construction of large scale distributed software-based solutions viable reality.

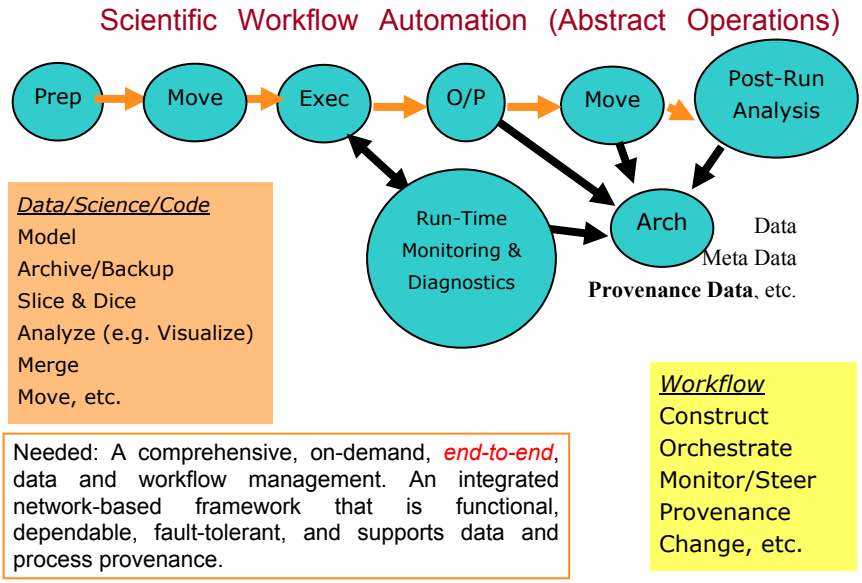

Fig. 5. Workflow abstraction.

Consider the Figure 4 abstraction in a somewhat different light (Figure 5). The flow starts with preparation of the domain-specific codes and inputs. In computationally very intense workflows, these preparatory activities may happen in environments that are different from the one where the code will actually execute. This is followed by moving of the data and codes to host (or grid) that will execute the simulation (e.g., a high-end supercomputer). Once the execution is scheduled (a request may wait in a queue for resources), the scientist may wish to monitor its run-time progress, handle run-time diagnostics, perhaps steer the computations, and certainly collect outputs and results. Outputs of large-scale computations may not remain where they are generated, but may move to a post-run data manipulation and

[2] For example, SciDAC (http://www.scidac.gov/)

analysis environment for slicing, dicing, analytics, visualization, and so on A lot of information, perhaps all, is archived in a permanent way. Furthermore, all through the process there is generation of meta-data (data about data and processes) that is either used directly (perhaps in "dashboards") or is part of the data and process lineage (provenance) information [57].

Implementation of workflow abstractions requires availability of a relevant set of IT-based operations in the form of either software applications or perhaps as commands built into operating systems used. In this context, it is very important to distinguish between a custom-made workflow solution (or a problem-solving environment), and a more canonical set of operations, methods, and solutions that can be composed into a scientific workflow. Former have been around for a long time [e.g., 24 and references therein], latter are emerging. For instance, sort, uniq, grep, ftp, ssh and so on, are typical unix operating system commands that scientists can rely on to be available for workflow construction. It is less certain that a complex tool like SAS (which can also sort data, but also does many other things) is available on all platforms of interest, or that some non-standard data moving utility application such as bbcp is readily available on all platforms of interest. Some, operations such as "slice and dice" or "visualize" data are usually available only in the form of specialized software packages or applications.

On the other hand, expectations are rising. *Scientific community now expects utility-like (appliance-like) on-demand access* to needed IT resources where use of IT-based solutions is not bound to a fixed location (such as a specific lab) and fixed resources (e.g., a particular operating system), but has moved to a (mobile) personal access device of a scientist (e.g., laptop or a PDA or a cell phone) and service-based delivery. Scientists would like to move away from the situation where they have to spend more time on IT development, support and workflow management (art) to a situation where IT support is a commodity and they can focus primarily on their basic scientific mission (Figure 6). They are looking for environment where application workflows are dramatically easier to develop and use. Yet, today a practical bottleneck is often still in the IT domain, i.e., in the *scientific workflow environment* of the end-user scientists.

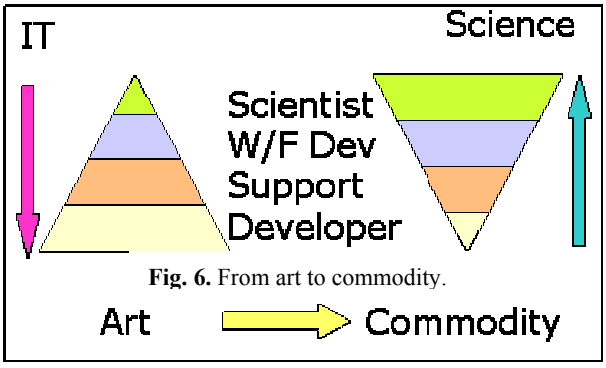

Fig. 6. From art to commodity.

The key to the solution, is an integrated scientific process support framework that is dependable, supports networked or distributed workflows, supports a range of couplings among its building blocks, provides fault-tolerant and data- and process-aware service-based delivery, and provides the capability to audit processes, data and results. Key characteristic of such a framework and its elements are [25, 26]: **reusability** (e.g., elements can be re-used in other workflows), **substitutability** (alternative implementations are easy to insert, very precisely specified interfaces are available, run-time component replacement mechanisms exist, there is ability to verify and validate substitutions, etc), **extensibility** (ability to readily extend system

component pool, increase capabilities of individual components, have an extensible architecture that can automatically discover new functionalities and resources, etc), and **composability** (easy construction of more complex functional solutions using basic components, reasoning about such compositions, etc.).

Components are **assembled** according to the rules specified by a **component model.** Their **coupling** can range from **tight** to **loose**, from **synchronous** and blocking to **asynchronous**. Components are assembled using their **interfaces**. **Component composition** assembles components to form a larger component, an application, or a workflow. All parts must conform to the **component model** or they do not fit together. A **component technology** is a concrete implementation of a component model.

Interoperability among components, or workflows built from components, is a major practical issue. Unless component technologies allow for interoperation among different technologies and component models (perhaps through standardized inter-workflow interfaces), there is a danger that workflows from different groups and communities (who invariably use different component technologies) will create "stove-pipes" that will hamper disciplinary and multidisciplinary project, data exchange and scientific discovery. Steps in that direction are standardization efforts related to workflow description languages, web-services and similar [e.g., 51, 52, 53].

5 Complexity and Usability

A major issue of concern with new technologies, and general purpose scientific workflow support environments are no exception in this context, is complexity and usability. End-to-end workflows involve three types of interactions human-to-human, human-to-machine and vice versa, and machine-to-machine. Human-to-human communications have a relatively slow information exchange rate and are tolerant of both semantic and syntactic errors. Machine-to-machine communications are at the other end of the spectrum. They can take place at very high rates but must use very exact and unambiguous protocols.

