Future Generation Computer Systems 25 (2009) 528–540

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Workflows and e-Science: An overview of workflow system features and capabilities

Ewa Deelman [a], Dennis Gannon [b], Matthew Shields [c], Ian Taylor [c,d,*]

[a] Information Sciences Institute, University of Southern California, USA
[b] Department of Computer Science, Indiana University, USA
[c] School of Computer Science, Cardiff University, UK
[d] The Center for Computation and Technology, LSU, USA

ARTICLE INFO

Article history:
Received 1 December 2007
Received in revised form
8 May 2008
Accepted 25 June 2008
Available online 10 July 2008

Keywords:
Scientific workflow
Grid computing
Computation
Web services
Distributed computing
Distributed systems
Cyberinfrastructure
Automation of scientific processes
e-Science

ABSTRACT

Scientific workflow systems have become a necessary tool for many applications, enabling the composition and execution of complex analysis on distributed resources. Today there are many workflow systems, often with overlapping functionality. A key issue for potential users of workflow systems is the need to be able to compare the capabilities of the various available tools. There can be confusion about system functionality and the tools are often selected without a proper functional analysis. In this paper we extract a taxonomy of features from the way scientists make use of existing workflow systems and we illustrate this feature set by providing some examples taken from existing workflow systems. The taxonomy provides end users with a mechanism by which they can assess the suitability of workflow in general and how they might use these features to make an informed choice about which workflow system would be a good choice for their particular application.

1. Introduction

Over the past two decades, we have seen a revolution in the way science and engineering are conducted. Computation has become an established "third branch" of science alongside theory and experiment. Supercomputers simulate large, physically complex systems modelled by partial differential equations. Computational tools are adopted in applications that involve complex data analysis and visualization steps. A typical experimental scenario is a repetitive cycle of moving data to a supercomputer for analysis or simulation, launching the computations and managing the storage of the output results. Scientific workflow systems aim at automating this cycle in a way to make it easier for scientists to focus on their research and not computation management. At the same time, the business community also addressed the automation of their business logic and tools and commercial computer companies soon followed to support this process. What emerged was a primitive science of workflow design. *Workflow Orchestration* refers to the activity of defining the sequence of tasks needed to manage a business or computational science or engineering process. A *Workflow* is a template for such an orchestration. A *Workflow Instance* is a specific instantiation of a workflow for a particular problem and includes the definition of input data. Within the scientific and engineering community these terms have a slightly broader meaning and here in this paper, we focus on the classification of such by considering four broad areas that describe the various aspects of the workflow life cycle (composition, mapping, execution and provenance). These four areas are then further subdivided to attempt to build a classification scheme for workflow systems through the use of a rich feature set that outlines a taxonomy to provide the basis for a categorisation axis for various workflow systems within the scientific workflow community.

There have been other publications that have compared a subset of workflow systems for scientific applications [1] but such classifications potentially suffer from the ever-changing world of distributed computing and the evolution of new workflow solutions that are appearing at a rather frequent rate. Other workflow taxonomy papers [2] promote specific aspects desired for workflow systems, such as the need for adaptive workflow management. Therefore, rather than focusing on a specific aspect

* Corresponding author.
E-mail address: Ian.J.Taylor@cs.cardiff.ac.uk (I. Taylor).

0167-739X/$ – see front matter
doi:10.1016/j.future.2008.06.012

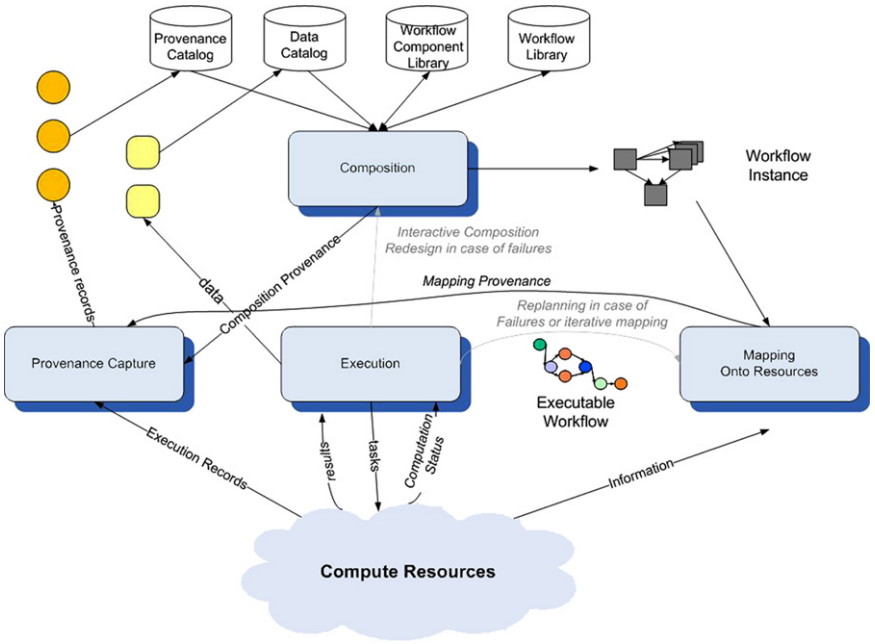

Fig. 1. The workflow lifecycle: composition, mapping, execution, and provenance capture.

or comparing every workflow solution that exists today, we focus here on the scientist's or distributed-application developer's view and attempt to extract a feature set that encapsulates the functionality exposed by various workflow systems in an attempt to create a generalised taxonomy for the field as a whole rather than comparing specific solutions. Therefore, the systems we include in this paper only constitute a subset of the many workflow systems that exist within the research and commercial communities and provide an illustrative means for the justification of our feature set.

It is hoped that such a taxonomy of features will be useful not only to scientists or application-developers when considering which features might be important for their problem domain but also to the workflow community as a whole in order to provide some basis for classification of systems in the future and possible interoperability. It is as inconceivable at this stage to see a one-size-fits-all workflow system that encompasses all of the desirable features in one solution as it is to ask for one programming language to be best for all problems. Therefore any decision about the choice of one workflow system or systems used must be informed in order to reduce the development time of applications as a whole in this area.

2. Workflow capabilities and the workflow lifecycle

At one level a workflow is a high-level specification of a set of tasks and the dependencies between them that must be satisfied in order to accomplish a specific goal. For example, a data analysis protocol consisting of a sequence of pre-processing, analysis, simulation and post-processing steps is a typical workflow scenario in e-Science applications. At the level of representation and execution, a workflow is a computer program and it can be expressed in any modern programming language. However, the task of writing a computer program in Perl or Java or Python to orchestrate a set of tasks on a wide-area distributed system goes well beyond the programming skills or patience of most scientific users. The goal of e-Science workflow systems [3] is to provide a specialized programming environment to simplify the programming effort required by scientists to orchestrate a computational science experiment.

In general we can classify four different phases of the workflow lifecycle (as seen in Fig. 1):

1. **Composition, Representation and Data Model:** the composition of the workflow (abstract or executable) through a number of different means e.g. text, graphical, semantic.
2. **Mapping:** Involves the mapping from the (abstract) workflow to the underlying resources.
3. **Execution:** Enactment of the mapped workflow on the underlying resources.
4. **Metadata and Provenance:** The recording of metadata and provenance information during the various stages of the workflow lifecycle.

During workflow composition the user creates a workflow either from scratch or by modifying a previously designed workflow. The user can rely on workflow component and data catalogs. The workflow composition process can be iterative, where portions of the workflow need to be executed before subsequent parts of the workflow are designed. Once the workflow is defined, all, or portions of the workflow can be sent for mapping and then execution. During that phase various optimizations and scheduling decisions can be made. Finally, the data and all associated metadata and provenance information are recorded and placed in a variety of registries which can then be accessed to design a new workflow. Even though we delineate data recording as a separate phase of the workflow lifecycle, this activity can and often is part of the workflow execution.

In the following four sections, we examine these four areas in detail by extracting a feature set within each category to define the common aspects of functionality that are inherent across workflow systems in general.

3. Composition, representation and execution models

3.1. Workflow composition

The composition of workflows is an important step in the workflow process. It allows a workflow user or programmer to specify the steps and dependencies in either a concrete or abstract fashion that represent the desired overall analysis. There are a number of mechanisms for specifying such flows or dependencies and the resulting graph can be stored in a

number of different formats, described in the next section. In some cases the workflows are described directly by indicating both computational steps and the data that flows through them. In other cases, the workflow composition goes through two phases, where first a workflow template (high-level workflow description, usually without specific data) is formed and then the template is instantiated with actual data. Examples of the latter approach are present in Wings [8] and in VLAM-G [94]. Templates can also be used to support workflow sharing and reuse. In this subsection, we outline existing methods for editing or composing workflows that exist in a number of workflow systems today. We categorise the composition methods into three broad categories: textual, graphical and mechanism-based semantic models.

