Hindawi Publishing Corporation
ISRN Software Engineering
Volume 2013, Article ID 404525, 15 pages
http://dx.doi.org/10.1155/2013/404525

Review Article

Workflow Systems for Science: Concepts and Tools

Domenico Talia

ICAR-CNR and University of Calabria, 87036 Rende, Italy

Correspondence should be addressed to Domenico Talia; talia@deis.unical.it

Received 3 December 2012; Accepted 23 December 2012

Academic Editors: J. Cao, B. C. Lai, and K. Thramboulidis

The wide availability of high-performance computing systems, Grids and Clouds, allowed scientists and engineers to implement more and more complex applications to access and process large data repositories and run scientific experiments in silico on distributed computing platforms. Most of these applications are designed as workflows that include data analysis, scientific computation methods, and complex simulation techniques. Scientific applications require tools and high-level mechanisms for designing and executing complex workflows. For this reason, in the past years, many efforts have been devoted towards the development of distributed workflow management systems for scientific applications. This paper discusses basic concepts of scientific workflows and presents workflow system tools and frameworks used today for the implementation of application in science and engineering on high-performance computers and distributed systems. In particular, the paper reports on a selection of workflow systems largely used for solving scientific problems and discusses some open issues and research challenges in the area.

1. Introduction

Workflows have emerged as an effective paradigm to address the complexity of scientific and business applications. The wide availability of high-performance computing systems, Grids and Clouds, allowed scientists and engineers to implement more and more complex applications to access and process large data repositories and run scientific experiments in silico on distributed computing platforms. Most of these applications are designed as workflows that include data analysis, scientific computation methods, and complex simulation techniques.

The design and execution of many scientific applications require tools and high-level mechanisms. Simple and complex workflows are often used to reach this goal. For this reason, in the past years, many efforts have been devoted towards the development of distributed workflow management systems for scientific applications. Workflows provide a declarative way of specifying the high-level logic of an application, hiding the low-level details that are not fundamental for application design. They are also able to integrate existing software routines, datasets, and services in complex compositions that implement scientific discovery processes.

In the most general terms, according to the Workflow Management Coalition [1–3], a workflow is "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules." The same definition can be used for scientific processes composed of several processing steps that are connected together to express data and/or control dependencies [4, 5]. The term process indicates a set of tasks linked together with the goal of creating a product, calculating a result, or providing a service. Hence, each task (or activity) represents a piece of work that forms one logical step of the overall process [6, 7].

A workflow is a well-defined, and possibly repeatable, pattern or systematic organization of activities designed to achieve a certain transformation of data. Workflows, as practiced in scientific computing, derive from several significant precedent programming models that are worth noting because these have greatly influenced the way we think about workflows in scientific applications. We can call

these the dataflow model in which data is streamed from one actor to another. While the pure dataflow concept is extremely elegant, it is very hard to put it in practice because distributing control in a parallel or distributed system can create applications that are not very fault tolerant.

Consequently, many workflow systems that use a dataflow model for expressing the computation may have an implicit centralized control program that sequences and schedules each step. An important benefit of workflows is that, once defined, they can be stored and retrieved for modifications and/or reexecution: this allows users to define typical patterns and reuse them in different scenarios [8]. Figure 1 shows an example of workflow of a scientific experiment of the Neptune projects led by University of Washington, which includes control nodes (processes) and data nodes (data value). This experiment involves collection of data by ocean buoys (containing a temperature sensor and an ocean current sensor), which is then processed by a scientific workflow. The scientific workflow is composed of four steps to process the data from the sensors and create visualization charts as output [9].

The definition, creation, and execution of workflows are supported by a so-called workflow management system (WMS). A key function of a WMS during the workflow execution (or enactment) is coordinating the operations of the individual activities that constitute the workflow.

Through the integrated use of computer science methods and technologies and scientific discovery processes, science went to a new era where scientific science methods changed significantly by the use of computational methods and new data analysis strategies that created the so-called *e-science paradigm*, as discussed by Bell et al. [10]. For example, the Pan-STARRS [11] astronomical survey uses Microsoft Trident Scientific Workflow Workbench workflows to load and validate telescope detections running at about 30 TB per year. Similarly, the USC Epigenome Center is currently using the Pegasus workflow system to exploit the Illumina Genetic Analyzer (GA) system to generate high-throughput DNA sequence data (up to 8 billion nucleotides per week) to map the epigenetic state of human cells on a genomewide scale. In this scenario, scientific workflows demonstrate their effectiveness as an effective paradigm for programming at high-level complex scientific applications that in general run on supercomputers or on distributed computing infrastructures such as Grids, peer-to-peer systems, and Clouds [12–14].

In this paper we discuss basic concepts of scientific workflows and present a selection of the most used scientific workflow systems such as Taverna, Pegasus, Triana, Askalon, Kepler, GWES, and Karajan, which provide innovative tools and frameworks for the implementation of application in science and engineering. These workflow systems run on parallel computers and distributed computing systems to get high-performance and large data access that are needed in the solution of complex scientific problems [15, 16]. Each one of those workflow management systems running in parallel and distributed computing environments has interesting features and represents a valid example of how workflow-based programming can be used in developing scientific

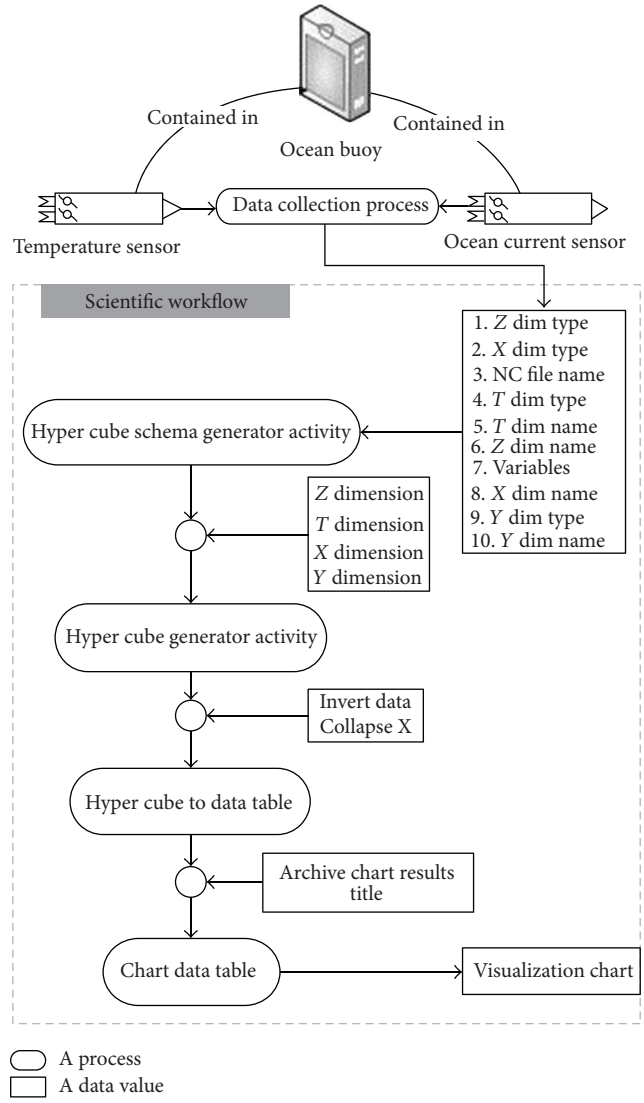

Figure 1: The scientific workflow defined in the Neptune project.

applications. We also discuss some open research issues in the area of scientific workflows that are investigated today and are worthy of further studies by the computer science community for providing novel solutions in the area.

The rest of the paper is organized as follows. Section 2 introduces the main programming issues in the area of scientific workflows. Section 3 presents a set of significant workflow management systems running on parallel/distributed computing systems. Section 4 discusses a set of main issues that are still open in the area of scientific workflows and of topics that must be investigated to find innovative solutions. Section 5 concludes the paper.

2. Workflow Programming

A main issue in workflow management systems is the programming structures they provide to the developer who needs to implement a scientific application. Moreover,

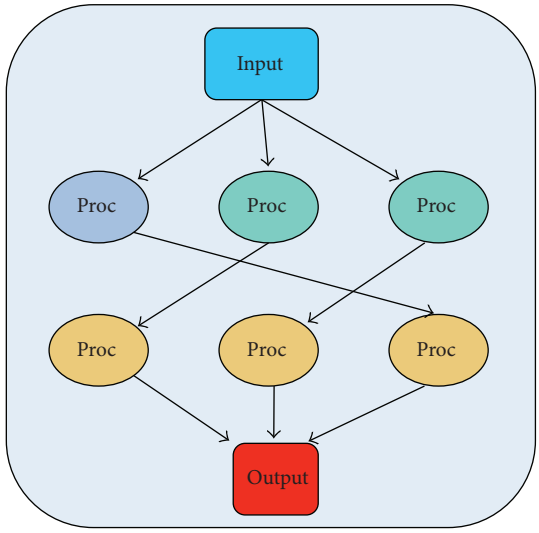

Figure 2: A simple DAG including data and control nodes.

whereas some systems provide a textual programming interface, others are based on a visual programming interface. These two different interfaces imply different programming approaches.