Human-to-machine interactions are the most critical from the complexity and usability point of view. Humans need to construct the workflow at some point, and humans are the recipients of the information that emerges from those workflows. As already mentioned, scientific community expects utility-like (appliance-like) on-demand access to needed IT resources and workflow technology and tools must meet cost, complexity, skill level to implement, usability, maintainability, reliability, availability, and other expectations of its users. If it fails to do so, i.e., the overhead brought on by the technology does not exceed the potential value added by its use, technology is typically not be used.

This is illustrated in Figure 7. Some technologies never make the break-even point, some "arrive" at or past the break-even point. A good example of a technology that was widely accepted, because it made access to networked information much more acceptable for a general user, is the Web. Scientific workflow technologies are now approaching the break-even point through reduction in the complexity of workflow construction, increased operational reliability, and provision of a suite of support functionalities and packages scientists expect. One such environment is Kepler.

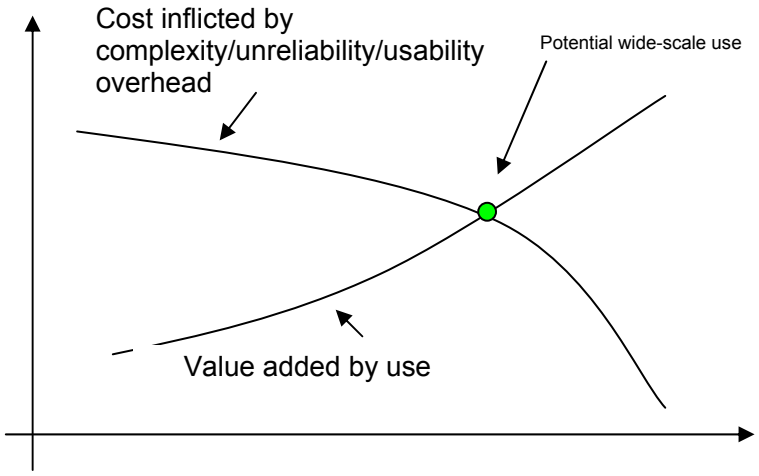

Fig. 7. The relationship between the value added by use of a technology and the overhead inflicted by complexity, reliability, usability and cost of the technology.

6 Kepler

Kepler [2] is an open source component-based scientific workflow support system based on the Ptolemy II framework [12]. Kepler is being developed through a large cross-project collaboration[3]. Basic components of the Kepler framework include: the Ptolemy II core, Kepler core extensions, Kepler object and repository manager, extensions for smart re-run and failure recovery, provenance support modules, a graphical user interface (GUI) layer based on the Ptolemy II Vergil GUI, an authentication layer, a library of generic, application and domain specific actors, and a repository for provenance information. While Kepler can operate without the GUI, it is a useful workflow construction and execution monitoring tool.

Figure 8 illustrates a GUI-level view of a simple workflow. In this case the director is called "PN Director" where PN stands for Process Networks, and as it name implies it implements the process network model. Actor icons can be dragged and dropped from the actor repository (shown on the left in the Figure 8) and connected together with dataflow arcs. Inputs can be parameterized (and their values automatically displayed on the desktop) or can come from files, or be hidden within icons. For example, one would change or input parameters by double clicking on an

[3] SEEK: Science Environment for Ecological Knowledge, SDM Center/SPA: SDM Center/Scientific Process Automation, Ptolemy II: Heterogeneous Modeling and Design, GEON: Cyberinfrastructure for the Geosciences, ROADNet: Real-time Observatories, Applications, and Data Management Network, EOL: Encyclopedia of Life, Resurgence, CIPRes: CyberInfrastructure for Phylogenetic Research, and others.

icon. Once workflow execution has started it can be paused, resumed or stopped, and the flow of information through the workflow can be monitored in real-time. Documentation is an integral part of the system.

Ptolemy II was originally developed to support modeling, simulation, and design of concurrent, real-time, embedded systems. Kepler project has extended this framework to provide access to and use of synchronous and asynchronous loosely and tightly coupled and networked resources and functionalities that are typically used in scientific workflows. From the end-user perspective, there are two principal groups of elements. One set are the computational models - represented by "directors", and the other set are the data-flow connected processing nodes called "actors".

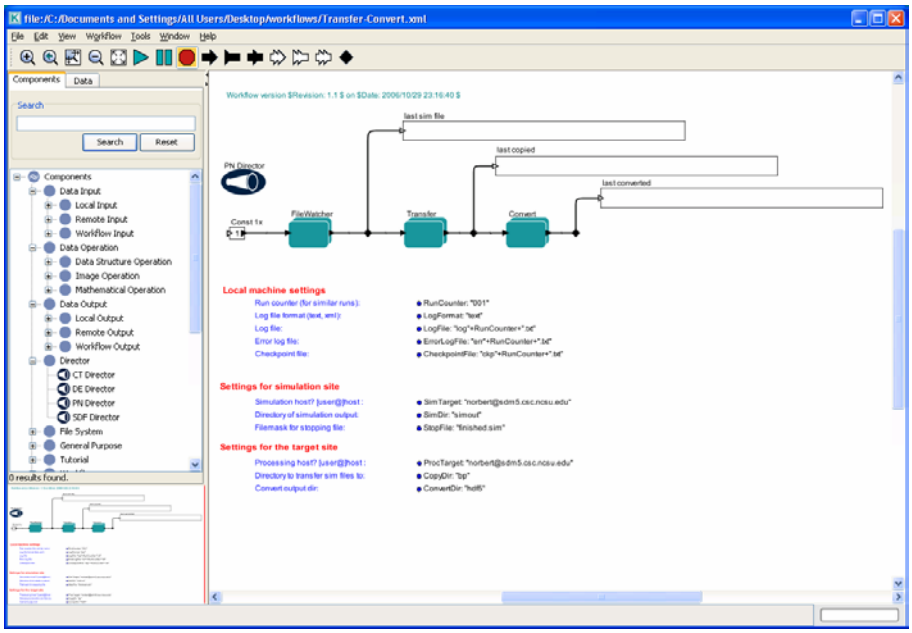

Fig. 8. Kepler GUI.

A director is an engine that controls the behavior and execution of the workflow components. In doing that, it implements different computational models, and thus it defines the semantics of the execution and of the interactions among the actors. While there are a number of open source and closed source scientific workflow support environments [e.g., 2, 3, 4, 42, 43, 55], a very unique and distinguishing feature of the Kepler framework is that (through Ptolemy II) it enables a very rich mixture of models of computation. Examples of realized computational domains range from continuous-time modeling, to dynamic data flow, to discrete-event modeling, to finite state machines, to process networks, to synchronous dataflow modeling, to discrete time and distributed discrete events, and so on.

