Provenance for Computational Tasks: A Survey

The problem of systematically capturing and managing provenance for computational tasks has recently received significant attention because of its relevance to a wide range of domains and applications. The authors give an overview of important concepts related to provenance management, so that potential users can make informed decisions when selecting or designing a provenance solution.

The *Oxford English Dictionary* defines provenance as "the source or origin of an object; its history and pedigree; a record of the ultimate derivation and passage of an item through its various owners." In scientific experiments, provenance helps us interpret and understand results: by examining the sequence of steps that led to a result, we can gain insights into the chain of reasoning used in its production, verify that the experiment was performed according to acceptable procedures, identify the experiment's inputs, and, in some cases, reproduce the result. Laboratory notebooks have been the traditional mechanism for maintaining such information, but because the volume of data manipulated in computational experiments has increased along with the complexity of analysis, manually capturing provenance and writing detailed notes is no longer an option—in fact, it can have serious limitations. Scientists and engineers expend substantial effort and time managing data and recording provenance information just to answer basic questions, such as, Who created this data product and when? Who modified it and when? What process created the data product? Did the same raw data lead to two data products?

The problem of systematically capturing and managing provenance for computational tasks is relevant to a wide range of domains and applications. Fortunately, this problem has received significant attention recently. Our goal with this survey article is to inform potential provenance technology users about different approaches and their trade-offs, thereby helping them make informed decisions while selecting or developing a provenance solution. Two other surveys also touch on the issue of provenance for computational tasks: R. Bose and J. Frew[1] provide a comprehensive overview that covers early work in the area as well as standards used in specific domains, and Y.L. Simmhan and colleagues[2] describe a taxonomy they developed to compare five systems. Our survey, in contrast, discusses fundamental issues in provenance management but isn't intended for specialists. Specifically, we identify three major components of provenance management and discuss different approaches used in each of them. We also cover recent literature and the current state of the art. Although we can't provide a comprehensive coverage of all systems due to space limitations, we do review a representative

1521-9615/08/$25.00 © 2008 IEEE
Copublished by the IEEE CS and the AIP

Juliana Freire, David Koop, Emanuele Santos, and Cláudio T. Silva
University of Utah

set, including those systems in wide use that illustrate different solutions. Applications that use provenance appear in other articles in this special issue. The problem of managing fine-grained provenance recorded for items in a database is out of scope for this survey—a detailed overview appears elsewhere.[3]

Provenance Management: An Overview

Before discussing the specific trade-offs among provenance systems and models, let's examine the general aspects of provenance management. Specifically, let's explore the methods for modeling computational tasks and the types of provenance information we can capture from these tasks. To illustrate these themes, we use a scenario that's common in visualizing medical data—the creation of multiple visualizations of a volumetric computer tomography (CT) data set.

Modeling Computational Tasks

To allow reproducibility, we can represent computational tasks with a variety of mechanisms, including computer programs, scripts, and workflows, or construct them interactively by using specialized tools (such as ParaView [www.paraview.org] for scientific visualization and GenePattern [www.broad.mit.edu/cancer/software/genepattern] for biomedical research). Some complex computational tasks require weaving tools together, such as loosely coupled resources, specialized libraries, or Web services. To analyze a CT scan's results, for example, we might need to preprocess data with different parameters, visualize each result, and then compare them. To ensure the reproducibility of the entire task, it's beneficial to have a description that captures these steps and the parameter values used. One approach is to order computational processes and organize them into scripts; the session log information that some software tools expose can also help document and reproduce results. However, these approaches have shortcomings—specifically, the user is responsible for manually checking-in incremental script changes or saving session log files. Moreover, the saved information often isn't in an easily queried format.

