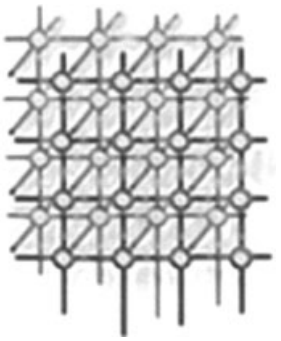
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (in press)
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1235



Automatic capture and efficient storage of e-Science experiment provenance

Roger S. Barga[1,*,†] and Luciano A. Digiampietri[2]

[1]*Microsoft Research, One Microsoft Way Redmond, WA 98052, U.S.A.*
[2]*Institute of Computing, University of Campinas, Sao Paulo, Brazil*

SUMMARY

For the first provenance challenge, we introduce a layered model to represent workflow provenance that allows navigation from an abstract model of the experiment to instance data collected during a specific experiment run. We outline modest extensions to a commercial workflow engine so it will automatically capture provenance at workflow runtime. We also present an approach to store this provenance data in a relational database. Finally, we demonstrate how core provenance queries in the challenge can be expressed in SQL and discuss the merits of our layered representation. Copyright © 2007 John Wiley & Sons, Ltd.

Received 28 November 2006; Revised 18 April 2007; Accepted 1 May 2007

KEY WORDS: provenance model; automatic provenance capture; efficient provenance storage

INTRODUCTION

Scientific workflows are now recognized as a crucial element of the 'cyberinfrastructure', facilitating e-Science. Typically sitting on top of a middleware layer, scientific workflows are a means by which scientists can model, design, execute, debug, re-configure and re-run their analysis and visualization pipelines. Part of the established scientific method is to create a record of the origins of a result, how it was obtained, experimental methods used, machine calibrations and parameters, etc. It is the same in e-Science, except provenance data are a record of the workflow activities invoked, services and databases accessed, data sets used, and so forth. Such information is useful for a scientist to interpret their workflow results and for other scientists to establish trust in the experimental result.

*Correspondence to: Roger S. Barga, Microsoft Research, One Microsoft Way Redmond, WA 98052, U.S.A.
†E-mail: barga@microsoft.com

In this paper, we describe how a workflow enactment engine can assume responsibility for generating workflow provenance automatically during runtime [1]. We argue that a single representation cannot satisfy all provenance queries. In some cases, it is suitable to provide an abstract description of the scientific process that took place, without describing specific codes executed, data sets or remote services invoked. In other cases, a precise description of the execution is exactly what is required to explain a result, including all operational details such as parameters and machine configurations. We present a provenance model that supports multiple levels of representation [2,3], which allows users to navigate from the abstract levels into lower levels and back again, and permits scientists to specify exactly what they wish to share from their experiment.

We are developing a system called REDUX to explore the benefits and challenges of automatically capturing experiment provenance, along with methods to store efficiently the resulting provenance data. REDUX will enable scientists to recall or restore provenance data to understand or explain an experiment and establish trust in the result. REDUX uses the Windows Workflow Foundation (WinWF) [4] as a workflow engine. This paper presents a current snapshot of our project, including an overview of our model for experiment provenance, implementation in a commercial workflow system, and techniques to store provenance data efficiently in a relational database. Our system addressed the challenges presented in the first provenance challenge (FPC) [5] and was able to answer all the proposed queries. Moreover a comparison with the approaches of the other participating teams allowed us to identify several ways in whose our system can be improved. This discussion can be found in sections Related Work and Discussion.

Clearly, we cannot present a complete system description in this paper; however, there are a number of concrete contributions to report on at this point in time. We present our model for workflow execution provenance, which consists of layers from an abstract description of the workflow (experiment design) to a execution details from an experimental run. We describe the storage of provenance data generated from our model on a relational database system. To make the storage of provenance data economically feasible, we identify properties of provenance data represented by our model that can significantly reduce the amount of storage required.

WORKFLOW EXECUTION PROVENANCE MODEL

In this section, we first motivate and then describe our model for representing result provenance captured during workflow execution.