This section discusses the current programming approaches used in scientific workflow systems and compares them. Often a scientific workflow is programmed as a graph of several data and processing nodes that include predefined procedures programmed in a programming language such as Java, C++, and Perl. According to this approach, a scientific workflow is a methodology to compose predefined programs that run single tasks, but that composed together represent a complex application that generally needs large resources for running and may take long running time to complete. For an introductive reference, see [17], it gives a taxonomy of main features of scientific workflows. In fact, it is not rare to have scientific workflows, for example, in the astronomy domain or in bioinformatics [18, 19], which take several days or weeks to complete their execution.

Workflow tasks can be composed together following a number of different patterns, whose variety helps designers addressing the needs of a wide range of application scenarios [20]. A comprehensive collection of workflow patterns, focusing on the description of control flow dependencies among tasks, has been described in [20]. The most common programming structure used in workflow management systems is the directed acyclic graph (DAG) (see Figure 2) or its extension that includes loops, that is the directed cyclic graph (DCG).

Of the many possible ways to distinguish workflow computations, one is to consider a simple complexity scale. At the most basic level one can consider linear workflows, in which a sequence of tasks must be performed in a specified linear order. The first task transforms an initial data object into new data object that is used as input in the next data-transformation task. The execution of the entire chain of tasks may take a few minutes, or it may take days, depending on the computational grain of each single task in the pipeline. When the execution time is short, the most common workflow programming tool is a simple script written, for instance, in Python or Perl or even Matlab. The case of longer running workflows often requires more sophisticated tools or programming languages.

At the next level of complexity, one can consider workflows that can be represented by a DAG, where nodes of the graph represent tasks to be performed and edges represent dependencies between tasks. Two main types of dependencies can be considered: data dependencies (where the output of a task is used as input by the next tasks) and control dependencies (where before to start one or a set of tasks some tasks must be completed). This is harder to represent with a scripting language without a substantial additional framework behind it, but it is not at all difficult to represent with a tool like Ant. It is the foundation of the Directed Acyclic Graph Manager (DAGMan) [21], a meta-scheduler of Condor [22], a specialized workload management system developed by the University of Wisconsin, and the execution engine of Pegasus. Applications that follow this pattern can be characterized by workflows in which some tasks depend upon the completion of several other tasks that may be executed concurrently [11].

A directed acyclic graph (DAG) can be used to represent a set of programs where the input, output, or execution of one or more programs is dependent on one or more other programs. According to this model, also in DAGMan programs are nodes in the graph, the edges (arcs) identify the program dependencies. For task execution, Condor finds computers for the execution of programs, but it does not schedule programs based on dependencies [23]. DAGMan submits tasks to Condor in an order represented by a DAG and processes the task results. An input file defined prior to submission describes the workflow as a DAG, and Condor uses a description file for each program in the DAG. DAGMan is responsible for scheduling, recovery, and reporting for the set of programs submitted to Condor.

The next level of workflow complexity can be characterized cyclic graphs, where cycles represent some form of implicit or explicit loop or iteration control mechanisms. In this case, the workflow graph often describes a network of tasks where the nodes are either services or some form of software component instances or represent more abstract control objects. The graph edges represent messages or data streams or pipes that exchange work or information among services and components.

The highest level of workflow structure is one in which a compact graph model is not appropriate. This is the case when the graph is simply too large and complex to be effectively designed as a graph. However, some tools allow one to turn a graph into a new first-class component or service, which can then be included as a node in another graph (a workflow of workflows or hierarchical workflow). This technique allows graphs of arbitrary complexity to be constructed. This nested workflow approach requires the use of more sophisticated tools both at the compositing stage and at the execution stage. In particular, this approach makes harder the task-to-resource mapping and the task scheduling

during the workflow execution that typically is done on large-scale distributed or parallel infrastructures such as HPC systems or Cloud computing platforms [24].

In the case of workflow enactment, two issues must be also taken into account, efficiency and robustness. In terms of efficiency, the critical issue is the ability to quickly bind workflow tasks to the appropriate computing resources. It also heavily depends on the mechanisms used to assign data to tasks, to move data between tasks that need them at various stages of the enactment. For instance, considering a service-oriented scenario, we cannot assume that web service protocols like SOAP should be used in anything other than task/service control and simple message delivery [25–29]. Complex ad/or data movement between components of the workflow must be either via an interaction with a data movement service, or through specialized binary-level data channel running directly between the tasks involved.

Robustness is another issue, making the reasonable assumption that some parts of a workflow may fail. It is essential that exception handling includes mechanisms to recover from failure as well as detecting it. Also failure is something that can happen to a workflow enactment engine. A related issue is the monitoring of the workflow. In addition, in restarting a workflow from a failure checkpoint, a user may wish to track progress of the enactment. In some cases the workflow is event driven and a log of the events that trigger the workflow processing can be analyzed to understand how the workflow is progressing. This is also an important aspect of debugging a workflow. A user may wish to execute the workflow step by step to understand potential errors in the flow logic.

Tools and programming interfaces for scientific workflow composition are important components of workflow systems. Through tools and interfaces, a developer can compose her/his scientific workflow and express details about data sources, task dependencies, resource availability, and other design or execution constraints. Most scientific workflow systems offers graphical interface that offers high-level mechanisms for composition, while a few systems exhibit tradition text-based programming interfaces. Workflow users exploit the features of interfaces that scientific workflows expose in order to build their workflow [30]. This stage corresponds to the design stage of a workflow. In general terms, two main workflow levels can be found on this regard, though there are other approaches that even differentiate more abstraction levels.

(i) *Abstract Workflows*. At this high level of abstraction a workflow contains just information about what has to be done at each task along with information about how tasks are interconnected. There is no notion of how input data is actually delivered or how tasks are implemented.

(ii) *Concrete Workflows*. The mapping stage to a concrete workflow annotates each of the tasks with information about the implementation and/or resources to be used. Information about method invocation and actual data exchange format is also defined.

In case a user is familiar with the technology and the resources available, they can even specify concrete workflows directly. Once a workflow specification is produced, it is sent to the workflow engine for the execution phase. At this stage, workflow tasks are mapped also onto third-party, distributed, and heterogeneous resources and the scientific computations are accomplished [31, 32].

3. Scientific Workflow Management Systems

As mentioned above, Workflow Management Systems are software environments providing tools to define, compose, map, and execute workflows. There are several WMSs on the market, most of them targeted to a specific application domain, such as in our case of scientific workflows [33–35]. In this section we focus on some significant WMSs developed by the research community, with the goal of identifying the most important features and solutions that have been proposed for workflow management in the scientific domain [36–43].

Although a standard workflow language like Business Process Execution Language (BPEL) [44–47] has been defined, scientific workflow systems often have developed their own workflow model for allowing users to represent workflows. Other than BPEL, other formalisms like UML, Petri nets [48, 49], and XML-based languages [50, 51] are used to express workflows. This feature makes difficult the sharing of workflow specification and limits interoperability among workflow-based applications developed by using different workflow management systems. Nevertheless, there are some historical reasons for that, as many scientific workflow systems and their workflow models were developed before BPEL existed [52].

This section presents a set of workflow management systems specifically designed for composing and running scientific applications. As we will discuss, such systems are very useful in solving complex scientific problems and offer real solutions for improving the way in which scientific discovery is done in all science domains, from physics, to bioinformatics, engineering, and molecular biology. Solutions they adopted are useful to be taken into account in new WMSs and represent features that workflow systems must include to support users in scientific application development.

Existing workflow models can be grouped roughly into the following two main classes [53].

(i) *Script-like systems*, where workflow descriptions specify workflows by means of a textual programming language that can be described by a grammar in an analogous way to traditional programming languages like Perl, Ruby, or Java. They often have complex semantics and an extensive syntax. These types of descriptions declare tasks and their parameters by a textual specification. Typically data dependencies can be established between them by annotations. These languages contain specific workflow constructs, such as sequence or loops, while do, or parallel constructs in order to build up a workflow. Examples of script workflow descriptions are GridAnt [54] and Karajan [55]. A commonly used script-based approach to

describe workflows, mainly in the business workflow community, is BPEL and its recent version for Web services that builds on IBM's Web Service Flow Language, WSFL.

(ii) *Graphical-based systems*, where workflow models specify the workflow with only a few basic graphical elements that correspond to the graph components such as nodes and edges. Compared with script-based descriptions, graphical-based systems are easier to use and more intuitive for the unskilled user mainly because of their graphical representation: nodes typically represent workflow tasks whereas communications (or data dependencies) between different tasks are represented as links going from one node to another. Workflow systems that support graph-based models often incorporate graphical user interfaces which allow users to model workflows by dragging and dropping graph elements. Purely graph-based workflow descriptions generally utilize DAGs. As mentioned before, directed acyclic graph-based languages offer a limited expressiveness, so that they cannot represent complex workflows (e.g., loops cannot be expressed directly).