Recently, workflows and workflow-based systems have emerged as an alternative to these ad hoc types of approaches. Workflow systems provide well-defined languages for specifying complex tasks from simpler ones; they capture complex processes at various levels of detail and systematically record the provenance information necessary for automation, reproducibility, and result sharing. In fact, workflows are rapidly replacing primitive shell scripts, as evidenced by the release of Apple's Mac OS X Automator, Microsoft's Workflow Foundation, and Yahoo!'s Pipes.

Workflows have additional advantages over programs and scripts, such as providing a simple programming model that allows a sequence of tasks to be composed by connecting one task's outputs to the inputs of another. Workflow systems can also provide intuitive visual programming interfaces that are easier to use for people who don't have substantial programming expertise. In addition, workflows have an explicit structure—we can view them as graphs, with nodes representing processes (or modules) and edges capturing the data flow between those processes (see Figure 1). Having this explicit structure enables the information to be explored and queried. A program (or script) is to a workflow what an unstructured document is to a (structured) database.

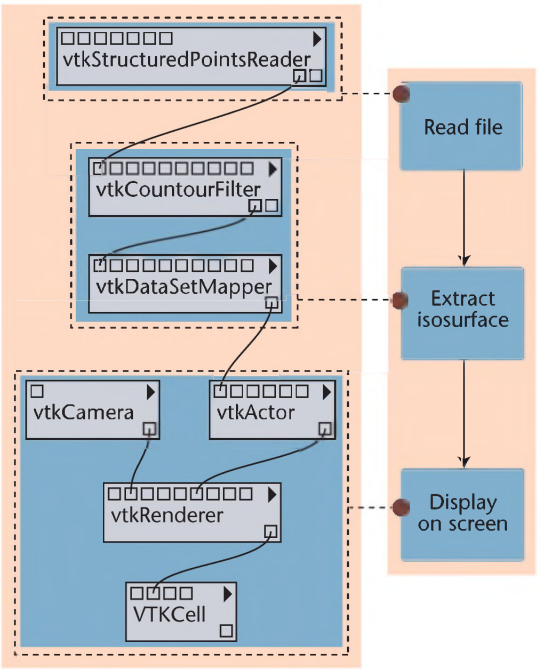
Another important concept related to computational tasks is *abstraction*, which lets us split complex tasks and represent them at different levels of granularity. As Figure 1 illustrates, we can use abstraction to create a simplified view of a workflow that hides some of its details. A researcher studying a CT scan data set, for example, might not know—or care about—the details of the Visualization Toolkit (VTK; www.kitware.com) library that created the visualizations. So, instead of displaying the four distinct modules that render the final image (Figure 1b), a user can abstract them into a single module with a more descriptive name such as "display on screen" (Figure 1c).

Different Forms of Provenance

We provided a high-level definition of provenance at the beginning of this article. When it comes to computational tasks, there are two forms of provenance: prospective and retrospective.[4] *Prospective provenance* captures a computational task's specification (whether it's a script or a workflow) and corresponds to the steps (or recipe) that must be followed to generate a data product or class of data products. *Retrospective provenance* captures the steps executed as well as information about the environment used to derive a specific data product—in other words, it's a detailed log of a computational task's execution. Moreover, retrospective provenance doesn't depend on the presence of prospective provenance—for example, we can capture information such as which process ran, who ran it, and how long it took without having prior knowledge of the sequence of computational steps involved.

```
1  import vtk
2
3  data = vtk.vtkStructuredPointsReader()
4  data.setFileName("../../../examples/data/head.120.vtk")
5
6  contour = vtk.vtkContourFilter()
7  contour.SetInput(0, data.GetOutput())
8  contour.SetValue(0, 67)
9
10 mapper = vtk.vtkPolyDataMapper()
11 mapper.SetInput(contour.GetOutput())
12 mapper.ScalarVisibilityOff()
13
14 actor = vtk.vtkActor()
15 actor.SetMapper(mapper)
16
17 cam = vtk.vtkCamera()
18 cam.SetViewUp(0,0,-1)
19 cam.SetPosition(745,-453,369)
20 cam.SetFocalPoint(135,135,150)
21 cam.ComputeViewPlaneNormal()
22
23 ren = vtk.vtkRenderer()
24 ren.AddActor(actor)
25 ren.SetActiveCamera(cam)
26 ren.ResetCamera()
27
28 renwin = vtk.vtkRenderWindow()
29 renwin.AddRenderer(ren)
30
31 style = vtk.vtkInteractorStyleTrackballCamera()
32 iren = vtk.vtkRenderWindowIneractor()
33 iren.SetRenderWindow(renwin)
34 iren.SetInteractorStyle(style)
35 iren.Initialize()
36 iren.Start()
```