Different kinds of result provenance

We frame our discussion using the provenance challenge workflow example. A scientist is at the center of an experiment as it unfolds, possibly over a period of days or weeks, interacting with the executing processes. This scientist may alter the parameters of *align_warp*, navigate between databases available online to issue queries, inject 'shim' code as needed (data transformations) in order to get the output from one step (workflow activity) to feed into the next step (workflow activity). The experiment (workflow) is specified in the abstract as a schedule of activities (warp an input image, followed by some other image transformations, followed by a image slicer activity, followed by a user-defined image converter), then instantiated with concrete services align_warp → reslice

DOI: 10.1002/cpe

→ softmean → slicer → convert) and invoked with specific parameters at runtime by the scientist (align_warp version 5 with parameters `-m 12` and `-q` → reslice version 5 without parameters → softmean with parameter `y null` → each instance of slicer with parameters, respectively, `-x .5`; `-y .5`; and `-z .5` → convert without parameters, producing a gif image), and dynamically altered if the scientist observes an anomaly (he used wrong reference images as input of align_warp, or he wants to change the value of a parameter, so the scientist alters some of the parameters and reruns the activity before continuing on).

Later, perhaps after several weeks, the scientist may ask the following questions about an experimental result:

- Which version of *align_warp* did I use?
- What codes (activities) did I invoke during workflow execution, and what were the parameters?
- What machine was used to execute the *softmean*?

A simple *derivation path* that records the workflow execution, identifying workflow instance, input variables and runtime parameters, would suffice to answer these provenance queries. However, other information can be associated with a result to provide additional context that more accurately explains the experiment execution and its result. Consider the following provenance queries:

- What experiments utilized *image converter* tools?
- Were any steps skipped in this experiment? Were any shims (data transformations) inserted?
- Did the experiment design differ between these results, and if so what were the differences?
- Are there branches in the experiment that have not been explored yet (executed) by some experiment?

These queries illustrate the need for provenance information captured at the *experiment design level* rather than simply the derivation path level. Both types of provenance information communicate context necessary to understand or explain an experiment and establish trust in the result, but how this information is captured, represented and queried is likely to be quite different. To capture the experimental process, we must adopt a more general notion of result provenance.

We argue the complete provenance for an experimental result is captured by a set of objects to which it is incrementally bound. This process begins with an abstract description of the workflow (experiment), before it is bound to specific services, tools, data sets, etc., and continues until the execution of the experiment. Intuitively, our layered provenance model holds some of these objects fixed while varying others. To capture flexibly the degrees of freedom in which an experiment can vary, without needlessly duplicating objects that stay fixed, we use *binding levels* to model our observation that experimental design (abstract workflow), a workflow instance, and even individual processing steps (workflow activities) acquire their identity in stages. In section Different Kinds of Result Provenance, we present our layered model for result provenance, in which each level represents a distinct stage of binding.

A layered model for result provenance

The first level in our model, illustrated in Figure 1, represents an abstract description of the experiment. This layer captures *abstract activities* in the workflow and *links/relationships* among the activities or ports of activities, where an abstract activity is a *class* of activities. The provenance