3.1. Taverna. Taverna [56, 57] is an open-source Java-based workflow management system developed at the University of Manchester. The primary goal of Taverna is supporting the life sciences community (biology, chemistry, and medicine) to design and execute scientific workflows and support in silico experimentation, where research is performed through computer simulations with models closely reflecting the real world. Even though most Taverna applications lie in the bioinformatics domain, it can be applied to a wide range of fields since it can invoke any web service by simply providing the URL of its WSDL document. This feature is very important in allowing users of Taverna to reuse code (represented as a service) that is available on the internet. Therefore, the system is open to third-part legacy code by providing interoperability with web services.

In addition to web services, Taverna supports the invocation of local Java services (Beanshell scripts), local Java API (API Consumer), R scripts on an R server (Rshell scripts), and imports data from a Cvs or Excel spreadsheet.

The Taverna suite includes the following four tools.

(i) *Taverna Engine*, used for enacting workflows.

(ii) *Taverna Workbench*, a client application that enables users to graphically create, edit, and run workflows on a desktop computer (see Figure 3).

(iii) *Taverna Server*, which enables users to set up a dedicated server for executing workflows remotely.

(iv) *A Command Line Tool*, for a quick execution of workflows from a command prompt.

These tools bring together a wide range of features that make it easier to find, design, execute, and share complex workflows. Such features include

(i) pipelining and streaming of data;

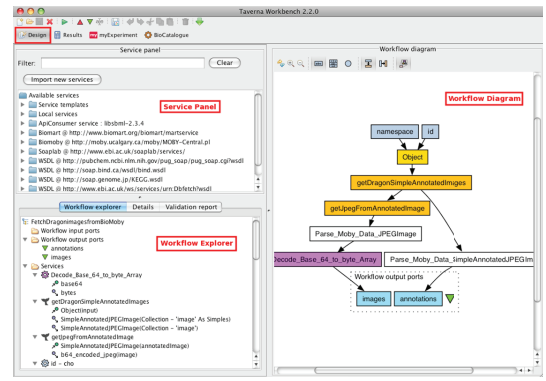

FIGURE 3: A snapshot of the Taverna Workbench.

(ii) implicit iteration of service calls;

(iii) conditional calling of services;

(iv) customizable looping over a service;

(v) failover and retry of service calling;

(vi) parallel execution and configurable number of concurrent threads;

(vii) managing previous runs and workflow results.

In addition, Taverna provides service discovery facilities and integrated support for browsing curated service catalogues, such as the BioCatalogues. Finally, even though Taverna was originally designed for accessing web services and local processes, it can also be used for accessing high-performance computing infrastructures through the use of the TavernaPBS plugin, developed at the University of Virginia, which allows a user to define workflows running on a cluster that uses a PBS queuing system.

3.2. Triana. Triana [58–60] is a Java-based scientific workflow system, developed at the Cardiff University, which combines a visual interface with data analysis tools. It can connect heterogeneous tools (e.g., web services, Java units, and JXTA services) in one workflow. Triana uses its own custom workflow language, although it can use other external workflow language representations such as BPEL, which are available through pluggable language readers and writers. Triana comes with a wide variety of built-in tools for signal-analysis, image manipulation, desktop publishing, and so forth and has the ability for users to easily integrate their own tools.

The Triana framework is based on a modularized architecture in which the GUI connects to a Triana engine, called Triana Controlling Service (TCS), either locally or remotely. A client may log into a TCS, remotely compose and run an application, and then visualize the result locally. Application can also be run in batch mode; in this case, a client may periodically log back to check the status of the application.

The Triana GUI, see Figure 4, includes a collection of toolboxes containing a set of Triana components, and a workspace where the user can graphically define the required application behavior as a workflow of components. Each

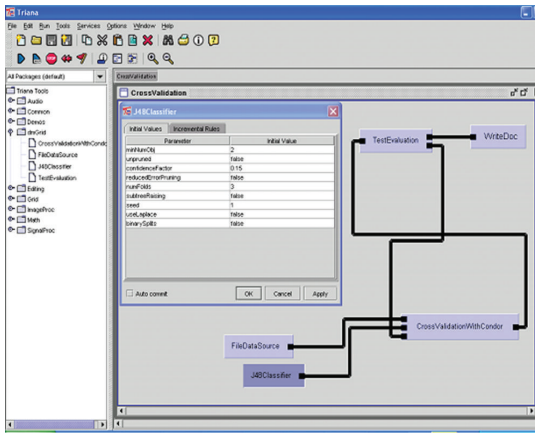

Figure 4: A snapshot of the Triana GUI used to compose a data analysis workflow.

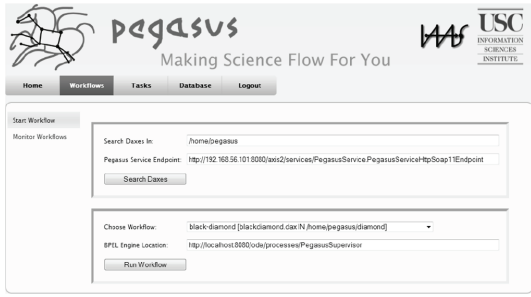

Figure 5: The Pegasus scientific workflow main menu.

component includes information about input and output data type, and the system uses this information to perform design-time type checking on requested connections in order to ensure data compatibility between components. Triana provides several workflow patterns, including loops and branches. Moreover, the workflow components are late bound to the services they represent, thus ensuring a highly dynamic behavior.

Beyond the conventional operational usage, in which applications are graphically composed from collections of interacting units, other usages in Triana include using the generalized writing and reading interfaces for integrating third-party services and workflow representations within the GUI.

3.3. Pegasus. The Pegasus system [61], developed at the University of Southern California, includes a set of technologies to execute workflow-based applications in a number of different environments, including desktops, clusters, Grids, and Clouds. Pegasus has been used in several scientific areas including bioinformatics, astronomy, earthquake science, gravitational wave physics, and ocean science.

The Pegasus workflow management system can manage the execution of an application formalized as a workflow by mapping it onto available resources and executing the workflow tasks in the order of their dependencies (see Figure 5). All the input data and computational resources necessary for workflow execution are automatically located by the system. Pegasus also includes a sophisticated error recovery system that tries to recover from failures by retrying tasks or the entire workflow, by remapping portions of the workflow, by providing workflow-level checkpointing, and by using alternative data sources, when possible. Finally, in order for a workflow to be reproduced, the system records provenance information including the locations of data used and produced, and which software was used with which parameters.

The Pegasus system includes the following three main components.

(i) *The Mapper*, which builds an executable workflow based on an abstract workflow provided by the user or generated by the workflow composition system. To this end, this component finds the appropriate software, data, and computational resources required for workflow execution. The Mapper can also restructure the workflow in order to optimize performance and add transformations for data management or to generate provenance information.

(ii) *The Execution Engine*, which executes in appropriate order the tasks defined in the workflow. This component relies on the compute, storage, and network resources defined in the executable workflow to perform the necessary activities.

(iii) *The Task Manager*, which is in charge of managing single workflow tasks, by supervising their execution on local or remote resources.

Recent research activities carried out on Pegasus investigated the system implementation on Cloud platforms and how to manage computational workflows in the Cloud for developing scalable scientific applications [62–65].

3.4. Kepler. Kepler [66] is a Java-based open source software framework providing a graphical user interface and a runtime engine that can execute workflows either from within the graphical interface or from a command line. It is developed and maintained by a team consisting of several key institutions at the University of California and has been used to design and execute various workflows in biology, ecology, geology, chemistry, and astrophysics.

Kepler is based on the concept of *directors*, which dictate the models of execution used within a workflow. Single workflow steps are implemented as reusable actors that can represent data sources, sinks, data transformers, analytical steps, or arbitrary computational steps (see Figure 6). Each actor can have one or more input and output ports, through which streams of data tokens flow, and may have parameters to define specific behavior. Once defined, Kepler workflows can be exchanged using an XML-based formalism.

By default, Kepler actors run as local Java threads. However, the system also allows spawning distributed execution threads through Web and Grid services. Moreover, Kepler supports foreign language interfaces via the Java Native Interface (JNI), which gives the user flexibility to reuse existing analysis components and to target appropriate computational

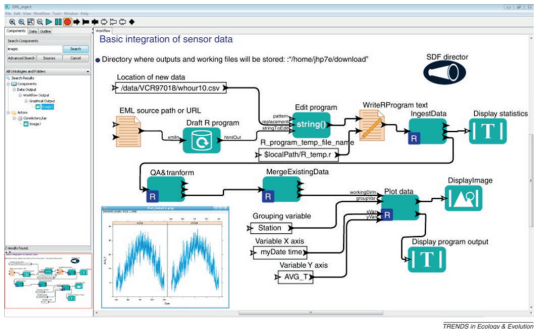

Figure 6: Example of a scientific workflow in Kepler [67].

tools. For example, Kepler includes a Matlab actor and a Python actor.

The Web and Grid service actors allow users to utilize distributed computational resources in a single workflow. The web service actor provides the user with an interface to easily plug in and execute any WSDL-defined web service. Kepler also includes a web service harvester for plugging in a whole set of services found on a web page or in a UDDI repository. A suite of data transformation actors (XSLT, XQuery, Perl, etc.) allows to link semantically compatible but syntactically incompatible web services together.