(a)



(b) (c)

Figure 1. Different abstractions for a data flow. (a) A Python script containing a series of Visualization Toolkit (VTK) calls; (b) a workflow that produces the same result as the script; and (c) a simplified view of the workflow that hides some of its details.

For our running example, Figure 2 illustrates prospective provenance captured as the definition of a workflow that produces two kinds of data products: a histogram and an isosurface visualization. The retrospective provenance (the left side of Figure 2a) contains information for each module about input and output data, the executing user, and the execution start and end times.

An important component of provenance is information about *causality*—the process description (or sequence of steps) that, together with input data and parameters, led to the data product's creation. We can infer causality from both prospective and retrospective provenance; Figure 2b illustrates the relationships in our running example. We can also represent causality as a graph in which nodes correspond to processes and data products and edges correspond to either data or data-process dependencies. Data-process dependencies (for example, the fact that the first workflow produced `head-hist.png`) are useful for documenting the data generation process, but we can also use them to reproduce or validate a process. Data dependencies are likewise useful—for example, in the event that we learn the CT scanner used to generate `head.120.vtk` is defective, we can examine the data dependencies and discount the results that rely on it.

Another key component of provenance is user-defined information—documentation that isn't captured automatically but that records important decisions and notes. This data often comes in the form of annotations—as Figure 2a illustrates, users can add them at different levels of granularity and associate them with different components of both prospective and retrospective provenance (such as modules, data products, or execution log records).

Three Key Components

A provenance management solution consists of three main components: a capture mechanism, a representational model, and an infrastructure for storage, access, and queries. In this section, we examine different classes of solutions for each of these components and discuss the trade-offs among them. To illustrate the approaches, we use examples that highlight some of the capabilities of existing provenance-enabled tools.