DOI: 10.1002/cpe

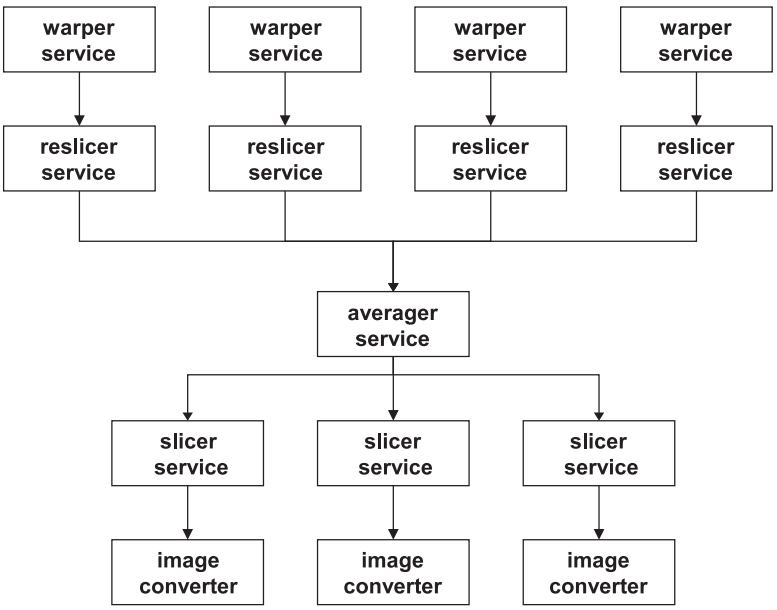


Figure 1. L0: abstract service descriptions.

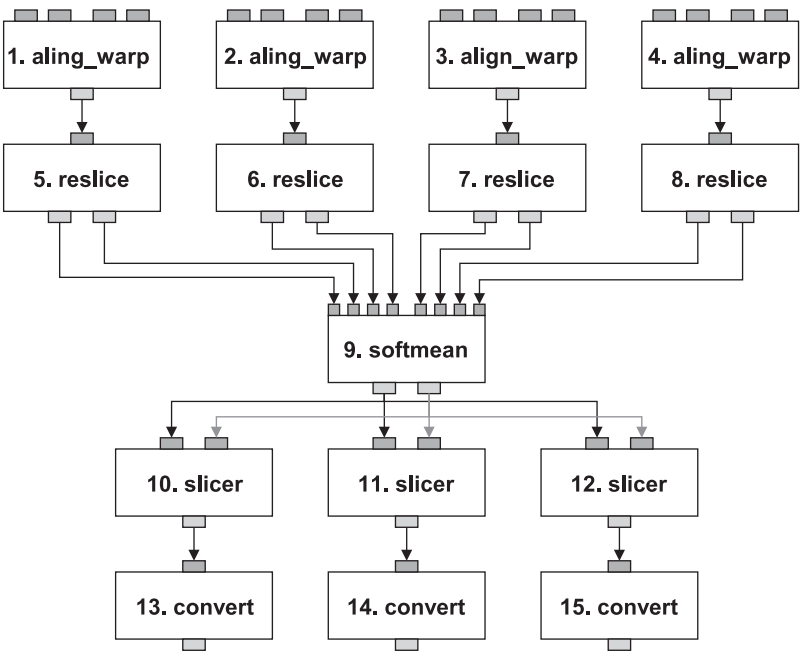


Figure 2. L1: service instantiation descriptions.