Actors encapsulate parameterized actions and have interfaces define by ports and parameters. Ports are used to communicate input and output data and streams, but

without call-return semantics. Communication semantics among ports is handled by the directors – one per workflow level – which provide flow control. Workflows can be nested (e.g., Figure 2), and different computational models can be used at different hierarchical levels so long the communication channels and actions that may cross level boundaries are compatible.

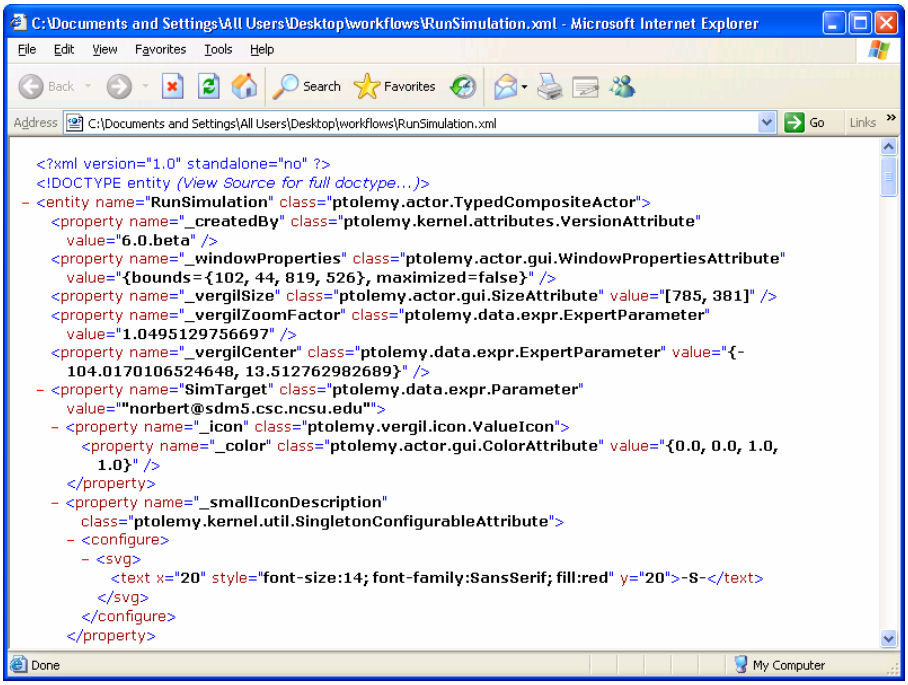

Fig. 9. Part of the XML-based description of the workflow shown in Figure 8.

Actors are typically collected in libraries, many of which are domain and data polymorphic. A number of supporting packages and actors are provided either as part of Ptolemy or as part of Kepler additions. This includes graph-theoretic manipulations, matrix and vector math, signal processing, data typing, handling of generic web services, customizable relational data-based management, command line wrappers for ssh, scp, ftp and similar, a level of Grid support (e.g., GridFTP, certificate generator), native R and Matlab support, SRB – the SDSC storage resource broker, communication with object resource brokers, image processing, visualization, textual and graphical outputs, etc. A number of functionalities are being added or improved, including large-scale robust data movers, more extensive provenance support, semantic-intensive actors, management for data-intensive and compute-intensive workflows, authentication and authorization, distributed execution, execution monitoring, fault tolerance, and scientific data management application-driven extensions such as access to or integration with parallel NetCDF, PVFS, MPI-IO, parallel-R, FastBit, and CCA.

Kepler workflows are recorded and can be exchanged as XML-based Modeling Markup Language (MoML) files [38]. Figure 9 shows part of the MoML description of the workflow shown in Figure 8. Kepler has many other desirable features. For example, if running in a distributed environment, it operates as a relatively loosely coupled system, while operating on a single platform it can operate as a very tightly coupled system. In addition, Kepler allows, using additional middleware that manipulates MoML files, dynamic construction of workflows. Kepler environment is very external-application friendly. It can invoke and communicate with existing tightly coupled external problem solving, analysis or visualization environments (e.g., R, SciRUN [62], Ensight [63]), as well as Grid-based resources..

Kepler is also very flexible in how far it extends into a workflow. It can contain the whole workflow and orchestrate it in a synchronous or asynchronous manner, or it can act only as a control layer, with separate provenance and "heavy lifting" computational, data movement and storage layers. The current implementation is primarily a single instance environment that virtualizes quite well. For example, NC State University Virtual Computing Laboratory [64] offers Kepler to its users as one of its images, and allows users to spawn multiple instances of Kepler workflows for simultaneous use by one or more end-users.

7 Run-Time Monitoring and Provenance

The need to provide run-time monitoring of scientific processes and collect provenance information has been recognized for some time now [e.g., 9, 10, 14, 39, 40 – 45, 57]. Provenance is the history of data, execution and conditions applied to a workflow run. Run-time monitoring may be part of the provenance meta-data, but it also may require collection of additional information and display of that information in a user-friendly format, for example on a "**dashboard**," so that run-time tracking, problem determination, computational steering, and other workflow-related feedback may take place. Such information may also be used to provide fault-tolerance related information (including check-point and recovery data and information), recreate of results and rebuilding of workflows, associate workflows with results it produced, create links between generated data in different runs, compare different runs, checkpoint a workflow and recover, debug and explain results. In general this information can attest to the lineage of the data (**data provenance**, such as intermediate and end results, file names and paths, data-base references, URLs, etc.), processes (**process provenance**, such as software version numbers, the actual workflow graphs or descriptions, events that occurred during a run, input data and parameters used, etc.), error and exception management (error and execution logs), and given the right tools, workflow design provenance.

Kepler currently implements an internal actor-based provenance mechanism [e.g., 2, 41, 54], and several optional portal-based and domain-oriented external process tracking and monitoring mechanisms and dashboards. Under consideration is incorporation of the VisTrails [55] provenance infrastructure into the Kepler framework. VisTrails has extensive support for process and data provenance [57, 58], including visual querying capabilities and multi-user support, which aids collaborative work.

8 Security

Authentication, authorization, access control and security are a major issue with almost any network-based solution available today. The issues take many shapes and forms. Practically all workflow support environments, including Kepler, face these problems. For example, actors need to manage data, programs, and computing resources in distributed and heterogeneous environments, and that has to happen under a variety of security conditions – from very stringent ones (possibly military grade) to relatively relaxed ones (e.g., academic institutions). While some of the components may be operating under "grid" authentication rules (e.g., via certificates, such as using the GAMA framework [59]), some may use LDAP [60] based authentication, while others may be using yet another approach. How does one reconcile these mechanisms to allow trusted exchange of information among the workflow components?