In addition to standard web services, Kepler also includes specialized actors for executing jobs on a Grid. They include actors for certificate-based authentication (ProxyInit), submitting jobs to a Grid (GlobusJob), as well as for Grid-based data transfer (GridFTP). Finally, Kepler includes actors for database access and querying: DBConnect, which emits a database connection token, to be used by any downstream DBQuery actor that needs it.

3.5. Askalon. Askalon is an application development and runtime environment, developed at the University of Innsbruck, which allows the execution of distributed workflow applications in service-oriented Grids [68]. Its SOA-based runtime environment uses Globus Toolkit as Grid middleware.

Workflow applications in Askalon are described at a high level of abstraction using a custom XML-based language called Abstract Grid Workflow Language (AGWL) [69]. AGWL allows users to concentrate on modeling scientific applications without dealing with the complexity of the Grid middleware or any specific implementation technology such as Web and Grid services, Java classes, or software components. Activities in AGWL can be connected using a rich set of control constructs, including sequences, conditional branches, loops, and parallel sections.

The Askalon architecture (Figure 7) includes a wide set of services.

(i) *Resource broker*, which provides for negotiation and reservation of resources as required to execute a Grid application.

(ii) *Resource monitoring*, which supports the monitoring of Grid resources by integrating existing Grid resource monitoring tools with new techniques (e.g., rule-based monitoring).

(iii) *Information service*, a general-purpose service for the discovery, organization, and maintenance of resource- and application-specific data.

(iv) *Workflow executor*, which supports dynamic deployment, coordinated activation, and fault-tolerant completion of activities onto remote Grid nodes.

(v) *Metascheduler*, which performs a mapping of individual or multiple workflow applications onto the Grid.

(vi) *Performance prediction*, a service for the estimation of the execution time of atomic activities and data transfers, as well as of Grid resource availability.

(vii) *Performance analysis*, a service that unifies the performance monitoring, instrumentation, and analysis for Grid applications and supports the interpretation of performance bottlenecks.

3.6. Weka4WS. Workflows are widely used in data mining systems to manage data and execution flows associated to complex applications [70]. Weka, one of the most used open-source data mining systems, includes the KnowledgeFlow tool that provides a drag-and-drop interface to compose and execute data mining workflows. Unfortunately, the Weka KnowledgeFlow allows users to execute a whole workflow only on a single computer. On the other hand, most data mining workflows include several independent branches that could be run in parallel on a set of distributed machines to reduce the overall execution time. The Grid Lab of University of Calabria implemented distributed workflow composition and execution in Weka4WS, a framework that extends Weka and its KnowledgeFlow environment to exploit distributed resources available in a Grid using web service technologies [71].

As shown in Figure 8, a workflow in Weka4WS is built through a GUI that offers an easy interface to select data, tasks, and links to connect them [72]. Thus, a workflow can be composed by selecting components from a tool bar, placing them on a layout canvas and connecting them together: each component of the workflow is demanded to carry out a specific step of the data mining process. A user operates on the KnowledgeFlow GUI to build a data mining workflow, typically composed by multiple data mining tasks: the local tasks are executed by invoking the local Weka library, while the remote ones are performed through a client module which acts as intermediary between the GUI and the remote web services. Each task is carried out in a thread of its own thus allowing running multiple tasks in parallel.

Weka4WS uses a service-oriented approach in which all the Weka data mining algorithms are wrapped as web services and deployed on Grid nodes. Users can compose and invoke those services in a transparent way by defining data mining workflows as in the original Weka KnowledgeFlow. This approach allows defining task-parallel, distributed data mining applications in an easy and effective way [73].

Task parallelism is a form of parallelism that runs multiple independent tasks in parallel on different processors,

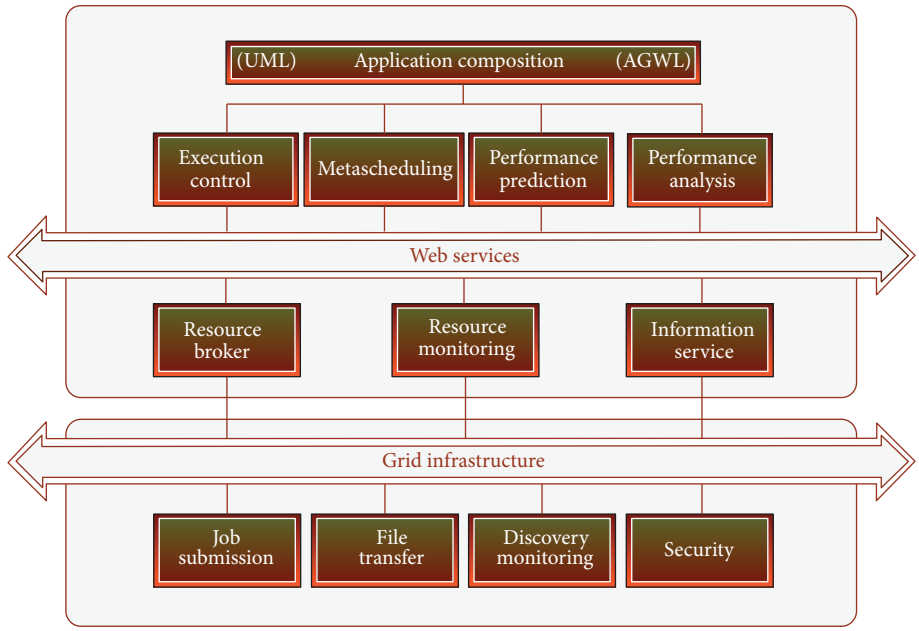

Figure 7: The general architecture of the Askalon system.

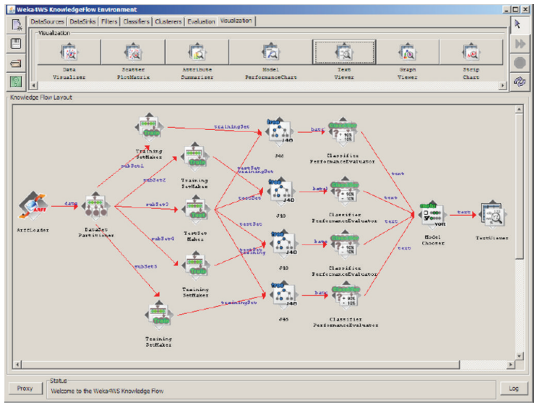

Figure 8: A workflow implementing distributed data mining application in Weka4WS.

available on a single parallel machine or on a set of machines connected through a network like the Grid. Another form of parallelism that can be effectively exploited in data mining workflows is data parallelism, a large data set is split into smaller chunks, each chunk is processed in parallel, and the results of each processing are then combined to produce a single result. Both parallelism forms aim to achieve execution time speedup, and a better utilization of the computing resources, but while task parallelism focuses on running multiple tasks in parallel so that the execution time corresponds to the slowest task, data parallelism focuses on reducing the execution time of a single task by splitting it into subtasks, each one operating on a subset of the original data.

The data-parallel approach is widely employed in distributed data mining as it allows to process very large datasets that could not be analyzed on a single machine due to memory limitations and/or computing time constraints. An example of data-parallel application is distributed classification: the dataset is partitioned into different subsets that are analyzed in parallel by multiple instances of a given classification algorithm; the resulting "base classifiers" are then used to obtain a global model through various selection and combining techniques. Weka4WS through the KnowledgeFlow programming interface supports implementation of workflows that use data parallelism and task parallelism.

3.7. GWES. The General Workflow Execution Service (GWES) [74] (formerly the GridWorkflow Execution Service) is a workflow system that implements a multilevel abstraction and semantic-based solution to facilitate the decoupling between tasks and resources or services. With the implementation of a plugin concept for arbitrary workflow activities, it has now a broader area of application, not limited to the orchestration of Grid and web services. The GWES coordinates the composition and execution process of workflows in arbitrary distributed systems, such as SOA, Cluster, Grid, or Cloud environments.

The GWES processes workflows that are described using the Grid Workflow Description Language (GWorkflowDL), which is based on the formalism of high-level Petri nets (HLPNs) [75]. In the workflow specifications, transitions representing tasks and tokens in the nets represent data flowing through the workflow. Hierarchical Petri nets are also exploited to model hierarchical workflow specifications and to support the multilevel abstraction. An abstract task on top of the hierarchy can be mapped dynamically at runtime by refining the workflow structure.

Figure 9 shows how GWES operates in the automatic mapping of dynamic workflows to determine the appropriate and available resources. User requests are first mapped to

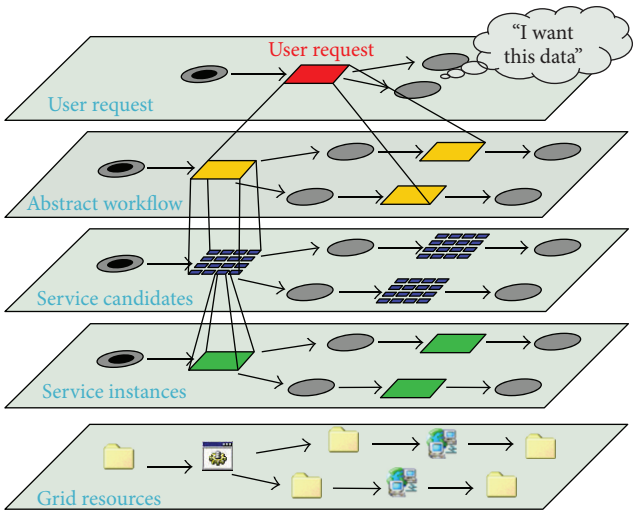

Figure 9: Abstract layers in the GWES workflow refinement process.