3.1.1. Textual workflow editing

Many workflow systems use a particular workflow language or representation (BPEL [4], SCUFL [5], DAGMan [6], DAX [7]), which has a specification that can be composed by hand using a plain text editor. While this has worked well for users of some systems, such as DAGMan, there are many examples where the task of writing the textual workflow program by hand can be extremely difficult or error-prone. This is particularly true if the workflow language has poor support for standard programming control constructs. For example, workflows for parameter sweeps can be pretty simple before parallelization but then become extremely complex after the parallelization has taken place, where the same algorithm is applied to multiple data sets and each data set is represented by one branch of the overall workflow graph. For these purposes, a scientist typically uses a script in a high-level language like Python or Ruby which is designed for expressing complex control and using it to generate the lower-level workflow primitives.

For example, Pegasus [7] 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 type of scripting language, or with the use of semantic technologies such as Wings [8]. In some cases, scientific applications want to provide users with an interface, which is only in the form of a metadata query. For example, in astronomy, users often do not want to know the details of the underlying system, instead they want to retrieve images of an area of the sky of interest to them. In such cases Pegasus is usually integrated into a portal environment where the user is presented with a Web form to fill in the desired metadata attributes. Inside the portal, the workflow instance is generated automatically based on the user's input and is given to Pegasus for mapping and then to DAGMan for execution. Examples of this approach can be seen in the Montage project [9,10] (an astronomy application), the Telescience portal [11] (a neuroscience application), and the Earthworks portal [12] (an earthquake science application). In all these applications, Pegasus and DAGMan are being used to run the application workflows on a national scale infrastructure such as the TeraGrid.

3.1.2. Graphical workflow editing

To simplify the life of scientific users, most e-Science workflow systems provide a graphical tool for composing workflows. While graphical programming has been a subject of experiment for many years, it has not proven to be a substitute for traditional text based programming for general purpose use. However it has achieved considerable success in the domains where the primary task is one of composition and orchestration. For example, systems such as AVS [13] and SciRun [14] allow users to design complex graphics applications by composing graphical filers and rendering modules. e-Science workflow systems such as Kepler [15], Triana [16], and Vistrails [17] have sophisticated graphical composition tools for building workflow around the idea of composing graphs where the nodes represent tasks and edges represent dependencies.

BPEL (Business Process Execution Language) [4] is the de facto standard for Web-service-based workflows with a number of implementations from Microsoft, IBM, BEA and Oracle as well as open source organizations. BPEL is a Turing complete programming language expressed in XML. Very few people write BPEL code by hand and a host of graphical tools exist to allow users to compose BPEL workflows. Several such tools exist for e-Science applications. For example, XBaya [18] is used in the LEAD project [19] and in some bioinformatics applications. XBaya includes a compiler that translates a graphical representation into several different textual forms. For example, in addition to BPEL, it can compile a graphical workflow representation into a Python program. Eclipse BPEL Designer [20] uses primarily visual modelling. Composition of invocations of service partner operations and control-flow expressions are specified through various BPEL primitives. The BPEL knowledge editors are mainly GUIs but there is some preliminary developments on Web-based modelling environments.

Graphical renderings of a workflow are easy for small workflows with fewer than a few dozen tasks. However many e-Science workflows are far more complex. Consequently most graphical tools allow some form of graphical nesting based on sub-workflow hierarchies. Another source of graphical complexity involves expressing "for-each" concurrency in a workflow. However, this problem can be addressed by providing specialized control primitives in the graphical vocabulary. We return to this topic in the following section.

Workflow systems can be generally classified into two broad categories: Task-based or service-based. Task-based systems (e.g. Pegasus) generally focus on the mapping and execution capabilities and leave the higher-level composition tasks to other tools, even employing the use of semantics, as described in Section 3.1.4. On-going work with both Kepler and Triana is also being undertaken with Pegasus to provide graphical choreography interfaces to the system. However, whereas task-level workflow systems focus on the resource-level functionality and fault-tolerance, service-level systems generally provide interfaces to certain classes of services for management and composition. One important factor therefore to their take up is the availability of current 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.

3.1.3. Worfklow components

One of Taverna's key values for example is the availability of services to the core system, current figures estimate this to be around 3500 mainly concentrated in the bioinformatics problem domain. Taverna has also began to share workflows through the myExperiment project [21] in order to make such workflows available to the community as a whole. Taverna has a GUI-based desktop application that uses semantic annotations associated with services. It employs the use of semantic-enabled helper functions which will be made available in the next public release of the software. Developers can incorporate new services through simple means 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 (i.e. to support grouping). Services within the pre-existing workflow can also be instantiated individually within the current workflow and a developer can create user-defined perspectives that allow a panel of pre-existing components to be specified. Within Taverna, there is also support for the configuration of the appearance of the graphical representation of the current workflow, so that, for example, a workflow can be suppressed to give higher level views (e.g. to remove the data translation (shim) services).

One of the most powerful aspects of Triana on the other hand 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's editing capabilities include: multi-level grouping for simplifying workflows, cut/copy/paste/undo, the 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 [22] 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 [23], image processing, text editing, for creating retinopathy workflows[1] and even data mining based on configurable Web Services [24,25] to aid in the composition process.

The CoG Kit's Karajan [26] includes mechanisms for the orchestration of tasks, parallel blocks, futures (a place holder for a data product that has not been generated yet), loops, functional programming language, hierarchical graphs and parallel control structures, which are a particular focus of the system. It uses either a scripting language, GridAnt, a simple graphical editor to create the workflows.

Kepler inherits a GUI for workflow composition and editing called Vergil from Ptolemy II with some modifications and extensions to make it more appropriate for scientific workflow composition. In this workflow editor, users can discover components called actors; computational controllers called directors; and data sets, discoverable through remote catalogs; then drag them to the canvass and connect them via ports. Complex sub-workflows can be aggregated into so-called composite actors. Like atomic actors, they have well-defined input and output ports as well as parameters. Parameters can be seen as special input ports that are usually, but not always, kept fixed during a workflow execution. Like Triana and Taverna, Kepler supports the standard composition features such as copy, paste, etc. Kepler has greater flexibility when it comes to representing ports for streaming data from a database type source, each column in a table can be exposed as a different port, enabling a tuple or record to be broken up into individual fields; alternatively the whole tuple can be sent as a single data object, typically what happens in Triana; or thirdly a set of records can be sent as an array object. This functionality can be reproduced in other systems but normally requires specific components rather than the system itself to perform it.

3.1.4. Semantic composition

Scientific workflows may consist of a number of execution components, where each component may run parameter sweep applications or similarly complex computations that involve dividing the data sets into smaller ones in order to support concurrent processing. Within such scenarios, the entire workflow can quickly run into several thousands of jobs for execution, especially in systems which do not support abstract workflow definitions. In order to generate such workflows scientists can create ad hoc scripts that convert the iterative nature of sets of jobs into workflow variants but this can get rather complex since data needs to be moved to the correct location, executables need to be placed and so on. Such an ad hoc approach therefore can run into problems when scaling to larger problem domains or translating to different problems.

Some scientists [27] have conducted research into providing fully automated workflow generation using artificial intelligence planning techniques for assisted workflow composition [28,29]. This is achieved through combining semantic representations of workflow components with formal properties of correct workflows. The work has been motivated through working with two application domains: physics-based seismic hazard analysis [30] and data-intensive natural language processing [31].

Wings [8] 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 data flow and that 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.

Other projects have used similar techniques in different domains to support workflow composition through planning and automated reasoning [32–34] and semantic representations [35]. As workflow representations become more declarative and expressive, they enable significant improvements in automation and assistance for workflow composition and in general for managing and automating complex scientific processes.

3.2. Workflow representation

Workflow representation can take a number of forms, at its most abstract level all workflows are merely a series of functional units, whether they are components, tasks, jobs or services, and the dependencies between them which define the order in which the units must be executed. There are a number of models from mathematics and computer science upon which the various workflow representation languages are based. The models include the ubiquitous directed graph variants, Petri nets [36], Unified Modelling Language (UML) [37], Business Process Modelling Notation (BPMN) [38] as well as several lesser known process modelling tools. The group of workflow languages based on these models changes very rapidly as new projects start and old ones become redundant so here we only provide a snapshot of the current state. The actual representation used should usually matter little to the average user of an individual workflow system until it comes to the matter of interoperability between workflow systems discussed in Section 7.