When accessing higher-security resources (e.g., in the National Laboratories) users are required to use encrypted connections (e.g., ssh-based, ssl-based, secure HTTP aided solutions, etc.) and often one-time passwords and other security devices. While secure connections are typically not an issue in workflow environments (e.g., Kepler has ssh and other appropriate actors), one time passwords can be an impediments since they may require use of special keys or security devices that prevent one-stop authentication paths, and in practice invariably require human intervention, thus slowing workflow related operations and communications that span authentication domains.

While workflow related security, authentication, authorization and associated access control have been studied extensively over the last 10 or more years, the problem is still here. A more encompassing solution remains work in progress when it comes to scientific workflow environments. Usable environments support different authentication mechanisms but until identity management and security are treated in a more uniform way, they may represent a major obstacle to interoperability among different workflow frameworks and solutions.

9 Fault-Tolerance

Application of the workflow technology to a specific domain or project, requires information about the domain, project content, participants (both developers and end-users), schedules, resources, other relevant technology, and development of the corresponding operational profiles for the scientific workflow system. Operational profile is the set of relative frequencies which tells us how often a particular scenario, function or capability occurs in practice [34]. Specifically, one would first identify and categorize workflow system users, functionalities and resources and frequency of use of each. This would allow mapping amongst them. This finally yields an operational profile that needs to be supported during the workflow system use. The mappings and the operational profile allow us to recognize functional alternatives and introduce adaptive or fault-tolerant behavior into the model.

There are two basic forms of run-time fault-tolerance: forward-recovery (which includes failure masking and redundancy based failover), and backward-recovery

(which includes check-pointing) e.g. [15, 16, 31]. Exception handling is a very traditional way of managing run-time problems [e.g., 16, 31]. It is also used in the workflow-oriented environments [e.g., 48, 49]. Exception handling can involve forward-recovery, backward-recovery, or graceful termination. More recently web-services community has recognized the need for some form of standardized fault-tolerance in the service provisioning through replication [56].

A run-time failure of a system is often the result of a series of events – sometimes that of a set of very complex and unexpected interactions. Typically, a failure is a result of either a system fault − a design-time developer or researcher error that materializes at run-time, or it is a user error at execution time, or there is an issue with the underlying infrastructure (including invoked services). An initial design error can become a fault in the initial product. This fault can propagate (as a series of defects) to the final executable version of the workflow. When the workflow encounters that defect during execution, the workflow enters an error-state. If that error-state, or its result, becomes visible to the end-user, it becomes a failure that may have anywhere from no consequences to catastrophic consequences. Similarly, a call to a workflow component that fails at run-time may again force the workflow into an error-state, and manifest as a failure. So, run-time workflow failures can be caused by one or more of the following non-comprehensive set of events:

- Use errors caused by end user, for example entering incorrect data.
- Error-state caused by network difficulties, such as congestion.
- Error-state initiated by workflow component faults due to a programming issue.
- Hardware faults and failures
- Error-states caused by failures of services, such as the unavailability of a certain service (e.g., actual web service, or a remote computational or storage service)
- Failures in any underlying software components (e.g., operating system kernel bugs, device misconfigurations, etc.)

9.1 Illustration

In illustrating fault-tolerant solutions in the context of network-based workflows we do not plan to discuss the cost of the services, and we make some **assumptions**:

- Failures of redundant services are **not correlated**, or at least the probability of correlated failures is very low. This basically means that the failure of one service doesn't affect another functionally equivalent (or perhaps replicated) service. For example, we assume that redundant services are hosted on separate perhaps geographically distributed servers, so that in the case one server fails, only one service will be affected. However, assumption also implies that failures are not caused by a basic algorithmic flaw that may be present in both services and may result in identical but wrong responses from all redundant services [16].
- In discussing system reliability, we will make the assumption that all workflow services have the same failure and recovery **probability**. This, of course may not be a realistic assumptions, but it provides a vehicle for the model discussion. An enormous amount of work has been already done by others in modeling and simulation of redundant components under a variety

of conditions, e.g., see [31, 33] and references therein, and the reader should consult that literature before deciding on a particular solution.

The first assumption does not apply to the case when there are no redundant or replicated services, i.e., if we assume that different services used by the workflow are deployed in a serial fashion[4]. This is the worst-case scenario where failure of one service means the failure of the entire workflow since no alternative services are being provided. Of course, one could actually have different functions performed in parallel and at different locations, if the (data) flow model allows that (and this is sometimes done). But, in practical terms a failure of either of those services again fails the workflow. Therefore, serialization assumption can still be used. We also assume that services are atomic, i.e., we do not discuss the option of seamless fail-over once a service has been engaged, we assume that that issue is handled through re-start of that part of the workflow.

Today's web services appear to have a varying and broad range of failure probabilities depending on their implementation and quality, how they are being hosted, who is hosting them, where they are being reached from, etc. One value for average failure probability found in the literature is 0.045, this translates into more than one failure a month. [17, 18, 19]. During peak-time operation, that failure probability may become as high as 0.2, or more than two failures to deliver a service on request as day, depending on the request type and duration. Of course, there are services that do much better and much worse than this range indicates. The level to which improvements need to be made may be domain specific. For example, educational workflows need to have availability of at least 0.95 [35]. We use these two numbers to provide an illustration of a possible range of service failure probabilities upon which one would have to improve, and to illustrate the power of a simple redundancy-based fault-tolerance strategy.

Consider a scientific workflow that invokes serially 3 different network-based services before it is done. Let one of those services fail. Then, the workflow will not finish successfully unless mitigation is put in place. For example, at the point of failure we could recover back to the point where the workflow was check-pointed, and then we could re-run the remaining part. Alternatively, we could try to mask the failure of the component service. We focus on the latter. Given the assumptions above, the probability that the workflow fails is the probability that at least one of the services fails [16,31]. In other words:

$$P_F = 1 - \prod_{i=1}^{n} (1 - p_i) \quad (1)$$

Where p_F is the probability that the workflow will fail, p_i is the probability that service i fails, and n is the number of serial services in the workflow. Using the example numbers and Eq. 1, on the "average" $P_F = 1 - (1-0.045)^3 = 0.129$, and in the "heavy load" case $P_F = 1 - (0.8)^3 = 0.488$. Obviously, reliability R (which is one minus failure probability) is not too good. Figure 10 shows a graph of the failure probability for this example. System failure probability grows considerably with the

[4] However, physical co-location of serial or replicated services runs another risk – that of the whole site power or other type of outage.

number of serial services (Equation 1). In the ”heavy load” situation, the failure probability is very close to one once the number of services in the workflow exceeds 15.