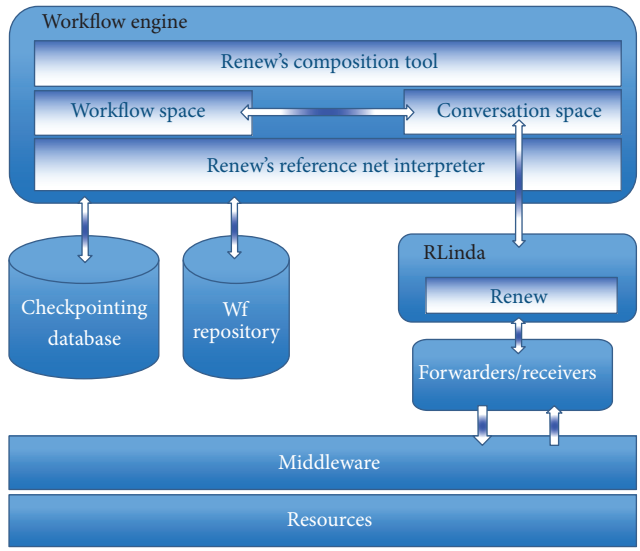

Figure 10: Components of the DVega architecture.

abstract workflows (yellow). Each activity will be initially assigned with the help of the resource matcher service candidates (blue) that provide the appropriate functionality. A scheduler selects one of the candidates (green) and carries out the activity to the appropriate resources. In accordance with the Petri net refinement paradigm, places and transitions within the net can be refined with additional Petri nets, thereby facilitating the modeling of large real-world systems.

However, the GworkflowDL does not support the inherent modeling of the dynamic refinement process itself and, consequently, the workflow structure is modified dynamically by the workflow engine.

This requires to the user a deep knowledge of the workflow engine functionality. For instance, there is no simple construct in the GWorkflowDL. Additionally, GWES also supports exception handling in the hierarchical scientific workflows thereby the workflow engine can modify part of the workflow structure upon a failure, providing great levels of dynamism and flexibility. Nevertheless, GWES does not support a clear separation of exception handling from the application data and control flow, apart from simple rescheduling techniques and checkpoint/restart functionalities.

3.8. DVega. DVega [76, 77] is a scientific workflow engine that adapts itself to the changing availability of resources, minimizing the human intervention. DVega utilizes reference nets [78], a specific class of Petri nets, for composing workflow tasks in a hierarchical way and the Linda [79] communication paradigm for isolating workflow tasks from resources.

Workflow tasks in DVega interact with resources by exchanging messages: in particular workflow tasks send/receive messages to/from Linda and the existing forwarder-receiver components in DVega are responsible for taking the messages from Linda and sending them to the resources and vice versa. DVega's architecture, shown in Figure 10, is completely built upon service-oriented principles and by means of a tuple space shields workflows from the heterogeneity of the middleware.

One of the aspects of the architecture that is worth highlighting is the mapping between tasks and resources. A workflow task receives its inputs, generates a tuple with them, and writes it into the tuple-space (Linda). After that, the workflow task is suspended until the expected result is back: once the tuple is in Linda, the corresponding proxy takes the tuple, transforms it into the suitable format, and forwards it to the destination resource. When the result arrives to the proxy, it is transformed into a message and written into the tuple space. Then, the workflow tasks withdraw the expected tuple containing the result. The main advantage of this approach is that a workflow task will have only to indicate the type of interaction required, without explicitly sending a message to a specific proxy. In consequence, proxies can be added and modified at runtime, without having to stop the execution, and can be shared by workflow tasks.

3.9. Karajan. The Java CoG Kit Karajan system allows users to compose workflows through an XML scripting language as well as with an equivalent more user-friendly language called K [54, 55]. Both languages support hierarchical workflow descriptions based on DAGs and have the ability to use sequential and parallel control structures such as *if*, *while,* and *parallel* in order to easily express concurrency. Moreover, the Karajan workflow framework can support hierarchical workflows based on DAGs including the sequential control structures and parallel constructs, and it can make use of underlying Grid tools such as Globus GRAM [80] for distributed/parallel execution of workflows.

The architecture of the Java CoG Kit Karajan framework, shown in Figure 11, contains the workflow engine that interacts with high-level components, namely, a visualization component that provides a visual representation of the workflow structure and allows monitoring of the execution,

a checkpointing subsystem that allows the checkpointing of the current state of the workflow, and a workflow service that allows the execution of workflows on behalf of a user. A number of convenience libraries enables the workflow engine to access specific functionalities.

Workflow specifications can be visualized in the system. The analysis of dynamism support in Karajan reveals that workflows can actually be modified during runtime through two mechanisms. The first one is through the definition of elements that can be stored in a workflow repository that gets called during runtime. The second one is through the specification of schedulers that support the dynamic association of resources to tasks. The execution of the workflows can either be conducted through the instantiation of a workflow on the user client or can be executed on behalf of the user on a service. Besides, the execution engine of the system also features workflow checkpointing and rollback.

3.10. DIS3GNO. An important benefit of workflows is that, once defined, they can be stored and retrieved for modifications and/or reexecution. This feature allows users to define typical data mining patterns and reuse them in different contexts. We worked in this direction by defining a service-oriented workflow formalism and a visual software environment, named DIS3GNO, to design and execute distributed data mining tasks over the Knowledge Grid [81], a service-oriented framework for distributed data mining on the Grid.

In DIS3GNO a workflow is represented as a directed acyclic graph whose nodes represent resources and whose edges represent the dependencies among the resources [82, 83]. In supporting user to develop applications, DIS3GNO is the user interface for the following two main Knowledge Grid functionalities.

(i) Metadata management: DIS3GNO provides an interface to publish and search metadata about data and tools.

(ii) Design and Execution management: DIS3GNO provides an environment to program and execute distributed data mining applications as service-oriented workflows, through the interaction with the execution management service of the Knowledge Grid.

The DIS3GNO GUI, depicted in Figure 12, has been designed to reflect this twofold functionality. In particular, it provides a panel (on the left) devoted to search resource metadata and a panel (on the right) to compose and execute data mining workflows. In the top-left corner of the window there is a menu used for opening, saving, and creating new workflows, viewing and modifying some program settings, and viewing the previously computed results present in the local file system. Under the menu bar there is a toolbar containing some buttons for the execution control (starting/stopping the execution and resetting the nodes statuses) and other for the workflow editing (creation of nodes representing datasets, tools or viewers, creation of edges, selection of multiple nodes, and deletion of nodes or edges).

Starting from the data mining workflow designed by a user, DIS3GNO generates an XML representation of the corresponding data mining application referred to as conceptual model. DIS3GNO passes the conceptual model to a given execution plan manager, which is in charge of transforming it into an abstract execution plan for subsequent processing by the resource allocation and execution service. This service receives the abstract execution plan and creates a concrete execution plan. To accomplish this task, it needs to evaluate and resolve a set of resources and services and choose those matching the requirements specified by the abstract execution plan. As soon as the resource allocation and execution service has built the concrete execution plan, it is in charge of coordinating its execution by invoking the coordinated execution of services corresponding to the nodes of the concrete execution plan. The status of the computation is notified to the execution plan manager, which in turn forwards the notifications to the DIS3GNO system for visualization.

In this way, DIS3GNO operates as an intermediary between the user and the Knowledge Grid, a service-oriented system for high-performance distributed KDD. All the Knowledge Grid services for metadata and execution management are accessed transparently by DIS3GNO, thus allowing the domain experts to compose and run complex data intensive workflows without worrying about the underlying infrastructure details.

4. Discussion and Research Issues

As we discussed through the previous sections, workflows enable scientists to develop complex simulations and to run faceted applications that are composed of a collection of software components, data sources, web services, and legacy code. In the same cases, such components are designed, developed, and run in collaborative environments.

On large-scale computing infrastructures used today for running large scientific applications, workflow systems provide an abstract representation of concurrent tasks and a distributed virtual machine to execute applications composed of many activities.

To address the big challenges in science and engineering, workflow programming formalisms including adequate abstractions for data representation and concurrent processing orchestration are needed. Besides the amount of data accessed and analyzed by the workflow tasks, the output data produced by each workflow node or stage needs to be stored and annotated with provenance and other metadata to interpret them in the future or reuse them in further executions or new workflows [84–87].

From the WMSs presented in the previous section, we have seen that typically workflow design and execution steps in a distributed scenario are complex and involve multiple stages that are part of the workflow lifecycle. They include

(i) textual or graphical composition,

(ii) mapping of the abstract workflow description onto the available resources,

(iii) scheduling [88], monitoring, and debugging of the subsequent execution.