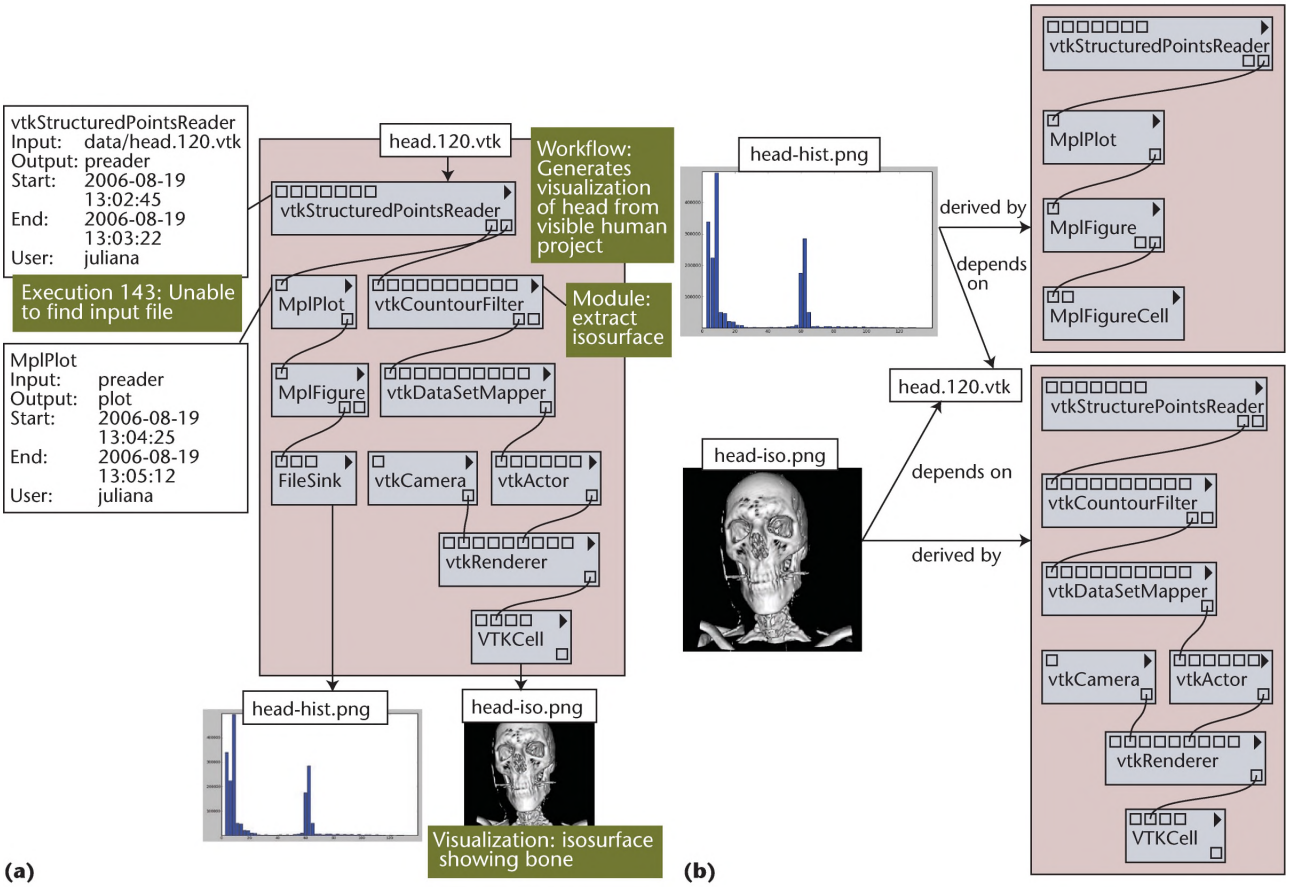


Figure 2. Prospective and retrospective provenance. (a) The workflow on the left generates two data products: a histogram of a structured data set's scalar values and an isosurface visualization of that data set. We also see some of the retrospective provenance collected during a run along with user-defined provenance in the form of annotations, shown in green boxes. (b) We derived the data products on the left from the workflow excerpts on the right, both of which depend on the input data set `head.120.vtk`.

Capture Mechanisms

A provenance capture mechanism needs access to a computational task's relevant details, such as its steps, execution information, and user-specified annotations. This type of mechanism falls into three main classes: workflow-, process-, and operating system- (OS-) based. Workflow-based mechanisms are either attached to or integrated in a workflow system; process-based mechanisms require each service or process involved in a computational task to document itself; and OS-based mechanisms need no modifications to existing scripts or programs; instead, they rely on the availability of specific functionality at the OS level.

A major advantage of workflow-based mechanisms is that they're usually tightly coupled with workflow systems, which enables a straightforward capture process through system APIs. Some early workflow systems (such as Taverna [http://taverna.sourceforge.net] and Kepler [http://kepler-project.org]) have been extended to capture provenance, but newer systems (such as VisTrails [www.vistrails.org]) support it from their initial design. Each service or tool in a process-based mechanism must be instrumented to capture provenance, with any information derived from autonomous processes pieced together to provide documentation for complex tasks.[5] OS-based mechanisms aren't coupled with workflows or processes at all, and thus require postprocessing to extract relationships between system calls and tasks.[6,7]

One advantage of OS-based mechanisms is that they don't require modifications to existing processes and are agnostic about how tasks are modeled—they rely on the OS environment's ability to transparently capture data and data-process dependencies at the kernel (via the filesystem interface)[7] or user levels (via the system call tracer).[6]

In contrast, both workflow- and process-based approaches require processes to be wrapped—in the former, so that the workflow engine can invoke them, and in the latter, so that instrumentation can capture and publish provenance information.