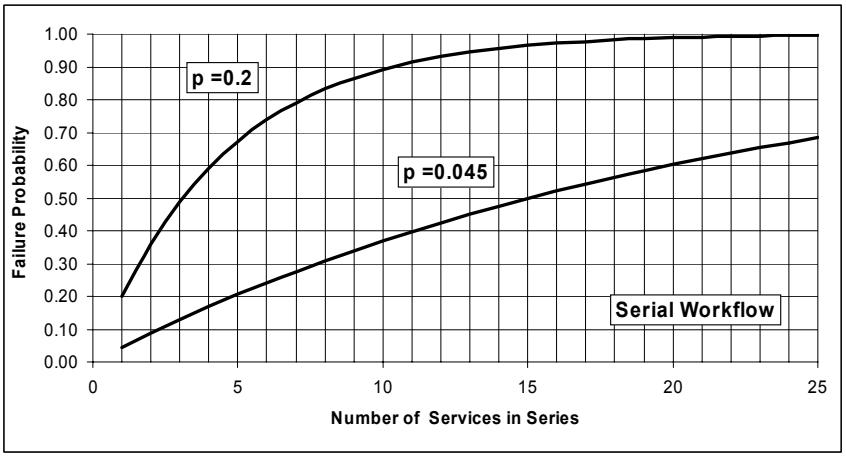

Fig. 10. Failure probability of a serial system without fault-tolerance

While the numbers in the graph may seem excessively pessimistic, it is clear that as the number of services in a complex workflow grows, service failures can have a dramatic effect on the whole system operation. A well known solution is to use multiple backup servers (service replication) in parallel to counter physical infrastructure and networking failures, and/or to use alternative but functionally equivalent services if other types of failures may be suspected. Then, if one of the services fails, the workflow can automatically switch to an alternative one.

Let’s assume that the probability that a service, or any of its alternative fails, is p. Then, the probability that such redundant service fails is that of a group of parallel components, i.e., all of them need to fail before an end-user visible failure occurs. For example, the reliability of a service with 3 alternatives is the probability that at least one of the alternatives is operational, i.e., $R = (1-p) + p(1-p) + p^2(1-p) = 1 - p^3$ Generalizing:

$$R = 1 - p^m \tag{2}$$

where m is the number of alternatives[5]. Applying (1) and (2) to the entire workflow, the failure probability of the workflow would be:

$$P_F = 1 - \left(\prod_{i=1}^{n}\sum_{j=0}^{m-1} p_i^{\,j}(1-p_i)\right) = 1 - \prod_{i=1}^{n}(1-p_i^m) \tag{3}$$

[5] Note: $(1-p) + p(1-p) + p^2(1-p) + \ldots + p^{m-1}(1-p) = 1 - p + p - p^2 + p^2 - p^3 + \ldots + p^{m-2} - p^{m-1} + p^{m-1} - p^m = (1 - p^m)$

where P_F is the probability that the workflow fails, n is the number of service in the workflow, m is the number of replicas of each service in the workflow (in this example, we assume that the number of replicas is the same for all services), and p_i is the probability that service i fails.

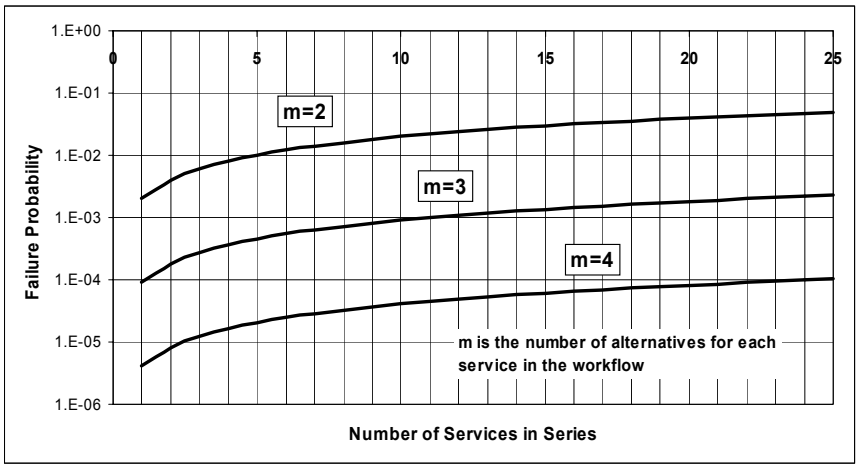

Fig. 11. Failure probability of a series – parallel model (p = 0.045)

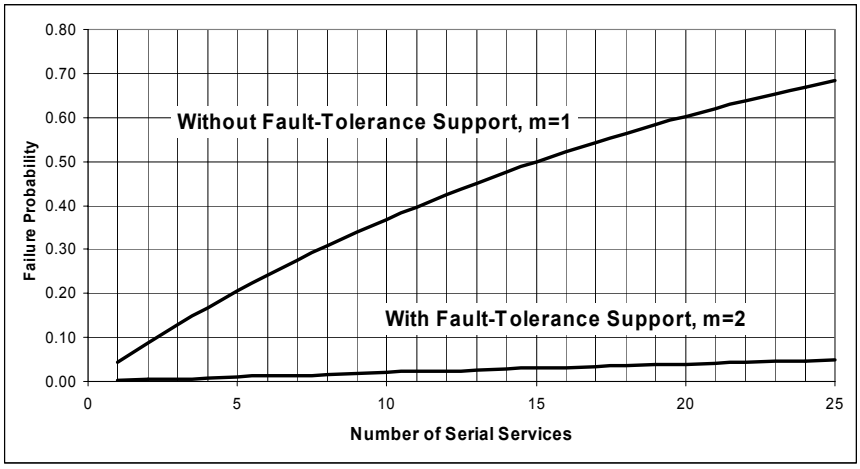

Fig. 12. Failure probability of a serial system with and without fault-tolerance.

Consider again our example with three "serial" services. Let each service have 2 backup services (m=3). Then the probability that at least one of the services fails (thus the workflow fails), and given our two illustration failures rates, $P_F = 1 – (1 – 0.045^3)^3 = 0.00027$ and $P_F = 1- (1- 0.2^3)^3 = 0.0238$. Figure 11 illustrates different

failure probabilities based on this fault-tolerance model given the assumption that p=0.045 for all services. Different lines represent the number of alternatives available for each service. We notice that, even when there is only one alternative for each service (m=2), failure probability is significantly lower than that of a workflow without any fault-tolerance support. For example for a workflow with 25 services to invoke, the failure probability goes down from almost 0.7 to 0.05 for m=2, down to 0.002 for m=3, and down to 0.0001 for m=4. Notice that in this case and with our assumptions, every time we add an alternative service, the failure probability can go down by a factor of 20 or more.

In comparing the two models, we find the potential for considerable improvement using redundancy. Of course, the caveat is that the redundant services must not exhibit significantly correlated failures, either due to their (co-)location, or for algorithmic or other reasons. For example, with p=0.045, n=3 and m=3, workflow failure probability is reduced from 0.129 to less than 10^{-3}. Figure 12 compares failure probability of a workflow with no fault-tolerant support, and a workflow with redundancy-based service-level fault-tolerance, where each service has only one backup service. Notice again that, even with only one backup service, the reliability increases dramatically.