3.3. Directed graphs

The most common representation is the directed graph, either acyclic (DAG) or the less used cyclic (DCG), which allow loops. Although many workflow systems use graphs, the actual representation of the graph in a language or format that can be used by the workflow tool varies, most systems use an XML based representation. In some cases, BPEL, the business domain workflow language for Web services, is being used directly for scientific workflows, examples include OMII-BPEL [39], the e-HTPX project for high throughput protein crystallography [40] and GPEL [41] used in LEAD. Condor's DAGMan is another commonly used DAG format for specifying workflows of dependent jobs, it is used by many users directly and is the underlying execution workflow representation for Pegasus. Pegasus uses a different representation, the DAX (DAG in XML), for its abstract workflow definition, which is the input to the Pegasus' mapping process. The Taverna workflow environment concentrates on the workflow composition of Web services and uses an XML-based DAG format

[1] UK STFC TRIACS project ST/F002033/1.

called *SCUFLE*. Triana uses a DCG format of its own although it is able to import and export from different formats including DAGMan and the Virtual Data Language [42]. Sedna/Eclipse BPEL Designer uses an XML-based representation of BPEL workflows and syntactic aspects are defined via XML Schemas. However, since BPEL is an event-driven workflow language based on messages, developers can use simple state machine representation to capture state at any point in time: messages received, sent, waiting for and BPEL process activity. Therefore, a snapshot can be made of the state of BPEL process. Kepler uses yet another directed graph representation MOML (MOdel Markup Language), as with Kepler's GUI, MOML is inherited from Ptolemy II. MOML is flexible but was not designed with scientific workflow applications in mind so Kepler has developed a new workflow format called KAR (Kepler Archives) that uses MOML but also incorporates JAR/TAR archive ideas to make workflows more self-contained and easier to share. The idea of self contained workflow archives is currently being addressed by both the Taverna and Triana projects, particularly in response to the myExperiment project and the ability to be able to share workflows in a portal environment.

3.4. Petri nets

Petri nets are a specialised class of directed graph and models based on these have been used by several workflow systems. Petri Nets are a formalism for describing distributed processes by extending state machines with concurrency, they graphically depict the structure of a distributed system as a directed bipartite graph with annotations. As such, a Petri Net has place nodes, transition nodes, and directed arcs connecting places with transitions. They are the basis for the Grid Workflow Description Language (GWorkflowDL) [43] the workflow language for K-Wf Grid project [44], and the Grid Flow Description Language (GFDL) [45] from the Grid-Flow project [46].

3.5. UML

The Unified Modelling Language (UML) is a standardised language for the modelling of object-oriented software. One of its aspects is an activity diagram which can be used to model dependencies between different activities and hence can be used as a model for workflow. Askalon [47] uses UML activity diagrams for the graphical representation of its workflow applications. These are then translated into the Abstract Grid Workflow Language (AGWL) [48] an XML representation for execution and storage.

3.6. Workflow execution control models

Most, if not all, workflow models fall into one of two classes: control or data flows. The two classes are similar in that they specify the interaction between individual activities within the group that comprise the workflow, but they differ in their methods of implementing that interaction.

In control-driven workflows, or control flows, the connections between the activities in a workflow represent a transfer of control from the preceding task to the one that follows. This includes control structures such as sequences, conditionals, and iterations. Data-driven workflows, or data flows, are designed to support data-driven applications. The dependencies represent the flow of data between workflow activities from data producer to data consumer.

There is also a growing set of hybrid workflow representations based on a combination of control and data flows. These hybrids use both types of dependencies as appropriate but are normally biased toward either data flow or control flow, using the other to better handle certain conditions. For instance, in a data-flow system such as Triana, there are situations where a downstream task needs to be activated but the upstream task produces no output. In this case, a *trigger* is used to switch the flow of control. In other hybrid control-flow systems, such as the CoG Kit's Karajan workflow, data dependencies can be represented by a *future*, the concept of data that has not yet been produced, which can block the control flow with a data-flow dependency.

A different, more flexible approach is taken in Kepler, where the semantics of the workflow computation is a pluggable component. The user designs the workflow using various workflow components, known as *actors* and indicating the dependencies between them. However the semantics of the computation model are then indicated by the user based on the *director*, a specific model of computation, that a user chooses. In that way, Kepler supports a variety of semantics such as data flow, control flow, continuous time where the dependencies represent the the value of a continuous time signal at some point in time, and discrete event, where the workflow components communicate through a queue of events in time. Kepler can also nest the models, for example at the top-level workflow there may be a process network (PN) director for loosely coupled, independent processes (possibly on different machines) or independent threads (proxying for processes on different machines), while at deeper levels inside of composite actors, more fine-grained, local computations might be described using an SDF director, which executes in a single thread.

3.7. Control flow model

Most control flow languages provide support not only for simple flows of control between components or services in the workflow but also for more complex control interactions such as loops and conditionals. Sometimes this support is implicit, as is the case with Petri Nets, and sometimes explicit, as in Karajan.

Users of workflow systems will often want more than the simple control constructs available to them. The ability to branch workflow based on conditions and loop over subsections of the workflow repeatedly is important for all but the simplest of applications. The argument is not whether these facilities should exist but how to represent them in the workflow language and to what degree the language should support them. For instance, is a single simple loop construct enough, or should the language support all loop types? (i.e. *while*, *for … next*, *repeat … until*.) In the case of conditional behaviour, the problem is determining whether the incoming value and the conditional value are equivalent. XBaya supports a set of control primitives including *for-each*, *conditional*, *while* and *exception*. These control constructs are overlays on the dataflow graph to simplify the expression of the workflow. For example, a *for-each* construct can be used to encode a fan-out of a data flow graph where the degree of fan-out is not known until runtime. An exception construct is required when the workflow designer needs to express a sub-workflow alternative path when a part of the flow may be subject to potential, known runtime failures. BPEL has control structures including branching and looping built into the language but only for pre-defined fairly simple data types. For simple cases where we are comparing integers or simple strings, checking the condition is straightforward and unambiguous. The problem appears when we have to compare complex, structured scientific data in scientific workflows. This type of data often needs domain-specific knowledge in order to perform comparisons. Consequently, the evaluation of the comparison must be accomplished by an external service or agent, or a separate component, as part of the workflow execution.

3.8. Data-flow models

Most data flow representations are very simple in nature, and unlike their control flow counterparts, contain nothing apart from component or service descriptions and the data dependencies between them; control constructs such as loops are generally not included. In Triana's workflow language, there are no control constructs at all; the dependencies between tasks are data dependencies, ensuring the data producer has finished before the consumer may start. Looping and conditional behaviour is performed through the use of specific components; a branch component with two or more output connections will output data on different connections, depending upon some condition. Loops are handled by making a circular connection in the workflow and having a conditional component break the loop upon a finishing condition, outputting to continue normal workflow execution. The benefit of both of these solutions to control behaviour in data flows is that the language representations remain simple. The downside is that the potential for running the workflow on different systems is reduced since the other system must have access not only to the workflow but to the components or services that perform the control operations.

3.8.1. Support for data collections

Although most data flow systems support the notion of single data object, some systems also introduce the notion of data collections as first-class entities in the workflow. The computational units in the workflow then operate not on single files for example, but on entire file collections. Examples of such systems are Askalon [49] and Wings [8], which also supports the notion of nested collections and collections of processing components. Askalon supports mapping a portion of a data set to an activity, independently distributing multiple data sets, not necessarily with the same number of data elements, onto loop iterations. COMAD [50], Collection Oriented Modelling and Design extensions to Kepler support both nested data collections and collection token streams, where data streams can be nested using a technique similar to SAX-based parsing of XML documents using paired open and closing tokens to delineate the streams. At the user level there is essentially only one input and one output port per actor; but actors can be configured to pick up only relevant parts of the "fat data stream".

It is clear that both control and data flow techniques are needed for scientific workflow languages. Limiting the language to one or the other, limits the usefulness of the tools built to use the language. It is also clear that constantly extending the language to include every programming construct will bloat the language and increase the complexity of the engines needed to execute it. Simple hybrid models with limited control constructs and support for data flow appear to stand the best chance of being interoperable with the most tools and frameworks but still contain enough functionality to be able to represent real scientific applications. However, inevitably there will not be a single model suitable for all situations.

4. Mapping workflows to resources

Workflow mapping refers to the process of generating an executable workflow based on a resource-independent workflow description, frequently called an *abstract workflow* or *worfklow instance*. In some case the user performs the mapping directly by selecting the appropriate resources. In other cases, the workflow system performs the mapping. In the latter case, users are allowed to design workflows at a level of abstraction above that of the target execution environment, which uses low-level commands and detailed resource management constructs.

Depending on the underlying execution model that 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 issues [51] when performing the mapping. In the case of workflows composed of stand-alone applications, the mapping issue not only involves finding the necessary resources to execute the computations and perform various optimizations but may also include modifications to the original workflow.

4.1. Resource definition and discovery

Workflow systems have different meanings for the concept of resource. In many systems, a resource is a service that can perform one of the tasks in the workflow definition. This service may be referred to as a REST [52] or WSDL-based Web service [53], or it may be a refer to a computational agent. A resource may even be a person. For example, workflows may require a human step, such as verification by a scientist that the workflow is proceeding correctly or it may require an administrator's approval prior to proceeding to the next step. In other cases a resource may be a computer upon which a required application is deployed or data archival system for storing results or containing required data sets.