Because workflow systems have access to workflow definitions and control their execution, they can capture both prospective and retrospective provenance. OS- and process-based mechanisms only capture retrospective provenance: they must reconstruct causal relationships through provenance queries. The ES3 system (http://eil.bren.ucsb.edu), for example, monitors the interactions between arbitrary applications and their environments (via arguments, file I/O, system, and calls), and then uses this information to assemble a provenance graph to describe what actually happened during execution.[6]

In fact, by capturing provenance at the OS level, we can record detailed information about all system calls and files touched during a task's execution. This forms a superset of the information captured in workflow- and process-based systems, whose granularity is determined by the wrapping provided for individual processes. Consider, for example, a command-line tool integrated in a workflow system that creates and depends on temporary files not explicitly defined in its wrapper. The causal dependencies the workflow system captures won't include the temporary files, but we can capture these dependencies at the OS level. However, because even simple tasks can lead to a large number of low-level calls, the amount of provenance that OS-based approaches record can be prohibitive, making it hard to query and reason about the information.[7]

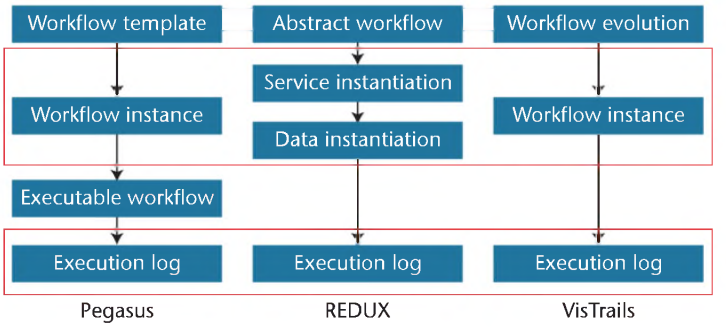


Figure 3. Layered provenance models. For REDUX, the first layer corresponds to an abstract description, the second layer describes the binding of specific services and data to the abstract description, the third layer captures runtime inputs and parameters, and the final layer captures operational data. Other models use layers in different ways. The top-layer in VisTrails captures provenance of workflow evolution, and Pegasus uses an additional layer to represent the workflow execution plan over grid resources.

Provenance Models

Researchers have proposed several provenance models in the literature.[9,10,12] All these models support some form of retrospective provenance, and most of those that workflow systems use provide the means to capture prospective provenance. Many of the models also support annotations.

Although these models differ in several ways, including their use of structures and storage strategies, they all share an essential type of information: process and data dependencies. In fact, a recent exercise to explore interoperability issues among provenance models showed that it's possible to integrate information that conforms to different provenance models (http://twiki.ipaw.info/bin/view/Challenge/SecondProvenanceChallenge).

Despite a base commonality, provenance models tend to vary according to domain and user needs. Even though most models strive to store general concepts, specific use cases often influence model design—for example, Taverna was developed to support the creation and management of workflows in the bioinformatics domain, and therefore provides an infrastructure that includes support for ontologies available in this domain. VisTrails was designed to support exploratory tasks in which workflows are iteratively refined, and thus uses a model that treats workflow specifications as first-class data products and captures the provenance of workflow evolution.