DOI: 10.1002/cpe

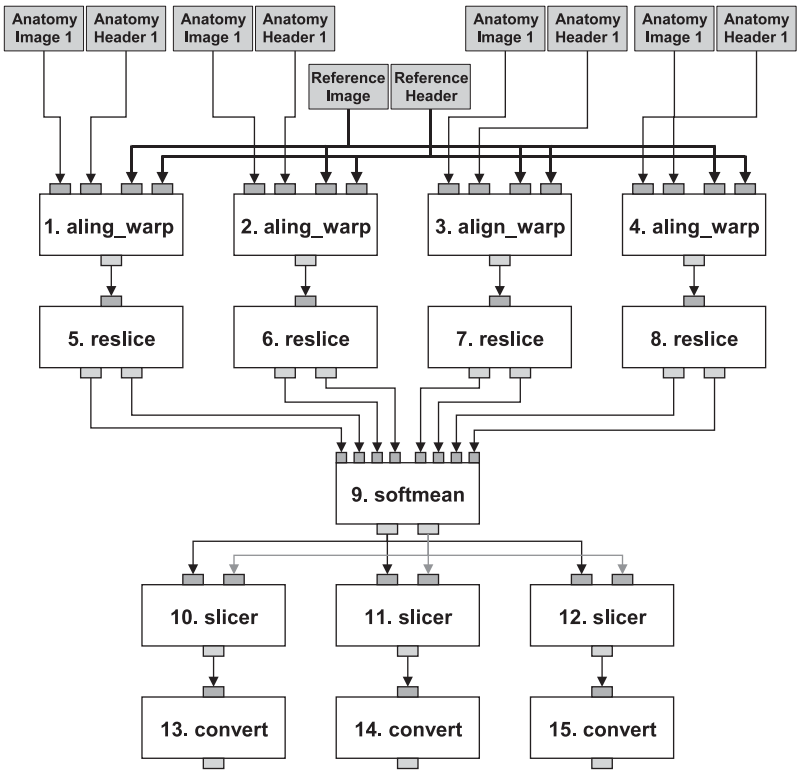


Figure 3. L2: data instantiation of workflow.

data for L0 support general queries, such as: *what experiments in my collection use web services?* In general, level L0 describes the design space workflow instances of the abstract model.

The second level in our model, illustrated in Figure 2, represents an instance of the abstract model. This layer captures bindings or *instances of activities* and additional relationships, as classes of activities are instantiated. This level refines abstract activities specified in the previous level with specific instances (for example, the abstract activity *Image Converter tool* is bound to the instance *convert tool*). From the information captured in Level L1 a user can pose provenance queries about specific activities: *What experiments (workflows) used the service convert?*

The next level of our model L2, illustrated in Figure 3, represents the execution of the workflow instance specified in level L1. From level L1 to L2, the model captures information provided at runtime that completes the specification of activities, including input data, parameters supplied at runtime, branches taken, activities inserted or skipped during execution, etc. Information captured at this level is sufficient to trace the execution of the workflow. In level L2, data sets and parameters are captured as the workflow is executed. From this, we can pose a number of queries on how the experiment was actually carried out. Such queries can involve parameters, data values supplied at runtime, which activities were inserted or skipped, etc. Moreover, if the workflow enactment engine

DOI: 10.1002/cpe

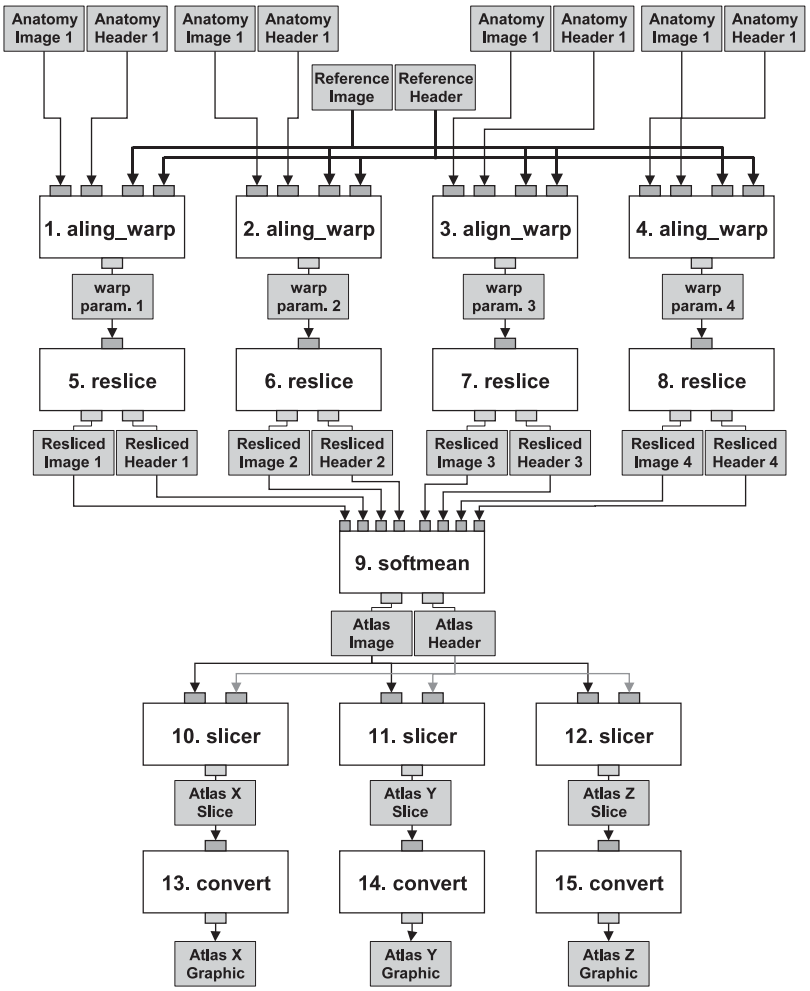


Figure 4. L3: run-time provenance from execution.