In the case of Web service resources, most workflow systems use a registry which contains descriptions of the services. As a new service is added to the application collection, its description is added to the registry. Public Web service discovery search engines also exist and these can be useful in posting collections of Web services that are important for specific domains. For example there are a large number of public Web services for bioinformatics applications that are used by the Taverna system (Section 3.1.3).

When the resources are computers and data systems, the resource discovery is typically done through a Grid information systems [54], which can list all the computers available to a virtual organization [55], their current load and status. As described below, this can be used by the mapping systems at runtime to select the best resource.

4.2. User-defined workflow mapping

Among the workflow systems that rely on the user to make the choice of resources or services are: Kepler, Sedna, Taverna, and VisTrails [17]. 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. Wings is a workflow composition tool which interfaces to the Pegasus workflow system and relies on its capabilities to perform the mapping of the Wings-generated abstract workflow onto the distributed resources.

4.3. Workflow mapping using an internal scheduler or external broker

Today, P-GRADE [56] can support both with task- and service-based workflows. The original version of the P-GRADE workflow engine that is based on Condor DAGMan supports only task-based workflows. The GEMLCA [57] extension of P-GRADE can support workflows where tasks and services can be applied in a mixed way within the same workflow. This extension of P-GRADE still uses DAGMan to control the node dependencies of the workflow graph but the task submission mechanism is extended with the services invocation mechanism of GEMLCA. When designing P-GRADE/GEMLCA workflows, the user is able to indicate that tasks/services can be scheduled by a broker. P-GRADE

interfaces to three different brokers: GTBroker [58] for Globus-based deployments, the LHC-broker [59] for LHC-based grids, and the gLite-broker [60] for gLite-based grids. These brokers can be used in a mixed way within the same workflow and hence different nodes of a P-GRADE workflow can simultaneously be executed on several grids [61]. When the user chooses a broker as a means of scheduling a job in the workflow, resource requirements for the job can be provided as well. An experimental enhancement of P-GRADE is combined with MOTEUR [62] by relying on two different execution engines. For task-based workflows, it interfaces to DAGMan, for services-based workflows, it uses MOTEUR. MOTEUR is able to determine during execution the data parallelism present in workflow and exploit it if the necessary resources are available.

Triana is able to interface to a variety of execution environments using the GAT (Grid Application Toolkit) [63] for task-based workflows and the GAP (Grid Application Prototype) [64] for service-based workflows. In the case of service-based workflow a user can provide the information about the services to invoke (or locate them via a repository) or a user can create a workflow and then map part of the workflow (using a group) 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 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 [65] 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 for example) 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. WS-RF [66], Web and P2P services using built-in deployment services for each binding.

Karajan [67] supports dynamic binding of tasks to resources. When defining a workflow, the user can specify the tasks at an abstract or concrete level, where a single workflow can be composed of a mix of tasks at different levels of abstraction. The concrete tasks are executed directly by an engine, whereas abstract tasks are sent at runtime to a scheduler for mapping onto a resource. Karajan supports pluggable schedulers and provides a simple built-in implementation. Karajan also supports user-defined task clustering, where the cluster is sent to a single resource for execution.

Similarly to Karajan, workflows specified in DAGMan can be a mixture of concrete and abstract tasks. When DAGMan is interfaced to a Condor task execution system [68] the abstract tasks are matched dynamically to the Condor resources. The selection is done by the Condor matchmaker [69], which matches the requirements of an abstract task specified in a *classad* with the resource preferences published in their *classads*.

In the case of workflows based on BPEL, resource selection can be explicit, i.e. specific service instances are selected prior to workflow enactment. However, it is often the case that the abstract workflow is bound to abstract WSDL descriptions. The process of selecting resources can be left to an external agent which matches the abstract WSDL documents to concrete instances at runtime. In the LEAD project, the science applications are deployed on a variety of remote supercomputers. Each remote application deployment is managed by a virtual Web service. The resource broker examines the execution queues on the remote resources and selects the appropriate resource and realizes the corresponding virtual Web service and passes the service invocation from the BPEL engine. As described later, this has an additional advantage of allowing the broker to monitor execution and to reschedule the application invocation on a different resource in case of failure.

4.4. Workflow optimizations

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 can be 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 four basic scheduling algorithms [70]: HEFT [71], min–min, round-robin, and random. 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 [7]. Before the workflow mapping, the original workflow can be partitioned into any number of sub-workflows. The granularity of the partitioning sets the mapping horizon for the workflow. For example if we have only one task per partition, then this is equivalent to just-in-time scheduling. On the other end of the spectrum, having the entire workflow in one partitions results in full-ahead planning. 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 [72]. This often results in a reduced workflow data storage footprint. Finally, Pegasus can also perform task clustering, treating a set of tasks as one for the purposed 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 granularity computational workflows. Pegasus has also been used in conjunction with resource provisioning techniques to improve the overall workflow performance [30,73,74].

The Askalon system, designed to support task-level workflows, has a rich environment for mapping workflows onto resources. It not only does the resource assignment but can also automatically provision the resources ahead of the workflow execution. Askalon contains two major components responsible for workflow scheduling: the scheduler and the resource management system (GridARM). GridARM serves as a data repository which provides the scheduler with all the information needed for scheduling, including the available resources and the applications deployed on the Grid. Apart from the basic functionalities, GridARM can also provide more advanced resource and application discovery techniques based on quality-of-service matching, and it can guarantee advance reservation of resources. The scheduler uses the information obtained from GridARM in order to perform the mapping of high level tasks specified in AGWL. In order to support the scheduler, Askalon has developed a performance analysis and prediction system which can estimate the runtime of the workflow tasks as well as data transfer times of data between tasks. The scheduler makes full-graph scheduling of scientific workflows, using one of the implemented scheduling algorithms. Currently, these include the HEFT algorithm, an advance reservation-based algorithm based on HEFT [75], and a general-purpose bi-criteria scheduling algorithm. As the Scheduler maps DAGs and the AGWL workflows may include loops, parallel loops, and conditional constructs, a preliminary workflow conversion is performed before the actual scheduling can start. Among the implemented workflow transformations are techniques employed in parallel compilers and include: branch

prediction, parallel loop unrolling, and sequential loop elimination. If the choices made during the transformation (such as branch prediction) were erroneous, the workflow is adjusted and rescheduled at runtime. Once the DAG is created, it is mapped onto the available resources based on a scheduling algorithm.

5. Execution

This section examines the execution of a workflow through the use of an execution engine or enactment subsystem. We cover three different aspects: the execution model, the fault tolerance mechanisms and the ability to adapt workflows (or sub-workflows) based on previous analysis steps in the cycle.

5.1. Execution model

As discussed in Section 3.1, workflow systems generally deal with services or jobs (or combinations or both). In this section we outline the models used by some workflow systems to illustrate the various execution scenarios that are currently available in systems.

Pegasus can map workflows onto a variety of target resources such as those managed by PBS [76], LSF [77], Condor [78], and individual machines. Authentication to remote resources is done via GSI [79] and Pegasus uses the DAGMan workflow engine for the execution of jobs. DAGMan interfaces to a local Condor queue managed by the *schedd* [80]. DAGMan uses the sched's API and logs to submit, query, and manipulate jobs, and does not directly interact with jobs independently. DAGMan can also uses Condor's Grid abilities (Condor-G) to submit 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.

Askalon supports workflow composition and modelling using the Unified Modelling Language (UML) standard and provides an XML-based Abstract Grid Workflow Language (AGWL) for application developers to use. The AGWL is given to a WSRF-based runtime system for scheduling and execution. Askalon contains a resource manager (GridARM based on the Globus tools [81]) that provides resource discovery, advanced reservation and virtual organization-wide authorization along with a dynamic registration framework for activity types and activity deployments.

The Java CoG Kit Karajan workflow framework can support hierarchical workflows based on DAGs with control structures and parallel constructs, it makes use of underlying Grid tools such as Globus GRAM [82] for the actual job submission. Workflows can be visualized and tracked by an engine and modified at runtime through interaction with a workflow repository or schedulers for dynamic allocation of resources to tasks. It has been demonstrated to scale to hundreds of thousands of jobs due to its efficient scalability-oriented threading mechanisms.

Triana supports job level execution through integration with the GridLab GAT, which can make use of GRMS, GRAM or Condor for the actual job submission; it also supports service-level execution through the GAP bindings to Web, WS-RF and P2P Services; and still contains an internal run-time execution engine for the local components written in Java or C. Kepler similarly supports Web services and Grid jobs through specific "actors" and local components through its own run-time engine.