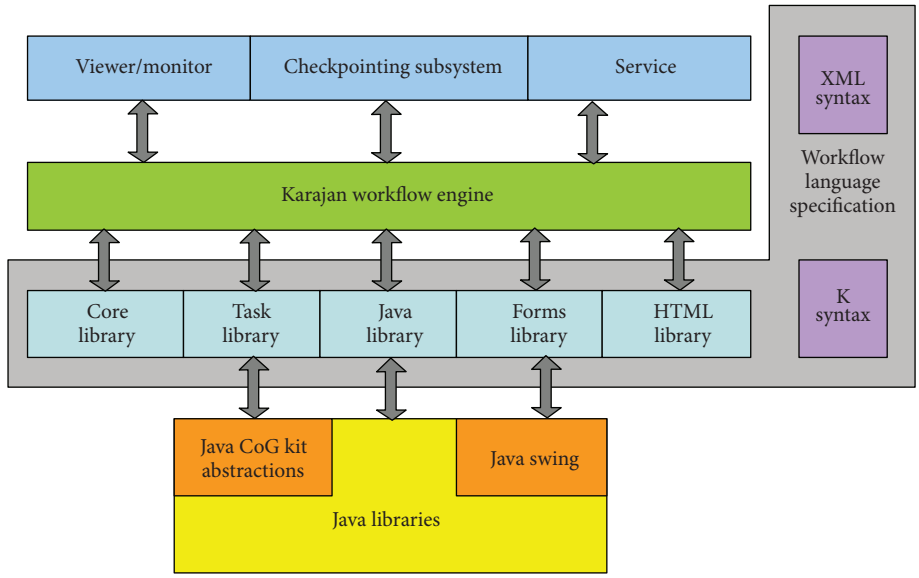

Figure 11: The software architecture of the Karajan workflow management system.

Data-driven applications are more and more developed in science to exploit the large amount of digital data today available [89, 90]. Adequate workflow composition mechanisms then are needed to support the complex workflow management process that includes workflow creation, workflow reuse, and modifications made to the workflow over time.

To address the main research issues and advance the current workflow management systems used for developing scientific applications, several issues are still open and several topics must be investigated to find innovative solutions [91]. Here we give a partial list that could be used as the basis of a research agenda towards the design of the next-generation scientific workflow systems. For each item of this list we introduce the basic aspects and sketch the road towards further investigations.

(i) *Adaptive Workflow Execution Models*. As the computing platforms become more and more dynamic and are based on virtualization, WMSs need to embody techniques for dynamically adapting the execution of workflows on such elastic infrastructures to benefic from their dynamic nature.

(ii) *High-Level Tools and Languages for Workflow Composition*. Whereas in the area of programming languages high-level constructs have been designed and implemented, often the basic building blocks of workflows are simple and regular. This scenario demands for further investigation towards higher and complex abstract structures to be included in workflow programming tools.

(iii) *Scientific Workflow Interoperability and Openness*. Interoperability is a main issue in large-scale experiments where many resources, data, and computing platforms are used. WMSs often are based on proprietary data and ad hoc formats. For this reason the combined use of different systems is inhibited; therefore standard formats and models are needed to support interoperability and ease cooperation among research teams using different tools.

(iv) *Big Data Management and Knowledge Discovery Workflows*. Some systems discussed in Section 3 are mainly devoted to support the implementation of workflows for data analysis and knowledge discovery. The large availability of massive and distributed data sources requires the development of tools and applications for data analysis and to extract useful knowledge hidden in it. In this area, current work [81, 92] shows that workflows are effective, but more research activities are needed to develop higher-level tools and scalable systems.

(v) *Internet-Wide Distributed Workflow Execution*. Typically WMSs run on parallel computers and on single site distributed systems. Recently, by the use of Grids and P2P systems, multisite workflows have been developed and have been executed on a geographical scale. This approach must be extended by at the Internet scale to solve bigger problems and involve several research teams and labs.

(vi) *Service-Oriented Workflows on Cloud Infrastructures*. As discussed in [92], the service-oriented paradigm will allow large-scale distributed workflows to be run on heterogeneous platforms and the integration of workflow elements developed by using different programming languages. Web and Cloud services are a paradigm that can help to handle also workflow interoperability, so this research issue needs a deeper investigation.

(vii) *Workflows Composition and Execution in Exascale Computing Systems*. Exascale computing systems in the next years will become the most interesting scalable platforms

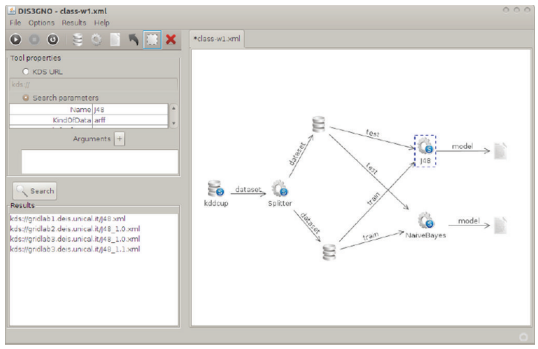

Figure 12: A screenshot of the DIS3GNO GUI composing a data analysis workflow.

where to run a huge amount of concurrent tasks; therefore models, techniques, and mechanisms for efficient execution of workflows on this massive parallel systems are vital.

(viii) *Fault-Tolerance and Recovery Strategies for Scientific Workflows*. Only a few of the scientific WMSs used today provide explicit mechanisms to handle with hardware and/or software failures. However, fault tolerance is important when large-scale or long-lived workflow-based applications are run on distributed platforms. To address those issues both implicit and explicit fault tolerance and recovery mechanisms must be embodied in future WMSs to reduce the impact of faults and to avoid reexecution of not completed workflows due to faults.

(xi) *Workflow Provenance and Annotation Mechanisms and Systems*. Data provenance in workflows is captured as a set of dependencies between data elements [93, 94]. It may be used for interpreting data and providing reproducible results, but also for troubleshooting and optimizing workflow efficiency. Research issues to be investigating in this area are related to efficient techniques to manage provenance data, to visualize and mine those data and improve provenance data interoperability.

5. Conclusions

Workflow systems support research and scientific processes by providing a paradigm that may encompass all the steps of discovery based on the execution of complex algorithms and the access and analysis of scientific data. For instance, in data-driven discovery processes, knowledge discovery tasks can produce results that can confirm real experiments or provide insights that cannot be achieved in laboratories.

Computational science today requires high-performance infrastructures that can be exploited by running on them complex workflows that integrate programs, methods, agents, and services coming from different organizations or sites. In such a way, algorithms, data, services, and other software components are orchestrated in a single virtual framework that specifies the execution sequence and the more appropriate scheduling of this collection of resources [95, 96].

Today the availability of huge amounts of digital data, often referred to as Big Data, originated a sort of data-centric science that is based on the intelligent analysis of large data repositories to extract the rules hidden in the raw data coming for laboratory experiments of physical phenomena [92, 97]. Workflows may help researchers in these discovery tasks by coding in a computational graph an entire scientific practice or methodology that is too complex or impossible to implement in a laboratory.

Moreover, many scientific workflows have to operate over heterogeneous infrastructure with possibility of failures, therefore dealing with such failures in a more coherent way, so that a similar set of techniques can be applied across workflow engines, is another important challenge to be addressed and solved [98–102].

To summarize the paper contribution, basic concepts of scientific workflows and workflow system tools and frameworks used today for the implementation of application in science and engineering have been introduced and discussed. Finally, the paper reported on a selection of workflow systems largely used for solving scientific problems and discussed open issues and research challenges in the area.

References

[1] Workflow Management Coalition, *Terminology and Glossary*, Document Number WFMC- TC-1011, Issue 3.0, 1999.

[2] D. Hollingsworth, *Workflow Management Coalition Specification: The Workflow Reference Model*, Document Number TC00-1003, v. 1.1, 1995.

[3] S. Jablonski and C. Bussler, *Workflow Management: Modeling Concepts, Architecture and Implementation*, Thomson International Computer Press, 1996.

[4] P. Grefen and R. N. Remmerts De Vries, "A reference architecture for workflow management systems," *Data and Knowledge Engineering*, vol. 27, no. 1, pp. 31–57, 1998.

[5] L. Liu, C. Pu, and D. D. Ruiz, "A systematic approach to flexible specification, composition, and restructuring of workflow activities," *Journal of Database Management*, vol. 15, no. 1, pp. 1–40, 2004.

[6] C. Lin and S. Lu, "Architectures of workflow management systems: a survey," Tech. Rep. TR-SWR-01-2008, 2008.

[7] D. Georgakopoulos, M. Hornick, and A. Sheth, "An overview of workflow management: from process modeling to workflow automation infrastructure," *Distributed and Parallel Databases*, vol. 3, no. 2, pp. 119–153, 1995.

[8] S. Bowers, B. Ludaescher, A. Ngu, and 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)*, 2006.

[9] S. S. Sahoo and A. Sheth, "Provenir ontology: towards a framework for eScience provenance management," in *Proceedings of the Microsoft eScience Workshop*, Pittsburgh, Pa, USA, October 2009.

[10] G. Bell, T. Hey, and A. Szalay, "Beyond the data deluge," *Science*, vol. 323, no. 5919, pp. 1297–1298, 2009.

[11] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: an overview of workflow system features and

capabilities," *Future Generation Computer Systems*, vol. 25, no. 5, pp. 528–540, 2009.