provides the necessary support, this provides sufficient information to serve as a redo log for the smart replay of the experiment. Our smart replay has functionality that allows the user to select what activities will be re-executed and what activities will use the stored result (produced in the previous execution of the workflow, stored in L3).

The final level of our model L3, illustrated in Figure 4, represents runtime specific information. Here, we capture operational details, such as the start and end time of workflow execution, start and end time of individual activity execution, status codes and intermediate results, information about the internal state of each activity, along with information about the machines to which each activity was assigned for execution.

DOI: 10.1002/cpe

Level L3, the runtime operational level, can contain a diversity of information. The kind of information to be stored will depend of the goals of the user. For example, in a system that utilizes a high-performance grid, the user may wish to record the processor in the grid on which the job was executed and when execution occurred. In other systems, the user may wish to store information about the internal status of each activity. These examples are supported by our provenance model, but they require cooperation from external software. In the grid example, the scheduler must send job information to the provenance system, while in the second example each activity must send activity status information to the provenance system.

Provenance model summary

To sum up, the benefits a multilayered model can bring are *explanation*, *validation*, *insight*, and *control*. Having an abstract model of the science being undertaken and being able to use this model to interpret behavior (services and data sets used), together make it possible to explain or reason about an experiment. For validation, knowing which individual activities were executed, identifying branches taken during execution, steps skipped or inserted, and parameters supplied at runtime are essential to establish trust in a result. The ability to compare a set of related experiment executions against an abstract workflow model, to identify common patterns or options not yet explored, can be used to gain insight into authoring new experimental designs. Finally, a layered model offers the scientist control over information they wish to share. A related benefit discussed later in the paper is optimization—a layered or factored model can expose opportunities to store efficiently provenance traces in a storage manager.

Storing provenance data

In this section, we consider how to store experimental result provenance. We believe a database management system is a promising candidate. A database stores data reliably, allows indices to be built on top of provenance data, and has a rich query interface that can be used to pose provenance queries. And, since the database is a separate entity from the workflow management system, the persistence of the result provenance is independent of the experiment (workflow).

This leads to an obvious question—*what database schema is best suited for storing result provenance?* To answer this, we first consider typical provenance queries, using queries from the FPC [5,6]. Then we consider advantages and disadvantages of various schemas to record provenance. Finally, we consider issues related to efficiently storing provenance.

Result provenance schema discussion

Our provenance schema design was driven in part by considering provenance queries, in particular queries from the FPC. The goal of the challenge was to foster understanding the expressiveness of the capabilities of provenance-related systems, including how each system answers a set of nine provenance queries. Some of the provenance queries in the challenge [6] are as follows:

FPC #4: Find all invocations of procedure *align_warp* using a twelfth order nonlinear 1365 parameter model that ran on a Monday.

DOI: 10.1002/cpe

FPC #6: Find all output averaged images of *softmean* procedures, where the warped images taken as input were *align_warped* using a 12th order nonlinear 1365 parameter model, i.e. where *softmean* was preceded in the workflow by *align_warp* procedure with argument -m 12.

FPC #8: A user has annotated anatomy images with a key–value pair *center* = *UChicago*. Find the outputs of the *align_warp* activity where the inputs were annotated with *center* = *UChicago*.

Based on these queries, we can see the result provenance data that need to be recorded can be broadly classified into the following two categories:

Dependency provenance: This refers to provenance used to record the fact that one computational step, or activity, either *depends on*, or was *derived from*, information computed in another step. For example, *align_warp* precedes *softmean*.