5.2. Fault tolerance

Fault tolerance again can be specific to the types of workflows being executed. At the job level, a number of mechanisms can be used at the operating system level for saving the state of an execution and resuming after a failure. Condor, for example, takes this approach. Other mechanisms can be employed, such as application-level check-pointing in order to migrate an execution to an other machine that may be more adept to executing the particular code. The Cactus worm [83] is a good illustration of this approach. For service-based systems, fault tolerance might involve retrying a service upon a time out or discovering another equivalent service that may be running elsewhere in order to continue with the execution. More typically in most of the current workflow systems fault tolerance, exception handling and recovery are ad hoc tasks undertaken by individual workflow designers rather than being part of the systems themselves.

Triana employs a passive approach by informing the user when a failure has occurred. The workflow can be debugged through examining the in-built provenance trace implementation and through a debug screen that produces verbose output during the execution process. During the execution, Triana will identify failures for components and provide feedback to the user if a component fails but it does not contain fail-safe mechanisms within the system for retrying a service for example. Kepler also has little or no generic capabilities for fault tolerance, individual workflow designers usually deal with the problem by encoding some of the exception handling and recovery mechanisms into the workflow itself. An extension to Kepler is being developed, a "smart re-run" feature based on data dependency information that can avoid unnecessary re-computations, similar to the way in which Pegasus works. Such a system would have similar knowledge about the workflow execution state even after an interruption so should enable the restart of a workflow from a check point. Kepler can also cope with unreliable data transfer, specifying a retry or fallback to an alternative transport protocol. This is a prototype of the "templates and frames" approach [84] which allows a structured composition of data-flow networks and control-flow (state-machines/transducers). Yet another approach is easily handled in COMAD: if an upstream actor creates or detects a fault, a special error tag/token can be injected into the COMAD data stream. A downstream "exception catcher" can then specify what to do with the faulty data; most actors would simply skip over data tagged as faulty.

Askalon supports fallback, task-level recovery, checkpointing and workflow-level redundancy. With all of the workflow systems that use external components or services the level of checkpointing is limited unless the components themselves support checkpointing that the systems can access. Generally checkpointing is done at the workflow application level and so it is only possible to restart a workflow from between components and not within a component itself. Askalon provides an implementation of this "light weight" checkpointing by providing URLs to intermediate data sets.

DAGMan also supports a number of recovery techniques. If DAGMan has crashed while submitting jobs to the underlying batch system, and the batch system continues to run jobs, DAGMan can recover its state upon restart (by reading logs provided by the batch system) without loosing information about the executing workflow. The DAGMan workflow management includes not only job submission and monitoring but also job preparation, cleanup, throttling, retry, and other actions necessary to ensure the successful workflow execution [85]. 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* 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.

Another approach is to use an external monitoring system. For example, if resource selection is deferred to runtime then the agent which selects the resource can monitor the part of the workflow using that resource. If the resource or application fails on that resource it is possible for the resource selection services to bring in a substitute. This approach is used in the LEAD project when running on the TeraGrid distributed computing resources [86]. The system is also capable of running multiple versions of the same workflow application component at the same time on different and continuing the workflow with the first instance that completes. This can improve throughput in time critical applications.

5.3. Adaptive workflow

Although data-driven systems have been around since the 1980s, the term Dynamic Data Driven Application Systems (DDDAS) was coined during a National Foundation for Science (NSF) workshop in 2000 [87]. DDDAS describes the ability for an application to dynamically incorporate data into an executing application and also for that data or some analysis of that data to steer the application process. This allows data being produced in real time by sensors or various other instruments to be fed into the computational workflow in order to allow the system to respond to data changes on-the-fly, which was considered a significant challenge for Grid computing. DDDAS techniques can take a number of forms at its simplest this involves following one branch instead of another in the workflow based upon a decision made using the input data. At the other end of the scale this may involve dynamic rewriting of the workflow itself to create an entirely new workflow, however in traditional programming models self-modifying code has been considered with some suspicion because of the possibilities for abuse. DDDAS systems require decision-making capabilities to be included within the workflow process. At the very least logic processes (e.g. if…then) should be supported and looping constructs are also often needed to support detection of certain thresholds or states (e.g. do…while loops or similar). Many of the workflow systems incorporate such features either at the language level or at the system level through specific components or constructs that support this type of behaviour.

There are a number of domains in which DDDAS can be applied including manufacturing process controls, resource management, weather and climate prediction, traffic management, disaster control systems, systems engineering, civil engineering, geo-exploration, social and behavioural modelling, cognitive measurement and bio-sensing. The LEAD [19] project for example is addressing the fundamental IT research challenges, and associated development, needed to create an integrated, scalable framework for identifying, accessing, preparing, assimilating, predicting, managing, analyzing, mining, and visualizing a broad array of meteorological data and model output independent of format and physical location. The storm modelling scenario is a good example of how this new paradigm affects the whole range of infrastructures involved in a system.

One possible use of these techniques is the situation where a computational solver of some description is used to explore a parameter space. If the solver is computationally expensive and the search space large, we could use a workflow with a course grained set of input parameters that explores the space at a high level. Areas of interest where the output data is evaluated to be above a certain threshold can be explored at a finer grained level by spawning a sub-workflow with a new input parameter set. This use case has been explored in Triana using Aspect Oriented Programming (AOP) techniques to perform the workflow rewriting, effectively creating sub-workflows that execute and feed back into the main workflow, and using Cactus as the computational solver. Other workflow tools could use this or other techniques to modify the workflow during execution. In principle Kepler workflows can modify themselves during execution. Ptolmey (on which Kepler is based) has some demos, e.g. where a variable number of instances of a "bank counter" are created; this is a "predictable" change of the workflow. More exciting things can be done: Sending a workflow (as if it were data) from one actor to another, where it is then executed. Nothing prevents you from changing the workflow in-flight. Users and tool builders have to be aware of the potential for abuse if open systems can be sent arbitrary workflows that can modify themselves, security and trust issues that have not been considered here then come into play.

6. Provenance

Data Provenance is a record of the history of the creation of a data object. If the data object was created as the result of a workflow then there must be a way to record the history of that creation. Specifically, the chain/graph of processes (including time stamp, program version number, component or service version number, execution host, library versions, etc) and intermediate data products back to the source data used to initialize the workflow.

The importance of data provenance cannot be underestimated. A complete provenance record for a data object allows us the possibility to reproduce the result and reproducibility is a critical component of the scientific method.

Recently, several workflow systems and other provenance-related research projects participated in a series of Provenance Challenge Workshops [88]. From the challenge it became clear that the various system capture similar information about workflow execution although they may present it in different ways. Some systems use internal structures to manage provenance information, some rely on external services, which can be quiet generic. Within Triana, for example, provenance is recorded locally as an internal format that has various levels of output. It can show the components executed, their parameters and even record the data sets in the provenance trace that pass through during execution. Triana is also integrated with external services such as those provided by the EU provenance project [89].

The Karma [90] provenance system is one that is largely workflow representation independent. Karma provides a searchable database of data provenance that has extensive capabilities for formulating data provenance queries. It can gather data from workflow systems in several different ways, but the simplest is for it to listen for the the state change events published during the execution of a workflow. For example, the Web services used in the LEAD project are all instrumented to produce a stream of events each time they are invoked. These events are published as WS-Eventing messages that are sent to a notification broker. The Karma system simply subscribes to the event stream. However, to make this work the workflow system must include workflow identification information along with each remote service invocation. This identification information is passed along in the service invocation events and is used by Karma to piece together the workflow history. Event streams are not the only way Karma works. Any workflow system that generates a detailed history of the actual execution could, in principle record that information in Karma.

6.1. Design provenance

In addition to tracking the provenance of data we can track the provenance of a workflow as it evolves from version to version. The VisTrails system [91] has an extensive set of capabilities for managing workflow design provenance, including the ability to explore variations on the design history or to see if two different workflows may have common elements or how they evolved from a common root.

The workflow design in VisTrails is done via a graphical user interface. When a user creates or modifies the workflow the system captures the user-made modifications. These can be additions/deletions of nodes and dependencies, modifications to parameter settings, etc. As a result the user is able to trace back exactly how the workflow was created, even including the avenues explored but not ultimately pursued. This type of provenance can also be very beneficial for workflow sharing or education, when collaborators are trying to understand how a particular workflow was created.

6.2. Provenance for transformed workflow execution

Since workflow systems may significantly modify the user-generated workflow before it is executed, the user may have trouble interpreting the results solely based on the execution provenance. For example, Pegasus is exploring the use of provenance tracking capabilities to record the workflow transformations performed during the workflow mapping process. This provenance includes information about which nodes in the workflow were clustered, which instance of the input data was selected for the computation, what intermediate data was selected for reuse, which execution sites where selected and why, and other information related to the workflow mapping process. As part of this work, Pegasus has been integrated with the PASOA provenance system [92,93].

7. Interoperability

With so many workflow systems out there and this paper touches only upon some of them, can we make a case for workflow interoperability, and if so what would this entail?