9.2 Implementations

To achieve some measure of fault-tolerance, there should be at least one backup server running identical copies of the services or there should be another implementation of the services. Also, the workflow should be able to switch to that server/service in the case the primary server/service fails. The most obvious method to implement that is to simply encode the extra service location (e.g., URL, IP number, or DNS name) within the code of the actors, i.e. hard code the location of the alternatives. This way, by using a simple control structure, the actor then tries to invoke the backup services when the primary service fails to respond. We should note that the mechanism of this method is transparent to the user. The only impact that it might have on the performance is a small delay, because the flow may have to wait to confirm timeout of the primary service before it tries the alternative, and the timeout may take a few seconds. The advantage of using hard-coded alternatives is that this provides very simple basic fault-tolerance. It makes the workflow more resilient with respect to simple failures of known services. But there are also several disadvantages. The most significant one is that when the location of the services changes, or when we want to add more backup services, then we need to change the code and recompile the actors and redistribute them. This complicates matters. One options it to provide an interface for the end user to input the location of the primary and alternative services. That also is not very efficient because it requires that the end user know the location of different servers. Thus, the need for a more versatile solution emerges. The next two solutions address that issue.

Instead of simply hard coding the location of different services, one could use a less intrusive and more dynamic approach. This approach stores all relevant information about the services in a file (the choice nowadays is an XML file). That XML file is then kept on a separate server to be accessed by the actors at run-time. An actor parses file, and stores the results in an internal data structure. An alternative is to provide the alternate service locations as actor parameters – this is what some Kepler actors do today. When the time comes to invoke a service, system retrieves

the relevant information and invokes the service at the primary location. When this succeeds, system continues executing the rest of the workflow. If the invocation fails, system can try an alternative service from the list. This approach presents a more versatile solution than the previous one since it does not require that the location of services be hard coded. Thus it allows adding and modifying the location of the services without modifying the source code. The user would still need to know the location of the XML file, and which alternative service is to be invoked. Both can be passed as actor parameters.

A third approach is to use UDDI [20] based registry of services. Using this method, the end user needs only to supply the name of the service the user is trying to invoke, and the actor then searches the repository to find matching services. Note that there is an underlying assumption that a proper set of compatible keywords (and ontologies) exists for all registered services. That service is then invoked. If this is successful, the actor continues its execution. If invocation fails, another search can occur to find yet another alternative service. The advantage of using this method over the previous ones is that the repository can regularly check whether a service is still online by using the heartbeat approach. This feature isn't supported in the previous method, i.e. in the case one of the services isn't available, the XML file or parameter entry has to be manually corrected, or it would keep on returning the location of the unavailable service. Another advantage is the flexibility in adding or modifying existing services. In order to add an additional service, all that needs to be done is to add the service to the repository through an easy-to-use web interface. Thus no configuration files or parameters need to be modified.

Figure 13 shows a screenshot of the parameters required for a fault tolerant web service actor. Only a keyword, method name, username and password (if username and password are used) are required. The actor automatically extracts the namespace and location URL from the retrieved services lists.

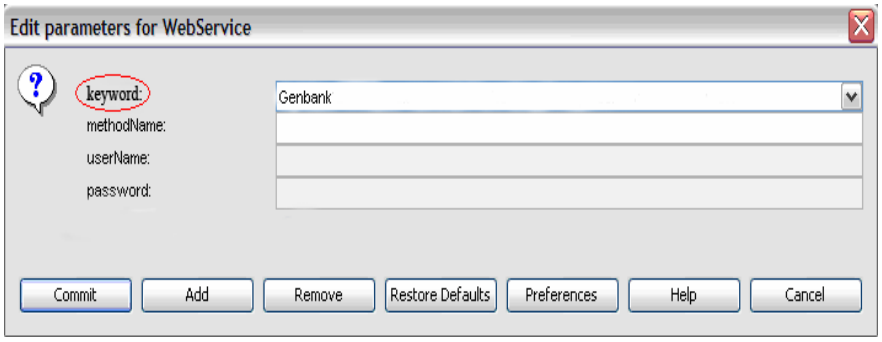

Fig. 13. A fail-over Kepler web-service actor.

It is possible that one of the services returned in the search isn't what we're looking for. This can be avoided by providing a more complex search phrase, for example in the case of the Genbank service illustration in Figure 13, the user could enter "SDM SPA Genbank" instead of just "Genbank". Another approach is to have a sample input and output, then invoke the service with that sample input and compare the outputs, to make sure that the service in question is indeed what we're

looking for. This approach involves more computation and a comparison step, and thus might cause a slight delay. But it can also be looked at as a way to provide validation, since in the case the outputs didn't match; we can consider that the service in question is either an incorrect one, or is not behaving properly.

9.3 Further Improvements

A natural alternative worth emphasizing in this context is to simply just re-try one or more times services that appear to have failed. Sometimes services do not respond due to oversubscription or network glitches, and re-trying solves the problem. Re-trying a service is a general (and very common) approach that can be used in conjunction with all methods discussed above. An example of that (at the workflow execution level) is the fault-tolerant shell [32]. However, all discussed solutions have some inherent limitations. The most important one is that they do not deal with the case when the web service is operational but is not behaving properly, for example not returning correct results, or where the workflow is in the middle of a conversation with the service when the service fails and the state of the service matters. Solutions discussed above can only handle problems caused by initial unavailability of the services.

A more complex, perhaps voting based scheme may be needed to deal with comparison of results, with semantic differences among alternatives, and with state-recovery. Further improvements need to include validation of the results before proceeding with the rest of the workflow. That can be done, for example, by submitting an appropriately selected sample request and comparing the result with a saved result before submitting the rest of the results. That approach may require sophisticated comparison algorithms since services interfaces may be quire complex, but this may be able to mitigate a correctness failure. Yet, another approach could involve invoking several identical services and comparing their results, then choosing the consensus response. Both methods present a possible solution to the validation issue, but might result in additional processing time, thus delaying workflow execution. They are also not comprehensive. In this context an issue that will need further work is handling of correlated failures. This requires a much more complex model. Interested readers may wish to consult [31, 33], and references therein, for more information on different fault-tolerance and reliability models. Software rejuvenation [27, 28] may be another solution that can be used to provide increased availability and failure-avoidance, but its discussion is beyond the scope of this paper.