[12] M. Sonntag, D. Karastoyanova, and E. Deelman, "Bridging the gap between business and scientific workflows: humans in the loop of scientific workflows," in *Proceedings of the 6th IEEE International Conference on e-Science (eScience '10)*, pp. 206–213, December 2010.

[13] J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *SIGMOD Record*, vol. 34, no. 3, pp. 44–49, 2005.

[14] E. Al-Shakarchi, P. Cozza, A. Harrison et al., "Distributing workflows over a ubiquitous P2P network," *Scientific Programming*, vol. 15, no. 4, pp. 269–281, 2007.

[15] K. Ostrowski, K. Birman, and D. Dolev, "Extensible architecture for high-performance, scalable, reliable publish-subscribe eventing and notification," *International Journal of Web Services Research*, vol. 4, no. 4, pp. 18–58, 2007.

[16] A. Lathers, M. H. Su, A. Kulungowski et al., "Enabling parallel scientific applications with workflow tools," in *Proceedings of the Challenges of Large Applications in Distributed Environments (CLADE '06)*, pp. 55–60, June 2006.

[17] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for e-Science: Scientific Workflows for Grids*, Springer, London, UK, 2007.

[18] M. Mirto, M. Passante, and G. Aloisio, "The ProGenGrid virtual laboratory for bioinformatics," in *Proceedings of the IEEE 25th International Symposium on Computer-Based Medical Systems (CBMS '12)*, 2012.

[19] M. Cannataro, A. Guzzo, C. Comito, and P. Veltri, "Ontology-based design of bioinformatics workflows on PROTEUS," *Journal of Digital Information Management*, vol. 2, no. 1, pp. 87–92, 2004.

[20] W. M. P. Van der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[21] Condor Team, DAGMan: A Directed Acyclic Graph Manager, July 2005, http://research.cs.wisc.edu/htcondor/dagman/.

[22] M. Litzkow, M. Livny, and M. Mutka, "Condor—a hunter of idle workstations," in *Proceedings of the 8th International Conference on Distributed Computing Systems*, pp. 104–111, IEEE Computer Society, New York, NY, USA, 1988.

[23] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger, "Workflow management in Condor," in *Workflows for e-Science*, pp. 357–375, Springer, New York, NY, USA, 2007.

[24] M. Mirto, M. Passante, and G. Aloisio, "A grid meta scheduler for a distributed interoperable workflow management system," in *Proceedings of the IEEE 23rd International Symposium on Computer-Based Medical Systems (CBMS '10)*, pp. 138–143, IEEE Computer Society, Washington, DC, USA, 2010.

[25] L. Zhang, J. Zhang, and H. Cai, *Services Computing*, Springer, 2007.

[26] T. Erl, *Service-Oriented Architecture Concepts, Technology and Design*, Pearson Education, 2005.

[27] M. Vouk and M. Singh, "Quality of service and scientific workflows," in *Proceedings of the Working Conference on Quality of Numerical Software*, pp. 77–89, 1996.

[28] A. Arsanjani, L. J. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, "S3: a service-oriented reference architecture," *IT Professional*, vol. 9, no. 3, pp. 10–17, 2007.

[29] C. Lin, S. Lu, Z. Lai et al., "Service-oriented architecture for VIEW: a visual scientific workflow management system," in *Proceedings of IEEE International Conference on Services Computing (SCC '08)*, pp. 335–342, 2008.

[30] Y. Gil, V. Ratnakar, E. Deelman, G. Mehta, and J. Kim, "Wings for Pegasus: creating large scale scientific applications using semantic representations of computational workflows," in *Proceedings of the 19th Innovative Applications of Artificial Intelligence Conference (IAAI '07)*, 2007.

[31] G. Juve and E. Deelman, "Resource provisioning options for large-scale scientific workflows," in *Proceedings of the 3rd International Workshop on Scientific Workflows and Business Workflow Standards in e-Science*, Indianapolis, Ind, USA, 2008.

[32] A. Tsalgatidou, G. Athanasopoulos, M. Pantazoglou et al., "Developing scientific workflows from heterogeneous services," *SIGMOD Record*, vol. 35, no. 2, pp. 19–25, 2006.

[33] J. A. Miller, D. Palaniswami, A. P. Sheth, K. J. Kochut, and H. Singh, "WebWork: METEOR2's web-based workflow management system," *Journal of Intelligent Information Systems*, vol. 10, no. 2, pp. 185–213, 1998.

[34] G. Alonso, R. Günthör, M. Kamath, D. Agrawal, A. Abbadi, and C. Mohan, "Exotica/FMDC: a workflow management system for mobile and disconnected clients," *Distributed and Parallel Databases*, vol. 4, no. 3, pp. 229–247, 1996.

[35] F. Leymann and D. Roller, "Business process management with FlowMark," in *Proceedings of the IEEE Computer Society International Conference (COMPCON '94)*, pp. 230–234, 1994.

[36] B. Ludäscher, I. Altintas, C. Berkley et al., "Scientific workflow management and the Kepler system," *Concurrency Computation Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[37] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo, "VisTrails: visualization meets data management," in *Proceedings of the ACM International Conference on Management of Data (SIGMOD '06)*, pp. 745–747, June 2006.

[38] Y. Zhao, M. Hategan, B. Clifford et al., "Swift: fast, reliable, loosely coupled parallel computation," in *Proceedings of International Workshop on Scientific Workflows (SWF '07)*, pp. 199–206, July 2007.

[39] S. Majithia, M. Shields, I. Taylor, and I. Wang, "Triana: a graphical web service composition and execution toolkit," in *Proceedings of IEEE International Conference on Web Services (ICWS '04)*, pp. 514–521, July 2004.

[40] A. Chebotko, C. Lin, X. Fei et al., "VIEW: a visual scientific workflow management system," in *Proceedings of International Workshop on Scientific Workflows (SWF '07)*, pp. 207–208, July 2007.

[41] P. Kacsuk and G. Sipos, "Multi-Grid, multi-user workflows in the P-GRADE Grid portal," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 221–238, 2005.

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

[43] T. Glatard, G. Sipos, J. Montagnat, Z. Farkas, and P. Kacsuk, "Workflow-level parametric study support by MOTEUR and the P-GRADE portal," in *Workflows for e-Science*, pp. 279–299, Springer, New York, NY, USA, 2007.

[44] Web Services Business Process Execution Language, Version 2.0, Primer, May 2007, https://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm.

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

[46] W. Emmerich, B. Butchart, L. Chen, B. Wassermann, and S. L. Price, "Grid service orchestration using the Business Process Execution Language (BPEL)," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 283–304, 2005.

[47] M. B. Juric, *Business Process Execution Language for Web Services BPEL and BPEL4WS*, Packt Publishing, 2nd edition, 2006.

[48] M. Alt, A. Hoheisel, H.-W. Pohl, and S. Gorlatch, "A grid workflow language using high-level petri nets," in *Proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics (PPAM '05)*, R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Wasniewski, Eds., vol. 3911 of *Lecture Notes in Computer Science*, pp. 715–722, Springer, New York, NY, USA, 2006.

[49] Z. Guan, F. Hern, and P. Bangalore, *Grid-flow: a grid-enabled scientific workflow system with a petri net-based interface [Ph.D. thesis]*, University of Alabama at Birmingham, 2006.

[50] M. Atay, A. Chebotko, D. Liu, S. Lu, and F. Fotouhi, "Efficient schema-based XML-to-Relational data mapping," *Information Systems*, vol. 32, no. 3, pp. 458–476, 2007.

[51] A. Chebotko, M. Atay, S. Lu, and F. Fotouhi, "XML subtree reconstruction from relational storage of XML documents," *Data and Knowledge Engineering*, vol. 62, no. 2, pp. 199–218, 2007.

[52] T. Andrews, F. Curbera, H. Dholakia et al., "Business process execution language for web services," version 1.1, 2003.

[53] F. Neubauer, A. Hoheisel, and J. Geiler, "Workflow-based Grid applications," *Future Generation Computer Systems*, vol. 22, no. 1-2, pp. 6–15, 2006.

[54] G. von Laszewski and M. Hategan, "Java CoG Kit Karajan/Gridant workflow guide," Tech. Rep., Argonne National Laboratory, Argonne, Ill, USA, 2005.

[55] G. von Laszewski, M. Hategan, and D. Kodeboyina, "Java CoG kit workflow," in *Workflows for e-Science*, pp. 143–166, Springer, New York, 2007.

[56] T. Oinn, M. Addis, J. Ferris et al., "Taverna: a tool for the composition and enactment of bioinformatics workflows," *Bioinformatics*, vol. 20, no. 17, pp. 3045–3054, 2004.

[57] T. Oinn, M. Greenwood, M. Addis et al., "Taverna: lessons in creating a workflow environment for the life sciences," *Concurrency Computation Practice and Experience*, vol. 18, no. 10, pp. 1067–1100, 2006.

[58] I. Taylor, M. Shields, I. Wang, and O. Rana, "Triana, applications within Grid computing and peer to peer environments," *Journal of Grid Computing*, vol. 1, pp. 199–217, 2004.

[59] I. Taylor, M. Shields, I. Wang, and A. Harrison, "Visual grid workflow in Triana," *Journal of Grid Computing*, vol. 3, no. 3-4, pp. 153–169, 2005.