Applications today can be composed of very heterogeneous components, some which involve having the user in the loop, some which deal with streaming data, some which require high-performance resources for their execution, etc. Currently, there is no single workflow system that can accommodate all these various requirements, just as there is no single programming language used for all applications. Therefore, workflow developers would like to be able to use a variety of engines for their work. Users do not necessarily want full interoperability between the various workflow systems, but they would like to invoke one workflow from another and to re-use their workflow descriptions.

Recently, there has been effort, particularly in the VLE-WFBus project [95], where a number of different workflow systems are made interoperable through a runtime infrastructure. Each of the workflow systems connected by a workflow bus is wrapped and treated as a sub-workflow. The role of the workflow bus is to propagate information about the data objects to the correct sub-workflows, schedule the sub-workflows, and interface to the execution environment.

In scientific workflows, defining and refining a workflow to a point where it can be relied on to produce scientifically meaningful results can be extremely time-consuming and can take months or years of experimentation and validation. Yet if it is encoded completely in a language for a workflow system that becomes extinct the capability to execute it is lost. Of course this problem is a very general software longevity concern that goes well beyond workflows. However, if there is a standard workflow provenance model we could use that to encode the nature of the workflow execution independent from the underling execution engine. if this representation is precise enough it may allow us to build "compilers" that can translate the provenance record directly back to a new workflow representation.

8. Conclusion

As e-Science applications have grown in complexity from simple batch executions of data analysis tasks, workflow has emerged as an important enabling technology. A host of tools supporting workflow design and enactment have been developed and are now in use in the scientific community. In many cases these scientific workflow systems were developed in close collaboration with the scientists and resulting systems are well designed to handle the use-cases of that community. Because scientific research is so diverse in the method used from one discipline to another, the resulting collection of workflow tools demonstrate a wide variety of capabilities.

In this paper we attempted to demonstrate this diversity of capabilities found in e-Science workflow. We have first explored it from the perspective of modes of expression. The language for defining a workflow can be as traditional as the textual representations used in conventional programming. However scientists have found that a graphical composition model is convenient for many applications. The most novel approaches may be those that compile a workflow from the specification of a scientists high-level queries. This last area is one that is the subject of current research.

We have also described workflow systems along the dimensions of internal representation and execution. In many cases graphs are used for the internal model, but several other interesting representations are also described. Execution models also vary widely from system to system. In many cases a data-driven data-flow model is used by the enactment engine. In other cases, data flow is seen as limiting, so it is enhanced with various control constructs.

Mapping a distributed scientific workflow to a set of computational and data resources is a task that may be done prior to execution, but it may also be done on-the-fly at runtime. In the later case, it is possible to be very adaptive in how the resources are selected and it may also enable better fault handling.

Finally, having good workflow tools has enabled us to automate the process of building data and workflow provenance. Combined with good data catalogs and data management systems, it is now possible to provide complete experimental workbenches for entire communities of scientific users. Data and workflows can be shared and, through community use and refinement, evolved to meet new challenges. Having data provenance allows a scientist to return to the point of creation of a data object to understand the workflow that created it and the original source of data that was used.

Acknowledgements

We would like to thank a number of contributors from the workflow systems we discussed in this paper: Tom Oinn, Dave De Roure ad Carole Goble (Taverna), Peter Kacsuk (P-Grade), Gregor von Laszewski (CoG Kit Karajan), Bruno Wassermann and Wolfgang Emmerich (Eclipse BPEL Designer), Thomas Fahringer and Radu Prodan (Askalon), Bertram Ludäscher (Kepler). Finally, we would like to thank all the contributors to Workflows for eScience book that also helped with the input to this text.

Ewa Deelman acknowledges the support of the National Science Foundation under grants: OCI-0722019 and CCF-0725332. Ian Taylor and Matthew Shields also acknowledges support for Triana under the STFC GridOneD, TRIACS and DART grants and the OMII UK WHIP project for helping provide portal workflow plug-ins.

References

[1] J. Yu, R. Buyya, A taxonomy of workflow management systems for grid computing, Tech. Rep. GRIDS-TR-2005-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, March 2005. http://www.gridbus.org/reports/GridWorkflowTaxonomy.pdf.

[2] Y. Han, A. Sheth, C. Bussler, A taxonomy of adaptive workflow management, in: Conference on Computer–Supported Cooperative Work, CSCW-98, Seattle, WA, 1998. http://ccs.mit.edu/klein/cscw98/.

[3] I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows for e-Science, Springer, New York, Secaucus, NJ, USA, 2007.

[4] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana, Business process execution language for web services version 1.1.

[5] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M.R. Pocock, A. Wipat, P. Li, Taverna: A tool for the composition and enactment of bioinformatics workflows, Bioinformatics 20 (17) (2004) 3045–3054.

[6] Condor team, DAGMan: A Directed Acyclic Graph Manager, July 2005. http://www.cs.wisc.edu/condor/dagman/.

[7] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, D. Katz, Pegasus: A framework for mapping complex scientific workflows onto distributed systems, Scientific Programming Journal 13 (3) (2005) 219–237.

[8] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, J. Kim, Wings for Pegasus: Creating large-scale scientific applications using semantic representations of computational workflows, in: Proceedings of the 19th Annual Conference on Innovative Applications of Artificial Intelligence, IAAI, Vancouver, British Columbia, Canada.

[9] G. Berriman, J. Good, A. Laity, A. Bergou, J. Jacob, D. Katz, E. Deelman, C. Kesselman, G. Singh, M. Su, et al. Montage: A grid enabled image mosaic service for the national virtual observatory, in: Astronomical Data Analysis Software and Systems, ADASS, XIII.

[10] G. Berriman, E. Deelman, J. Good, J. Jacob, D. Katz, C. Kesselman, A. Laity, T. Prince, G. Singh, M. Su, Montage: A grid-enabled engine for delivering custom science-grade mosaics on demand, in: Proceedings of SPIE 5493, 2004, pp. 221–232.

[11] A. Lathers, M. Su, A. Kulungowski, A. Lin, G. Mehta, S. Peltier, E. Deelman, M. Ellisman, Enabling parallel scientific applications with workflow tools, in: Proceedings of Challenges of Large Applications in Distributed Environments, CLADE.

[12] J. Muench, et al. SCEC earthworks science gateway: Widening SCEC community access to the TeraGrid, in: TeraGrid 2006 Conference.

[13] H. Lord, Improving the application development process with modular visualization environments, ACM SIGGRAPH Computer Graphics 29 (2) (1995) 10–12.

[14] S.G. Parker, M. Miller, C.D. Hansen, C.R. Johnson, An integrated problem solving environment: The SCIRun computational steering system, in: Proceedings of the 31st. Hawaii International Conference on System Sciences, HICSS-31, 1998, pp. 147–156.

[15] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, S. Mock, Kepler: An extensible system for design and execution of scientific workflows, in: 16th International Conference on Scientific and Statistical Database Management, SSDBM, IEEE Computer Society, New York, 2004, pp. 423–424.

[16] I. Taylor, M. Shields, I. Wang, A. Harrison, Visual grid workflow in triana, Journal of Grid Computing 3 (3–4) (2005) 153–169.

[17] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, H. Vo, Managing the evolution of dataflows with visTrails, in: IEEE Workshop on Workflow and Data Flow for Scientific Applications, SciFlow 2006.

[18] S. Shirasuna, XBaya workflow composer. http://www.extreme.indiana.edu/xgws/xbaya.

[19] K. Droegemeier, et al., Service-oriented environments in research and education for dynamically interacting with mesoscale weather, Computing in Science and Engineering 7 (6) (2005) 12–29.

[20] Eclipse, Eclipse BPEL Project, see Web site at: http://www.eclipse.org/bpel.

[21] The myExperiment Project, http://www.myexperiment.org.

[22] I. Taylor, Triana generations, in: Scientific Workflows and Business Workflow Standards in e-Science in Conjunction with Second IEEE International Conference on e-Science, Amsterdam, Netherlands, 2006.

[23] I. Taylor, E. Al-Shakarchi, S.D. Beck, Distributed audio retrieval using Triana, DART, in: International Computer Music Conference, ICMC, 2006, November 6–11, at Tulane University, USA, 2006, pp. 716–722.

[24] Data Mining Tools and Services for Grid Computing Environments (DataMiningGrid), http://www.datamininggrid.org/.

[25] A. Ali, O. Rana, I. Taylor, Web services composition for distributed data mining, in: ICPP 2005 Workshops, International Conference Workshops on Parallel Processing, IEEE, New York, 2005, pp. 11–18.

[26] G. von Laszewski, M. Hategan, Java CoG Kit Karajan/Gridant workflow guide, tech. rep., Technical Report, Argonne National Laboratory, Argonne, IL, USA, 2005.

[27] Y. Gil, Workflow composition: Semantic representations for flexible automation, in: Workflows for e-Science, Springer, New York, 2007, pp. 244–257.