10 Summary

As scientific discovery and problem solving becomes more complex and more dependent on high-end information technology, comprehensive end-to-end data and process management solutions are needed to reduce the IT burden on the scientist. A group of technologies, called scientific workflow support frameworks, that do that is maturing and we expect to see an increased use of these solutions. One such technology is the Ptolemy II based environment called Kepler. This paper has discussed some of the issues associated with practical automation of scientific

processes and workflows and has illustrated this with workflows developed using the Kepler framework and tools. The topics covered include general workflow development concepts, the impact of information technology overhead and complexity in this context, the structure of Kepler, and issues related to provenance and fault-tolerance. Open issues for high-end scientific workflow technologies include autonomic behavior (auto-recovery and fault-tolerance), authentication and security management, data and provenance management, run-time data and process monitoring and steering (e.g., via "dashboards"), semantic level verification and validation of workflows during construction and at run-time, and development of appropriate end-user visual and other interfaces (e.g., workflow construction "wizards" and persistent portals).

We expect that scientific workflow support technologies will play a critical role in the world of peta- and exa-scale computing supported research and discovery, and in the next 10 years will become a standard feature of the cyberinfrastructure. A key issue that will need to be resolved in this context will be **interoperability**. While many workflow related standardization efforts are under way, it is currently not trivial to exchange more complex information and service, not to mention workflows themselves, among for example Kepler, Taverna [61] and Windows Workflow Foundation [4] environments. In the future, there will be many scientific workflow support environments in operation – some open source, some not. But, unless there is a relatively widely accepted way of exchanging information and services among the workflows constructed in different environments, different scientific communities may be have difficulties collaborating. To avoid "stove-piping" and impediments to progress that that brings, it is very important that open interfaces be defined now and that the existing and new workflow environments be architected and implemented in the spirit of SOA [50].

11 Acknowledgments

We would like thank our colleagues working on the U.S. Department of Energy (DOE) Scientific Data Management Center project for their support and interest. This work has been supported in part by the DOE SciDAC grants DE-FC02-01ER25484 and DE-FC02-07ER25809, IBM Shared University Program, and StrikeIron Inc. The Kepler and Vistrails projects are also funded by grants from the National Science Foundation.

12 References

1. D. Georgakopoulos, M. Hornick, and A. Sheth, "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure," Distributed and Parallel Databases, Vol. 3(2), April 1995.
2. "Kepler Project" Website, 2006. http://kepler-project.org
3. TRIANA Project, October 2006, http://www.trianacode.org/
4. Windows Workflow Foundation (http://msdn2.microsoft.com/en-us/netframework/aa663328.aspx) http://www.microsoft.com/windowsserversystem/virtualserver/default.mspx

5. 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.
6. B. Ludäscher and C. A. Goble. “Guest Editors: Introduction to the Special Section on Scientific Workflows.” SIGMOD Record, 34(3), 2005.
7. R. Mount et al., Department of Energy, Office of Science report, “Data Management Challenge”. Nov 2004, http://www.er.doe.gov/ascr/Final-report-v26.pdf
8. Altintas, S. Bhagwanani, D. Buttler, S. Chandra, Z. Cheng, M. Coleman, T. Critchlow, A. Gupta, W. Han, L. Liu, B. Ludäscher, C. Pu, R. Moore, A. Shoshani, and M. Vouk, “A Modeling and Execution Environment for Distributed Scientific Workflows”, demonstration track, 15th Intl. Conference on Scientific and Statistical Database Management (SSDBM), Boston, Massachussets, 2003.
9. R.I. Balay, Vouk M.A., Perros H., “Performance of Network-Based Problem-Solving Environments,” Chapter 18, in Enabling Technologies for Computational Science Frameworks, Middleware and Environments, editors Elias N. Houstis, John R. Rice, Efstratios Gallopoulos, Randall Bramley, Hardbound, ISBN 0-7923-7809-1, 2000
10. M.A Vouk., and M.P. Singh, "Quality of Service and Scientific Workflows," in The Quality of Numerical Software: Assessment and Enhancements, editor: R. Boisvert, Chapman & Hall, pp.77-89 , 1997.
11. M.P. Singh, Vouk M.A., “Scientific workflows: scientific computing meets transactional workflows,” Proceedings of the NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions, Univ. Georgia, Athens, GA, USA; 1996, pp.SUPL 28-34.
12. “The Ptolemy II Project” website, 2005. http://ptolemy.eecs.berkeley.edu/ptolemyII/
13. S.E. Elmaghraby, "On generalized activity networks," J. Ind. Eng., Vol. 17, 621-631, 1966.
14. R.L. Dennis, D.W. Byun, J.H. Novak, K.J. Galluppi, C.C. Coats, and M.A. Vouk, "The Next Generation of Integrated Air Quality Modeling: EPA's Models-3," Atmospheric Environment, Vol 30 (12), pp 1925-1938, 1996.
15. J.C. Laprie, and C. Beounes, “Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures”, IEEE Computer Society Press, Volume 23, Issue 7, Pages: 39 – 51, July 1990.
16. D.F. McAllister, and M.A. Vouk, "Software Fault-Tolerance Engineering," Chapter 14 in Handbook of Software Reliability Engineering, McGraw Hill, pp. 567-614, January 1996.
17. ACME Laboratories, “Web Servers Comparison”,, http://www.acme.com/software/thttpd/benchmarks.html, 1998.
18. Iyengar, A.; MacNair, E.; Nguyen, T. , “An analysis of Web server performance”. Global Telecommunications Conference, 1997. GLOBECOM '97., IEEE Volume 3, 3-8 Nov. 1997 Page(s):1943 - 1947 vol.3
19. Lloyd Ian, “Government website failure – Is it so shocking?” March 06, http://www.webstandards.org/2006/03/31/government-web-site-failure-is-it-so-shocking-2/
20. “OASIS UDDI “, OASIS Open website 2005 http://www.uddi.org