Because the provenance information a model must represent varies both by type and specificity, it's advantageous to structure a model as a set of layers to enable a normalized, configurable representation. The ability to represent provenance at different levels of abstraction also leads to simpler queries and more intuitive results. Consider the REDUX system,[16] which uses the layered model depicted in Figure 3. The first layer corresponds to an abstract description of a workflow, in which each module corresponds to a class of activities. This abstract description is bound to specific services and data sets defined in the second layer—for example, in the workflow shown in Figure 1, the abstract activity `extract isosurface` is bound to a call to the `vtkContourFilter`—a specific implementation of isosurface extraction provided by VTK. The third layer captures information about input data and parameters supplied at runtime, and the fourth layer captures operational details, such as the workflow execution's start and end time.

Structuring provenance information into multiple layers leads to a *normalized representation*

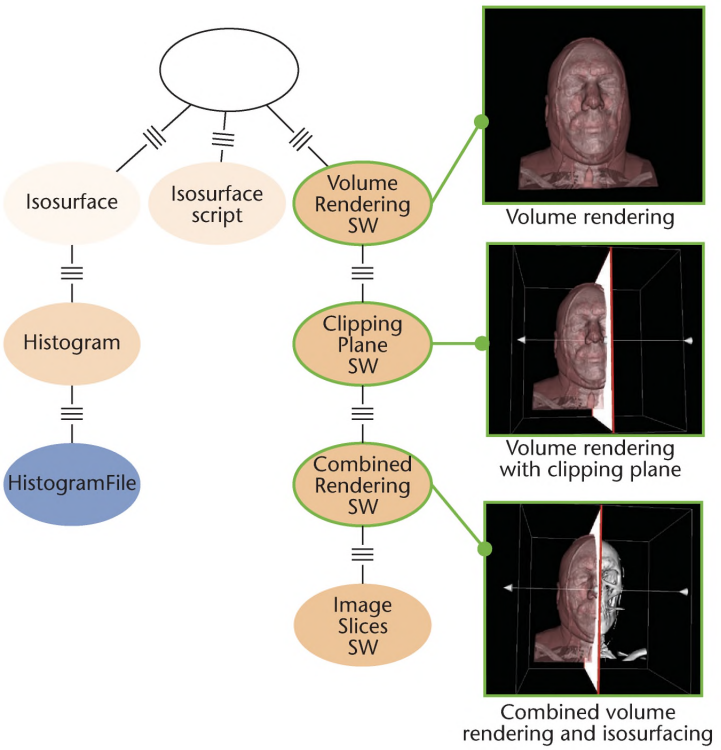


Figure 4. Workflow evolution exploration interface. The tree's nodes correspond to workflows, and an edge between two nodes corresponds to changes performed on the parent workflow to obtain its child. Data products derived by the different workflows are shown on the right.

that avoids the storage of redundant information. Some models, for example, store a workflow's specification once for all of its executions;[12,16] in models with a single layer, such as Kepler's, the specification of a workflow instance must be saved to the provenance model every time the workflow is executed along with any runtime information.[17] This not only incurs high storage overheads but also negatively impacts query performance.

Although many models provide storage for workflow specification and execution information, the layers differ across systems. For those that schedule scientific workflow execution on the grid (such as Pegasus[11]), it's important to save scheduling information as well as execution provenance. To support higher-level semantic queries, it might be useful to add additional layers of application-specific metadata and ontologies, such as in Taverna.[18] VisTrails includes an additional layer that records information about workflow evolution.[12] In addition to causality information that relates tasks to data products, it keeps track of how these tasks evolve over time—in effect, a trail of the workflow refinement process. The VisTrails change-based provenance model records information about modifications to a task, akin to a database transaction log. For workflows, such changes include a module's addition or deletion, the addition of a connection between modules, and parameter value modifications. One major benefit of this approach is that it's concise and uses substantially less space than the alternative, which stores multiple versions of a task specification. It also leads to an interface (see Figure 4) that presents a workflow evolution's history as a tree, letting scientists return to previous versions intuitively.