[28] J. Kim, M. Spraragen, Y. Gil, An intelligent assistant for interactive workflow composition, in: IUI'04: Proceedings of the 9th International Conference on Intelligent User Interface, ACM Press, New York, 2004, pp. 125–131.

[29] P. Maechling, H. Chalupsky, M. Dougherty, E. Deelman, Y. Gil, S. Gullapalli, V. Gupta, C. Kesselman, J. Kim, G. Mehta, B. Mendenhall, T. Russ, G. Singh, M. Spraragen, G. Staples, K. Vahi, Simplifying construction of complex workflows for non-expert users of the Southern California earthquake center community modeling environment, ACM SIGMOD Record 34 (3) (2005) 24–30.

[30] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman, S. Callaghan, D. Okaya, H. Francoeur, V. Gupta, Y. Cui, K. Vahia, T. Jordan, E. Field, Workflows for e-Science, Springer, New York, 2007, pp. 143–166. Ch. SCEC CyberShake Workflows — Automating Probabilistic Seismic Hazard Analysis Calculations.

[31] K. Knight, D. Marcu, Machine Translation in the Year 2004, in: Proceedings of the 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP, vol. 5, IEEE Computer Society, New York, 2005, pp. 965–968.

[32] S. Thakkar, J.L. Ambite, C.A. Knoblock, Composing, optimizing, and executing plans for bioinformatics web services, in: Data Management, Analysis and Mining for Life Sciences, VLDB Journal 14 (3) (2005) 330–353 (special issue).

[33] D. McDermott, Estimated-Regression planning for interactions with web services, in: M. Ghallab, J. Hertzberg, P. Traverso (Eds.), 6th International Conference on Artificial Intelligence Planning and Scheduling, AAAI Press, Menlo Park, CA, 2002.

[34] S. Mcllraith, T. Son, Adapting golog for programming in the semantic web, in: Fifth International Symposium on Logical Formalizations of Commonsense Reasoning, 2001, pp. 195–202 (in press).

[35] myGrid, http://www.mygrid.org.uk/.

[36] ISO/IEC 15909-1, High-level Petri nets — Part 1: Concepts, definitions and graphical notation, 2004.

[37] M. Fowler, K. Scott, UML Distilled, Addison-Wesley, 1997.

[38] T. Fletcher, C. Ltd, P. Furniss, A. Green, R. Haugen, BPEL and Business Transaction Management: Choreology Submission to OASIS WS-BPELTechnical Committee., Published on Web.

[39] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, S.L. Price, Grid service orchestration using the Business Process Execution Language (BPEL), Journal of Grid Computing 3 (3–4) (2005) 283–304.

[40] R. Allan, D. Meredith, e-HTPX–HPC, grid and web-portal technologies in high throughput protein crystallography, in: Proceedings of the UK e-Science All Hands Meeting, Nottingham, UK.

[41] Y. Wang, C. Hu, J. Huai, A new grid workflow description language, in: Proc. of the 2005 IEEE Int'l Conf. on Services Computing, vol. 2, pp. 257–260.

[42] I. Foster, J. Voeckler, M. Wilde, Y. Zhao, Chimera: A virtual data system for representing, querying, and automating data derivation, in: 14th International Conference on Scientific and Statistical Database Management, SSDBM'02, IEEE Computer Society Press, New York, 2002, pp. 37–46.

[43] M. Alt, A. Hoheisel, H.-W. Pohl, S. Gorlatch, A grid workflow language using high-level petri nets, in: R. Wyrzykowski, J. Dongarra, N. Meyer, J. Wasniewski (Eds.), Proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics PPAM'2005, in: Lecture Notes in Computer Science, vol. 3911, Springer, New York, 2006, pp. 715–722.

[44] KWF, http://www.kwfgrid.eu/.

[45] Z. Guan, et al. Grid-flow: A grid-enabled scientific workflow system with a petri net-based interface, Ph.D. Thesis, University of Alabama at Birmingham, 2006.

[46] GridFlow Home Page, http://gridflow.ca/.

[47] T. Fahringer, R. Prodan, R. Duan, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, M. Wieczorek, ASKALON: A grid application development and computing environment, in: 6th International Workshop on Grid Computing, IEEE Computer Society Press, New York, 2005, pp. 122–131.

[48] T. Fahringer, J. Qin, S. Hainzer, Specification of grid workflow applications with AGWL: An abstract grid workflow language, in: International Symposium on Cluster Computing and the Grid, CCGRID 2005, vol. 2, IEEE Computer Society Press, New York, 2005, pp. 676–685.

[49] J. Qin, T. Fahringer, Advanced data flow support for scientific grid workflow applications, in: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Supercomputing 2007, SC-07.

[50] S. Bowers, T. McPhillips, B. Ludaescher, A provenance model for collection-oriented scientific workflows, Concurrency and Computation: Practice and Experience.

[51] I. Brandic, S. Pllana, S. Benkner, An approach for the high-level specification of QoS-aware grid workflows considering location affinity, Scientific Programming 14 (3) (2006) 231–250.

[52] R.T. Fielding, Architectural styles and the design of network-based software architectures, Ph.D. Thesis, University of California, Irvine, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[53] Web Services Description Language (WSDL) 1.1, Tech. Rep., W3C, 2001. http://www.w3.org/tr/wsdl.

[54] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid information services for distributed resource sharing, in: HPDC'01: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing, HPDC-10'01, IEEE Computer Society, Washington, 2001, pp. 181–184.

[55] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: Enabling scalable virtual organization, The International Journal of High Performance Computing Applications 15 (3) (2001) 200–222.

[56] A. Kertész, G. Sipos, P. Kacsuk, Brokering multi-grid workflows in the P-GRADE portal, in: Euro-Par 2006: Parallel Processing, vol. 4375, Springer, Berlin, 2007, pp. 138–149.