21. “StrikeIron Web Services Business Directory”, StrikeIron Inc. 2005. http://www.strikeiron.com
22. “Apache Web Services Project: jUDDI” website. 2005 http://ws.apache.org/juddi/
23. “Soap UDDI Project” website, 2005. http://soapuddi.sourceforge.net/
24. Elias N. Houstis, John R. Rice, Efstratios Gallopoulos, Randall Bramley, “Enabling Technologies for Computational Science Frameworks, Middleware and Environments”, Hardbound, ISBN 0-7923-7809-1, 2000
25. Crnkovic and M. Larsson (editors), Building Reliable Component-Based Software Systems, Artech House Publishers, ISBN 1-58053-327-2, 2002, http://www.idt.mdh.se/cbse-book/
26. Common Component Architecture Forum, http://www.cca-forum.org/, accessed February 2006
27. Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, “Software Rejuvenation: Analysis, Module and Applications”, in Proc. of 25th Symposium on Fault Tolerant Computing, FTCS-25, pages 381–390, Pasadena, California, June 1995.
28. K. Vaidyanathan; Trivedi, K.S. “A comprehensive model for software rejuvenation”. IEEE Transactions on Dependable and Secure Computing, Volume 2, Issue 2, April-June 2005 Page(s):124 - 137
29. S.E. Elmaghraby, Baxter E.I., and Vouk M.A., "An Approach to the Modeling and Analysis of Software Production Processes," Intl. Trans. Operational Res., Vol. 2(1), pp. 117-135, 1995.
30. G. Chin. Jr., Leung, L.R., Schuchardt, K.L., and Gracio, D.K. (2002). New Paradigms in Collaborative Problem Solving Environments for Scientific Computing. In Proceeding of the 2002 International Conference of Intelligent User Interfaces (IUI 2002), (Jan. 13-16, San Francisco, CA). ACM Press, New York.
31. M.A Vouk, "Software Reliability Engineering of Numerical Systems," Chapter 13, in Accuracy and Reliability in Scientific Computing, Editor: Bo Einarsson, ISBN 0-89871-584-9, SIAM, 2005, pp. 205-231 [PDF - Draft]
32. Cooperative Computing Lab at the University of Notre Dame (http://www.cse.nd.edu/~ccl/software/ftsh/)
33. M.R. Lyu (ed.), Software Fault Tolerance, Trends-in-Software Book Series, Wiley, 1994
34. J.D. Musa, [a]Operational Profiles in Software-Reliability Engineering, IEEE Software, vol. 10, no. 2, pp. 14-32, Mar. 1993.
35. M. Vouk, R.L. Klevans, and D.L. Bitzer, "Workflow and End-User Quality of Service Issues in Web-Based Education," IEEE Trans. On Knowledge Engineering, to Vol 11(4), July/August 1999, pp. 673-687.
36. Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure, January 2003, http://www.nsf.gov/od/oci/reports/atkins.pdf
37. Department of Energy, Office of Science, “Data Management Report”. May 2004, http://ultralight.caltech.edu/gaeweb/portal/misc/2005/05DMW/Final-report.pdf
38. Edward A. Lee and Steve Neuendorffer. MoML — A Modeling Markup Language in XML — Version 0.4. Technical report, University of California at Berkeley, March, 2000.

39. International Provenance and Annotation Workshop (IPAW'06), Chicago, Illinois, May 3-5, 2006, http://www.ipaw.info/ipaw06/
40. Simmhan, Y. L., Plale, B., Gannon, D., A survey of data provenance in e-science. In *SIGMOD Rec.* 34(3): 31-36, 2005
41. Altinats, I., Barney O., Jaeger-Frank, E. "Provenance Collection Support in Kepler Scientific Workflow System," Proc. of the IPAW'06, www.ipaw.info/ipaw06/proceedings/CameraReady_s5_2.pdf
42. Foster, I., Voeckler, J., Wilde, M., Zhao, Y., "Chimera, A Virtual Data System for Representing, Querying, and Automating Data Derivation," In Proceedings of the 14th Conference on Scientific and Statistical Database Management, 2002
43. Greenwood, M., Goble, C., Stevens, R., Zhao, J., Addis, M., Marvin, D., Moreau, L., Oinn, T., "Provenance of e-Science Experiments - experience from Bioinformatics," In Proceedings of The UK OST e-Science second All Hands Meeting 2003 (AHM'03)
44. Groth, P., Luck, M., Moreau, L. "A protocol for recording provenance in service-oriented grids," In Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04), 2004
45. Bavoil, L., Callahan, S., Crossno, P., Freire, J., Scheidegger, C., Silva, C., and Vo, H., "Vistrails: Enabling interactive multipleview visualizations." In IEEE Visualization 2005, pages 135–142, 2005
46. Some examples of open source scientific workflow solutions: BioPipe, BizTalk, BPWS4J, DAGMan, GridAnt, Grid Job Handler, GRMS (GridLab Resource Management System), GWFE (Gridbus Workflow Engine), GWES (Grid Workflow Execution Service), IT Innovation Enactment Engine, JIGSA, Kepler, Karajan, OSWorkflow, Pegasus (uses DAGMan), ScyFLOW, SDSC Matrix, SHOP2, Taverna, Triana, wftk, YAWL Engine, WebAndFlo, WFEE, etc. see http://www.gridworkflow.org/snips/gridworkflow/space/Workflow+Engines, http://www.extreme.indiana.edu/swf-survey/
47. Win Bausch, Cesare Pautasso, Reto Schaeppi, Gustavo Alonso, "BioOpera: Cluster-Aware Computing," CLUSTER 2002, pp. 99-106
48. Claus Hagen, Gustavo Alonso, "Flexible Exception Handling in the OPERA Process Support System," ICDCS 1998, pp. 526-533
49. Fabio Casati, Stefano Ceri, Stefano Paraboschi, and Giuseppe Pozzi, "Specification and Implementation of Exceptions in Workflow Management Systems," ACM Transactions on Database Systems 24(3), Sept. 1999
50. Service Oriented Architecture (SOA), Wikipedia, 2006 (http://en.wikipedia.org/wiki/Service-oriented_architecture), also http://www-306.ibm.com/software/solutions/soa/, and references therein.
51. OASIS, http://www.oasis-open.org/ (e.g., BPEL)
52. OWL, http://www.w3.org/TR/owl-features/
53. Web Services standards at http://www.w3.org/TR (e.g., WSDL and similar).
54. KEPLER provenance framework at http://kepler-project.org/Wiki.jsp?page=KeplerProvenanceFramework
55. VisTrails (http://www.vistrails.org)
56. J. Salas, F. Perez, M. Patia-Martinez, R. Jiminez-Peris, "WS-Replication: A Framework for Highly Available Web Services," WWW Conf., Edinburgh, Scotland, May 2006.

57. J. Freire, C. Silva, S. Callahan, E. Santos, C. Scheidegger and H. T. Vo, "Managing Rapidly-Evolving Scientific Workflows," International Provenance and Annotation Workshop (IPAW), LNCS 4145, pages 10-18, 2006. Springer.
58. C. Scheidegger, D. Koop, E. Santos, H. Vo, S. Callahan, J. Freire, and C.Silva. "Tackling the Provenance Challenge One Layer at a Time," submitted to Concurrency And Computation: Practice And Experience. (Special issue on the first Provenance Challenge.)
59. Grid Account Management Architecture (http://grid-devel.sdsc.edu/gridsphere/gridsphere?cid=gama), SDSC, 2005, and Mueller, GEON, 2006 (http://www.geongrid.org/presentations/webcasts/Mueller_GAMA_GEON_May06.ppt)
60. LDAP, SEEK (http://seek.ecoinformatics.org/Wiki.jsp?page=CertificateAuthorityDesign)
61. Taverna Project Website (http://taverna.sourceforge.net/)
62. SciRUN (http://software.sci.utah.edu/scirun.html/)
63. Ensight (http://www.ensight.com/home/index.php)
64. Virtual Computing Laboratory (VCL) - http://vcl.ncsu.edu