[60] I. Taylor, E. Al-Shakarchi, and S. D. Beck, "Distributed audio retrieval using Triana, DART," in *Proceedings of the International Computer Music Conference (ICMC '06)*, pp. 716–722, New Orleans, Lo, USA, November 2006.

[61] E. Deelman, G. Singh, M. H. Su et al., "Pegasus: a framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[62] A. Nagavaram, G. Agrawal, M. Freitas, G. Mehta, R. Mayani, and E. Deelman, "A cloud-based dynamic workflow for mass spectrometry data analysis," in *Proceedings of the 7th IEEE International Conference on e-Science (e-Science '11)*, December 2011.

[63] J. S. Vöckler, G. Juve, E. Deelman, M. Rynge, and B. Berriman, "Experiences using cloud computing for a scientific workflow application," in *Proceedings of the 2nd International Workshop on Scientific Cloud Computing (ScienceCloud '11)*, pp. 15–24, June 2011.

[64] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow wnsembles in IaaS clouds," in *Proceedings of the 24th IEEE/ACM International Conference on Supercomputing (SC '12)*, 2012.

[65] G. Juve, E. Deelman, B. Berriman, B. P. Berman, and P. Maechling, "An evaluation of the cost and performance of scientific workflows on Amazon EC2," *Journal of Grid Computing*, vol. 10, no. 1, pp. 5–21, 2012.

[66] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows," in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM '04)*, pp. 423–424, IEEE Computer Society, New York, NY, USA, 2004.

[67] J. H. Porter, P. C. Hanson, and C.-C. Lin, "Staying afloat in the sensor data deluge," *Trends in Ecology and Evolution*, vol. 27, no. 2, pp. 121–129, 2012.

[68] T. Fahringer, R. Prodan, R. Duan et al., "ASKALON: a grid application development and computing environment," in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pp. 122–131, November 2005.

[69] T. Fahringer, Q. Jun, and S. Hainzer, "Specification of Grid workflow applications with AGWL: an abstract Grid workflow language," in *Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGrid '05)*, pp. 676–685, May 2005.

[70] A. Congiusta, G. Greco, A. Guzzo et al., "A data mining-based framework for grid workflow management," in *Proceedings of the 5th International Conference on Quality Software (QSIC '05)*, pp. 349–356, IEEE Computer Society Press, September 2005.

[71] D. Talia, P. Trunfio, and O. Verta, "Weka4WS: a WSRF-enabled Weka toolkit for distributed data mining on Grids," in *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pp. 309–320, Porto, Portugal, 2005.

[72] M. Lackovic, D. Talia, and P. Trunfio, "A framework for composing knowledge discovery workflows in grids," in *Foundations of Computational Intelligence*, A. Abraham, A. Hassanien, A. Carvalho, and V. Snášel, Eds., vol. 6 of *Data Mining Theoretical Foundations and Applications, Studies in Computational Intelligence*, pp. 345–369, Springer, 2009.

[73] D. Talia, P. Trunfio, and O. Verta, "The Weka4WS framework for distributed data mining in service-oriented Grids," *Concurrency Computation Practice and Experience*, vol. 20, no. 16, pp. 1933–1951, 2008.

[74] A. Hoheisel, "User tools and languages for graph-based Grid workflows," *Concurrency Computation Practice and Experience*, vol. 18, no. 10, pp. 1101–1113, 2006.

[75] K. Jensen and G. Rozenberg, *High-Level Petri Nets: Theory and Application*, Springer, London, UK, 1991.

[76] R. Tolosana-Calasanz, J. A. Banares, O. F. Rana, P. Á lvarez, J. Ezpeleta, and A. Hoheisel, "Adaptive exception handling for scientific workflows," *Concurrency Computation Practice and Experience*, vol. 22, no. 5, pp. 617–642, 2010.

[77] R. Tolosana-Calasanz, J. A. Bañares, P. Álvarez, J. Ezpeleta, and O. Rana, "An uncoordinated asynchronous checkpointing

model for hierarchical scientific workflows," *Journal of Computer and System Sciences*, vol. 76, no. 6, pp. 403–415, 2010.

[78] R. Valk, "Petri nets as token objects: an introduction to elementary object nets," in *Proceedings of the 19th International Conference on Application and Theory of Petri Nets (ICATPN '98)*, J. Desel and M. Silva, Eds., vol. 1420 of *Lecture Notes in Computer Science*, pp. 1–25, Springer, Lisbon, Portugal, June 1998.

[79] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, 1985.

[80] M. Feller, I. Foster, and S. Martin, "GT4 GRAM: a functionality and performance study," in *Proceedings of the TERAGRID Conference*, Madison, Wis, USA, 2007.

[81] D. Talia and P. Trunfio, "How distributed data mining tasks can thrive as knowledge services," *Communications of the ACM*, vol. 53, no. 7, pp. 132–137, 2010.

[82] E. Cesario, M. Lackovic, D. Talia, and P. Trunfio, "Programming knowledgediscovery workflows in service-oriented distributed systems," *Concurrency and Computation: Practice and Experience*. In press.

[83] E. Cesario, M. Lackovic, D. Talia, and P. Trunfio, "Service-oriented data analysis in distributed computing systems," in *High Performance Computing: From Grids and Clouds to Exascale*, I. Foster, W. Gentzsch, L. Grandinetti, and G. Joubert, Eds., pp. 225–245, IOS Press, Lansdale, Pa, USA, 2011.

[84] A. Chebotko, X. Fei, C. Lin, S. Lu, and F. Fotouhi, "Storing and querying scientific workflow provenance metadata using an RDBMS," in *Proceedings of the 3rd IEEE International Conference on E-Science and Grid Computing*, pp. 611–618, December 2007.

[85] I. Altintas, O. Barney, and E. Jaeger-Frank, "Provenance collection support in the kepler scientific workflow system," in *Proceedings of the International Provenance and Annotation Workshop (IPAW '06)*, pp. 118–132, 2006.

[86] E. Deelman and A. Chervenak, "Data management challenges of data-intensive scientific workflows," in *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID '08)*, pp. 687–692, May 2008.

[87] Open Provenance Model, 2009, http://twiki.ipaw.info/bin/view/OPM/WebHome.

[88] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Scientific Programming*, vol. 14, no. 3-4, pp. 217–230, 2006.

[89] C. Anderson, "The end of theory: the data deluge makes the scientific method obsolete," *Wired*, vol. 16, no. 7, 2008.

[90] G. Erbach, "Data-centric view in e-Science information systems," *Data Science Journal*, vol. 5, pp. 219–222, 2006.

[91] Y. Gil, E. Deelman, M. Ellisman et al., "Examining the challenges of scientific workflows," *Computer*, vol. 40, no. 12, pp. 24–32, 2007.

[92] D. Talia and P. Trunfio, *Service-Oriented Distributed Knowledge Discovery*, CRC Press, Boca Raton, Fla, USA, 2012.

[93] R. Bose and J. Frew, "Lineage retrieval for scientific dataprocessing: a survey," *ACM Computing Surveys*, vol. 37, no. 1, pp. 1–28, 2005.

[94] L. Moreau, *Concurrency and Computation: Practice and Experience*, Special Issue on the First Provenance Challenge, 2008.

[95] C. Lin, S. Lu, X. Fei et al., "A reference architecture for Scientific workflow management systems and the VIEW SOA solution," *IEEE Transactions on Services Computing*, vol. 2, no. 1, pp. 79–92, 2009.

[96] Y. Han, A. Sheth, and C. Bussler, "A taxonomy of adaptive workflow management," in *Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '98)*, Seattle, Wash, USA, 1998.

[97] T. Hey, S. Tansley, and K. Tolle, *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Microsoft Research, Redmond, Wash, USA, 2009.

[98] M. Lackovic, D. Talia, R. Tolosana-Calasanz, J. A. Bañares, and O. F. Rana, "A taxonomy for the analysis of scientific workflow faults," in *Proceedings of the 2nd International Workshop on Workflow Management in Service and Cloud Computing (WMSC '10), in conjunction with (CSE '10)*, pp. 398–403, December 2010.

[99] L. Ramakrishnan, D. Nurmi, A. Mandal et al., "VGrADS: enabling e-Science workflows on grids and clouds with fault tolerance," in *Proceedings of the ACM/IEEE International Conference on High Performance Computing and Communication (SC '09)*, November 2009.

[100] T. Samak, D. Gunter, M. Goode, E. Deelman, G. Juve, and F. Silva, "Failure analysis of distributed scientific workflows executing in the cloud," in *Proceedings of the 8th International Conference on Network and Service Management (CNSM '12)*, 2012.

[101] M. Kamath and K. Ramamritham, "Failure handling and coordinated execution of concurrent workflows," in *Proceedings of the 14th International Conference on Data Engineering (ICDE '98)*, pp. 334–341, IEEE Computer Society Washington, February 1998.

[102] R. Tolosana-Calasanz, M. Lackovic, O. Rana, J. Banares, and D. Talia, "Characterizing quality of resilience in scientific workflows," in *Proceedings of the 6th Workshop on Workflows in Support of Large-Scale Science*, pp. 117–126, ACM, New York, NY, USA, November 2011.