Annotation provenance: Annotations are assertions made by users regarding an object. For example, a user can annotate a workflow activity or data set with a summary of its origins or perceived quality.

Dependency provenance can be used to answer the FPC queries FPC #4 and FPC #6 presented above, while annotation provenance can be used to answer query FCP #8; other queries from the challenge fit into one of these two categories as well.

In our system, REDUX, provenance for an experiment is captured by a set of distinct levels, to which the result is incrementally bound. The model, in particular Levels L2 and L3, capture dependency provenance, so the question is how to best store these provenance data in the database. Provenance data from a single experiment can be stored in the following ways:

- *Single provenance relation*: In this method, a single relation is maintained for each level of our provenance model. As the experiment unfolds, a dependency relationship is established between separate levels and the provenance of both levels is retrieved from the database, updated and stored back into the database.
- *Multiple provenance relations*: In this method, multiple relations are maintained for each level, and keys are used to maintain the relationship between two relations. A new relation is added to an experimental result provenance when, for example, a service is bound to a workflow, or an activity is executed, etc. To illustrate, we present the schema for Level L0, the abstract workflow provenance level, in Figure 5, Part 1.

In the first schema, abstract activities are stored in the *ActivityType* relation. Predefined properties, such as proxy for *InvokeWebService* are linked in the *ActivityType_Property* relation (this relation links *Property* and *ActivityType*). Each property has a *DataType*, which can be a basic data type or semantic type. The *WorkflowType* contains basic kinds of workflows, such as sequential, state machine, event driven, etc. A tuple in the relation *AbstractWorkflow_ActivityType* represents a different workflow activity (of some *ActivityType*) that belongs to the *AbstractWorkflow*. This schema is capable of representing all abstract workflows (experiment design) that WinWF [4] can implement.

Figure 5, Part 2 illustrates the schemas that represent Level L2 (executable workflow). Level L2 is connected to Level L1 through foreign keys: *WorkflowModelId* from the relation *ExecutableWorkflow* and *WorkflowModel_ActivityId* from the relation *ExecutableWorkflow_ExecutableActivity*. In Level L2, we record two kinds of information: (i) assignment of actual values to activities and (ii) assignment of input values and parameters. With this information, the workflow is ready to be

DOI: 10.1002/cpe

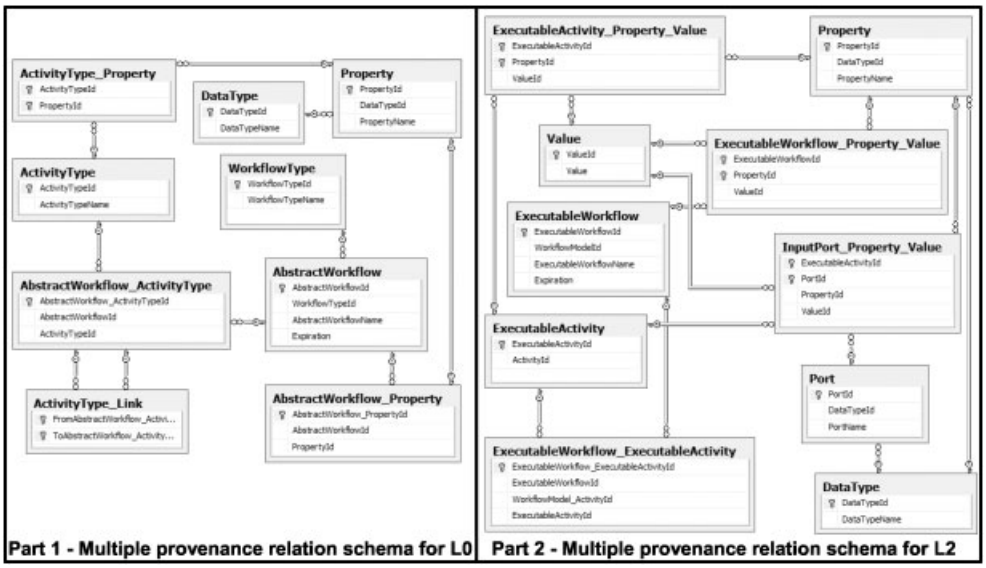


Figure 5. Multiple provenance relation schemas for L0 and L2.

executed and we can capture runtime information, such as branches taken, activities skipped, or inserted at runtime, etc.