[57] T. Delaitre, T. Kiss, A. Goyeneche, G. Terstyanszky, S. Winter, P. Kacsuk, GEMLCA: Running legacy code applications as grid services, Journal of Grid Computing 3 (1–2) (2005) 75–90.
[58] A. Kertesz, Brokering solutions for Grid middlewares, in: Pre-proc. of 1st Doctoral Workshop on Mathematical and Engineering Methods in Computer Science.
[59] A. Peris, P. Lorenzo, F. Donno, A. Sciab, S. Campana, R. Santinelli, LCG-2 User Guide, LHC Computing Grid Manuals Series. Available via: https://edms.cern.ch/file/454439//LCG-2-UserGuide.pdf.
[60] S. Burke, S. Campana, A. Peris, F. Donno, P. Lorenzo, R. Santinelli, A. Sciaba, gLite 3.0 User Guide, 2006.
[61] P. Kacsuk, T. Kiss, G. Sipos, Solving the grid interoperability problem by P-GRADE portal at workflow level, in: Proc. of the Grid-Enabling Legacy Applications and Supporting End User Workshop, in conjunction with HPDC 6 3–7.
[62] T. Glatard, G. Sipos, J. Montagnat, Z. Farkas, P. Kacsuk, Workflow-level parametric study support by MOTEUR and the P-GRADE portal, in: Workflows for e-Science, Springer, New York, 2007, pp. 279–299.
[63] G. Allen, K. Davis, K.N. Dolkas, N.D. Doulamis, T. Goodale, T. Kielmann, A. Merzky, J. Nabrzyski, J. Pukacki, T. Radke, M. Russell, E. Seidel, J. Shalf, I. Taylor, Enabling applications on the grid: a gridlab overview, in: Grid Computing: Infrastructure and Applications, International Journal of High Performance Computing Applications 17 (4) (2003) 449–466 (special issue).
[64] I. Taylor, M. Shields, I. Wang, O. Rana, Triana applications within grid computing and peer to peer environments, Journal of Grid Computing 1 (2) (2003) 199–217.
[65] J. Nabrzyski, Grid(Lab) resource management system (GRMS), Tech. Rep., GridLab, 2004. http://www.man.poznan.pl/coe/documents/Gridlab_GRMS_WP.pdf.
[66] K. Czajkowski, D.F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, The WS-resource framework, Tech. Rep., The Globus Alliance, 2004.
[67] G. von Laszewski, M. Hategan, D. Kodeboyina, Java CoG kit workflow, in: Workflows for e-Science, Springer, New York, 2007, pp. 143–166.
[68] P. Couvares, T. Kosar, A. Roy, J. Weber, K. Wenger, Workflows for e-Science, Springer, New York, 2007, Ch. Workflow Management in Condor, pp. 357–375.
[69] D. Epema, M. Livny, R. van Dantzig, X. Evers, J. Pruyne, A worldwide flock of condors: Load sharing among workstation clusters, in: Resource Management in Distributed Systems, Future Generation Computer Systems 12 (1) (1996) 53–65 (special issue).
[70] T. Braun, H. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, et al., A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed Computing 61 (6) (2001) 810–837.
[71] H. Topcuoglu, S. Hariri, M. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing.
[72] A. Ramakrishnan, G. Singh, H. Zhao, E. Deelman, R. Sakellariou, K. Vahi, K. Blackburn, D. Meyers, M. Samidi, Scheduling data-intensiveworkflows onto storage-constrained distributed resources, in: CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, IEEE Computer Society, Washington, DC, USA, 2007, pp. 401–409.
[73] G. Singh, C. Kesselman, E. Deelman, Adaptive pricing for resource reservations in shared environments, in: GRID, 2007, pp. 74–80.
[74] G. Singh, C. Kesselman, E. Deelman, A provisioning model and its comparison with best-effort for performance-cost optimization in grids, in: HPDC'07: Proceedings of the 16th International Symposium on High Performance Distributed Computing, ACM, New York, NY, USA, 2007, pp. 117–126.
[75] M. Wieczorek, M. Siddiqui, A. Villazon, R. Prodan, T. Fahringer, Applying advance reservation to increase predictability of workflow execution on the grid, in: E-SCIENCE'06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, IEEE Computer Society, Washington, DC, USA, 2006, p. 82.
[76] R. Henderson, D. Tweten, Portable batch system: External reference specification, Ames Research Center.
[77] S. Zhou, LSF: Load sharing in large-scale heterogeneous distributed systems, in: Proc. Workshop on Cluster Computing, 1992, pp. 1995–1996.
[78] M. Litzkow, M. Livny, M. Mutka, Condor — A hunter of idle workstations, in: Proceedings of the 8th International Conference on Distributed Computing Systems, IEEE Computer Society, New York, 1988, pp. 104–111.
[79] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke, Security for grid services, in: Twelfth International Symposium on High Performance Distributed Computing, HPDC-12, IEEE Computer Society Press, New York, 2003, pp. 48–57.
[80] J. Meehean, M. Livny, A service migration case study: Migrating the Condor schedd, in: Midwest Instruction and Computing Symposium, 2005.
[81] The Globus Alliance. See Web site at: http://www.globus.org.
[82] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, A resource management architecture for metacomputing systems, in: D.G. Feitelson, L. Rudolph (Eds.), IPPS/SPDP'98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, in: Lecture Notes in Computer Science, vol. 1459, Springer Verlag, London, 1998, pp. 62–82. http://www.globus.org.
[83] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, J. Shalf, The cactus worm: Experiments with dynamic resource discovery and allocation in a grid environment, The International Journal of High Performance Computing Applications 15 (4) (2001) 345–358. http://citeseer.ist.psu.edu/article/allen01cactus.html.
[84] S. Bowers, B. Ludaescher, A. Ngu, T. Critchlow, Enabling scientific workflow reuse through structured composition of dataflow and control-flow, in: Proceedings of the 22nd International Conference on Data Engineering Workshops, ICDEW'06-Volume 00.
[85] P. Couvares, T. Kosar, A. Roy, J. Weber, K. Wenger, Workflow management in condor, in: Workflows for e-Science, Springer, New York, 2007, pp. 357–375.
[86] The TeraGrid Project, http://www.teragrid.org/http://www.teragrid.org/.
[87] F. Darema, Grid computing and beyond: The context of dynamic data driven applications systems, in: Proceedings of the IEEE, vol. 93, 2005, pp. 692–697.
[88] R. Bose, I. Foster, L. Moreau, Report on the International Provenance and Annotation Workshop, IPAW'06, 3–5 May 2006, Chicago, ACM SIGMOD Record 35 (3) (2006) 51–53.
[89] L. Chen, V. Tan, F. Xu, A. Biller, P. Groth, S. Miles, J. Ibbotson, L. Moreau, A proof of concept: Provenance in a service oriented architecture, in: Proceedings of the UK OST e-Science Second All Hands Meeting 2005, AHM05.
[90] Y.L. Simmhan, B. Plale, D. Gannon, Performance evaluation of the karma provenance framework for scientific workflows, in: International Provenance and Annotation Workshop, IPAW, Springer, Berlin, 2006.
[91] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, H. Vo, VisTrails: Visualization meets data management, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, 2006, pp. 745–747.
[92] S. Miles, E. Deelman, P. Groth, K. Vahi, G. Mehta, L. Moreau, Connecting scientific data to scientific experiments with provenance.
[93] S. Miles, P. Groth, E. Deelman, K. Vahi, G. Mehta, L. Moreau, Provenance: The bridge between experiments and data, Computing in Science and Engineering 10 (3) (2008) 38–46.
[94] A.S.Z. Belloum, D.L. Groep, Z.W. Hendrikse, B.L.O. Hertzberger, V. Korkhov, C.T.A.M. de Laat, D. Vasunin, Vlam-g: a grid-based virtual laboratory, Future Generation Computer Systems 19 (2) (2003) 209–217.
[95] Z. Zhao, S. Booms, A. Belloum, C. de Laat, B. Hertzberger, Vle-wfbus: a scientific workflow bus for multi e-science domains, in: Proceedings of the 2nd IEEE International conference on e-Science and Grid computing, IEEE Computer Society Press, Amsterdam, The Netherlands, 2006, pp. 11–19.

Ewa Deelman is an Assistant Research Professor at the USC Computer Science Department and a Project Leader at the USC Information Sciences Institute. Dr. Deelman's research interests include the design and exploration of collaborative scientific environments based on distributed technologies, with particular emphasis on workflow management as well as the management of large amounts of data and metadata. At ISI, Dr. Deelman is leading the Pegasus project, which designs and implements workflow mapping techniques for large-scale workflows running in distributed environments. Prior to joining ISI in 2000, she was a Post Doctoral fellow at UCLA conducting research in the area of performance prediction of large-scale applications on high performance machines. Dr. Deelman received her Ph.D. from Rensselaer Polytechnic Institute in Computer Science in 1997 in the area of parallel discrete event simulation. Dr. Deelman is an Associate Editor responsible for Grid Computing for the Scientific Programming Journal and a chair of the OGF Workflow Management Research Group.

Dennis Gannon's research interests include cyberinfrastructure, programming systems and tools, distributed computing, computer networks, parallel programming, computational science, problem solving environments and performance analysis of Grid and MPP systems. He led the DARPA HPC + + project and he was one of the architects of the Department of Energy SciDAC Common Component Architecture (CCA) project. This work has led to a framework for building component-based scientific applications. He was a partner in the NSF Computational Cosmology Grand Challenge project and the NCSA Alliance where he is helped to lead an effort to design Grid "Portals" which are desktop frameworks for Grid access. He was a co-founder the Java Grande Forum. He is the co-chair of the Global Grid Forum working groups on Grid Computing Environments and he was on the steering committee of the GGF. He is currently on the Executive Steering Committee of the NSF Teragrid Project and deeply involved with the Science Gateways efforts there. He is also a co-pi on the NSF LEAD project which is building cyberinfrastructure for dynamic, adaptive weather prediction. He is the Science Director for the Indiana Pervasive Technology Labs and the past Chair of the Department of Computer Science at Indiana University. Dr. Gannon was the Program Chair for the IEEE 2002 High Performance Distributed Computing Conference. He also served as General Chair of the 1998 International Symposium on Scientific Object Oriented Programming Environments (ISCOPE) and the 2000 ACM Java Grande Conference, and Program Chair for the 1997 ACM International Conference on Supercomputing as well as the 1995 IEEE Frontiers of Massively Parallel Processing. He was the Program Chair for the International Grid Conference, Barcelona, 2006.

Matthew Shields is currently a Senior Research Associate at the Computer Science at Cardiff University. His research is concerned with the investigation of technologies to help developers and application scientists utilise Grid computing and he is the main developer of the Triana workflow environment, discussed in this paper. His research interests include loosely coupled, standards based message passing interfaces for distributed computing, such as Web and Grid services, and Peer to Peer computing. Matthew is one of the co-editors for Workflows for eScience, published by Springer.

Ian Taylor is a Senior lecturer in distributed systems at the School of Computer Science and concurrently holds an adjunct Assistant Professorship at the Center for Computation & Technology at Louisiana State University. He has a Ph.D. in Physics and Music and is the co-ordinator of Triana activities at Cardiff (http://www.trianacode.org). Through this he has been active in many major projects including GridLab, CoreGrid and GridOneD applying distributed techniques and workflow for Grid and P2P computing in application areas ranging from healthcare and distributed audio to astrophysics. Ian is a member of oversight committee for DPAC project providing data analysis for the ESA GAIA mission to map the Galaxy. Ian has written a professional book for Springer entitled P2P, Web Services and Grids and has been the lead edited another called Workflows for eScience with Dennis Gannon, Ewa Deelman and Matthew Shields. He has published over 50 scientific papers, been a guest editor for the Journal of Grid Computing on workflow and he is a co-chair for the Workflow Management Research Group in the OGF.