Due to space limitations, we do not present schemas for other levels in our model, and refer readers to the REDUX project entry in the FPC [6] for a detailed database design. In addition to the workflow levels, we also capture annotation provenance in our model. We permit users to attach annotation to a workflow at all levels, and they can attach annotations to other objects, such as events, individual workflow activities, parameters, input data, etc. In our current implementation, annotations are attribute name–value pairs, consisting of the name of the annotation and its value. Annotation metadata are represented with a separate table for each annotation type. For example, all annotations with integer values are stored in one table and all attributes with string values are stored in a separate table. This approach allows indexing of annotations based on annotation type.

RELATED WORK

This section discusses issues, advantages, and possible combinations of other approaches presented in the FPC. A summarized figure of the approaches can be found in [5]. We highlight and extend the discussion on provenance modeling, workflow visualization, and query language and query visualization.

There is great diversity in the modeling of workflows and their provenance. But the most interesting thing to be discussed here is a common concern found in other participant approaches: information reuse. To facilitate sharing and re-usability, we model our experiments in four abstraction layers. Another interesting approach is the use of *templates* presented by [7]. This facilitates the

DOI: 10.1002/cpe

instantiation of a workflow facilitating the reuse. We can compare the *workflow template* approach with our *abstract workflow*/*workflow model* approach. One important difference of our system is that we represent all the layers in the same data model. Thus, the user can insert details from the *executable workflow* before finishing, constructing the *abstract workflow*. The MyGrid [8] approach also allows the user to work with different granularities. Still discussing the workflow model, the vistrail model [9] allows the storing of multiple versions of a workflow. This is a scalable solution that allows the reuse of data among the several versions of the same workflow. REDUX allows the construction of workflows as arbitrary graphs (with cycles). The majority of teams (e.g., [8,10,11]) represent workflows as DAGs (Directed Acyclic Graphs). This solution is fine to prevent deadlocks but restricts the construction of workflows (it does not allow the presence of loops).

To facilitate the use (and re-use) of workflow activities, one common solution is to store the full API of any activity in the provenance model [8,9]. Our system and others (e.g., [9]) had problems to identify the interface of the activities in the workflow of the provenance challenge. However, this system works well with web services. The workflow visualization strategies vary among the solutions. Some approaches do not make a distinction (graphically) between the activities and the data in the workflow [8]. Other approaches present graphically all details about the workflow execution [7]. One interesting approach is the visualization strategy of workflow evolution presented by [9] where each node corresponds to a different version of a given workflow. Our solution allows the navigation among the abstraction layers (from executable workflow to abstract workflow and vice versa) but does not provide an easy navigation mechanism among workflows.

Our system uses the SQL query facilities enacted with a graphical tool to facilitate queries about workflows (and their provenance). To facilitate these queries some provenance challenge participant teams developed their own query language [9] which can deal very well with workflow and provenance queries. Another interesting approach was the use of query mechanisms over structured data (as XML or RDF data)—such as, the use of SPARQL. One important issue about queries that was not yet addressed by our approach is about query answer visualization.

We developed a prototype to allow the visualization of sub-workflows that are answers of a given provenance query. However, we did not develop mechanisms to facilitate the visualization of complex data objects. These mechanisms can be found in the work of [8,9] where the user can graphically visualize the classification of the resulting objects. This approach, when combined with the use of trees to present the results of a query, seems to be a great solution.

DISCUSSION

In this paper, we argue the collection of provenance data should be the responsibility of the workflow system, and the resulting provenance data managed by a relational store. We present a snapshot of the REDUX project. In our system, provenance collection and management are transparent, so users do not have to take any special actions to have the provenance of their experimental result collected and maintained. By making the system responsible for the generation and collection of result provenance during execution, we free the scientists from having to track provenance manually. We envisage these provenance data to be a first class data resource in its own right, allowing users to both query experiment holdings to explain or establish trust in a result, and to drive the creation of future experiments.

DOI: 10.1002/cpe

We present a multilayer model, to incrementally capture the provenance that a result is bound to. This model allows users to navigate from abstract levels into lower detailed execution levels, depending on the amount of detail required to validate a result. Our representation also allows a scientist to specify exactly what they wish to share, or retain, from an experiment. When stored in a relational database, this provenance data can be indexed and queried as a first class data product using conventional tools and techniques. Moreover, we can apply techniques that make the storage of this data product economically feasible.

Our system handily dealt with the workflow in the Provenance Challenge though our single difficulty was to identify a detailed API of the workflow activities. Almost all approaches presented in the FPC adequately addressed the challenges of the event and were able to answer the provenance queries. The interchange of information among the participant groups allowed us to identify aspects that can be improved in our system and we look forward to the next generation of provenance systems that will be the result from this event.

REFERENCES

1. Bowers S, McPhillips T, Ludaescher B. A provenance model for collection-oriented scientific workflows. *Concurrency and Computation*: *Practice and Experience*. DOI: 10.1002/cpe.1226.
2. Medeiros CB, Perez-Alcazar J, Digiampietri L, Pastorello G, Santanche A, Torres R, Madeira E, Bacarin E. WOODSS and the web: Annotating and reusing scientific workflow. *ACM SIGMOD Record* 2005; **34**(3):18–23.
3. Pastorello GZ Jr, Medeiros CB, Resende SM, Rocha HA. Interoperability for GIS document management in environmental planning. *Journal of Data Semantics* 2005; **3**:100–124 (*Lecture Notes in Computer Science*, vol. 3534). Springer: Berlin.
4. Andrew P *et al*. *Presenting Windows Workflow Foundation*. SAMS Publishing: Indianapolis, 2006.
5. Moreau L, Ludäscher B, Barga RS, Digiampietri L, Goldbeck J, Simmhan YL, Plale B, Krenek A, Turi D, Goble C, Stevens R, Zhao J, Freire J, Frew J, Metzger D, Slaughter P, Cohen-Boulakia S, Davidson S, Cohen S, Bowers S, McPhillips T, Podhorszki N, Altintas I, Holland D, Seltzer M, Deelman E, Gil Y, Kim J, Mehta G, Ratnakar V, Myers J, Futrelle J, Miles S, Munroe S, Groth P, Jiang S, Schuchardt K, Gibson T, Stephan E, Chin G, Clifford B, Wilde M, Foster I. The first provenance challenge. *Concurrency and Computation*: *Practice and Experience*. DOI: 10.1002/cpe.1233.
6. The First Provenance Challenge. http://twiki.ipaw.info/bin/view/Challenge/ [13 November 2006].
7. Kim J, Deelman E, Gil Y, Mehta G, Ratnakar V. Provenance trails in the wings/pegasus system. *Concurrency and Computation*: *Practice and Experience*. DOI: 10.1002/cpe.1228.
8. Zhao J, Goble C, Stevens R, Turi D. Mining taverna's semantic web of provenance. *Concurrency and Computation*: *Practice and Experience*. DOI: 10.1002/cpe.1231.
9. Scheidegger C, Koop D, Santos E, Vo H, Callahan S, Freire J, Silva C. Tackling the provenance challenge one layer at a time. *Concurrency and Computation*: *Practice and Experience*. DOI: 10.1002/cpe.1237.
10. Krenek A, Sitera J, Matyska L, Dvorak F, Mulac M, Ruda M, Salvet Z. Glite job provenance—a job-centric view. *Concurrency and Computation*: *Practice and Experience*. DOI: 10.1002/cpe.1252.
11. Miles S, Groth P, Munroe S, Jiang S, Assandri T, Moreau L. Extracting causal graphs from an open provenance data model. *Concurrency and Computation*: *Practice and Experience*. DOI: 10.1002/cpe.1236.