Storing, Accessing, and Querying Provenance

Several approaches exist for capturing and modeling provenance, but only recently has the problem of storing, accessing, and querying started to receive attention. Researchers have used a wide variety of data models and storage systems, ranging from specialized Semantic Web languages and XML dialects stored as files to tuples stored in relational database tables. One of the advantages of filesystem storage is that users don't need additional infrastructure to store provenance information. On the other hand, a relational database provides centralized, efficient storage that a group of users can share.

Infrastructure for effectively and efficiently querying provenance data is a necessary component of a provenance management system, especially when large volumes of information are captured. When an OS-based approach such as Provenance-Aware Storage Systems (PASS) captures very fine-grained provenance,[7] for example, the volume of information can be overwhelming, making it difficult to explore. One of the queries in the First Provenance Challenge asked for the process used to generate a specific data product, and PASS returned more than 5 Mbytes of data (http://twiki.ipaw.info/bin/view/Challenge/FirstProvenanceChallenge). In contrast, REDUX returned a single tuple consisting of a few bytes for the same query.

Provenance overload can also be a problem for some workflow-based systems. Because a workflow's execution can take multiple steps and run several times, the amount of information stored for a single workflow can be very large. Biton and colleagues proposed a solution that uses abstractions through the creation of *user views*.[19] The user indicates which modules in the workflow specification are relevant, and the system presents the provenance information according to these preferences. Of course, this approach works best in workflow

REDUX

```
SELECT Execution.ExecutableWorkflowId, Execution.ExecutionId, Event.EventId, ExecutableActivity.ExecutableActivityId
from Execution, Execution_Event, Event, ExecutableWorkflow_ExecutableActivity, ExecutableActivity,
    ExecutableActivity_Property_Value, Value, EventType as ET
where Execution.ExecutionId=Execution_Event.ExecutionId
and Execution_Event.EventId=Event.EventId
and ExecutableActivity.ExecutableActivityId=ExecutableActivity_Property_Value.ExecutableActivityId
and ExecutableActivity_Property_Value.ValueId=Value.ValueId and Value.Value=Cast('-m 12' as binary)
and ((CONVERT(DECIMAL, Event.Timestamp)+0)%7)=0 and Execution_Event.ExecutableWorkflow_ExecutableActivityId=
   ExecutableWorkflow_ExecutableActivity.ExecutableWorkflow_ExecutableActivityId
and ExecutableWorkflow_ExecutableActivity.ExecutableWorkflowId=Execution.ExecutableWorkflowId
and ExecutableWorkflow_ExecutableActivity.ExecutableActivityId=ExecutableActivity.ExecutableActivityId
and Event.EventTypeId=ET.EventTypeId and ET.EventTypeName='Activity Start';
```

VisTrails

```
wf{*}: x where x.module='AlignWarp' and x.parameter('model')='12'
       and (log{x}: y where y.dayOfWeek='Monday')
```

MyGrid

```
SELECT ?p
where (?p <http://www.mygrid.org.uk/provenance#startTime> ?time) and (?time > date)
using ns for <http://www.mygrid.org.uk/provenance#> xsd for <http://www.w3.org/2001/XMLSchema#>

SELECT ?p
where <urn:lsid:www.mygrid.org.uk:experimentinstance:HXQOVQA2ZI0>
(?p <http://www.mygrid.org.uk/provenance#runsProcess> ?processname .
?p <http://www.mygrid.org.uk/provenance#processInput> ?inputParameter .
?inputParameter <ont:model> <ontology:twelfthOrder>)
using ns for <http://www.mygrid.org.uk/provenance#> ont for <http://www.mygrid.org.uk/ontology#>
```

Figure 5. Provenance query implemented by three different systems. REDUX uses SQL, VisTrails uses a language specialized for querying workflows and their provenance, and myGrid uses SPARQL.

systems that support abstractions (such as VisTrails, Taverna, and Kepler), but the ability to create views of provenance data would benefit OS- and process-based provenance models as well.

The ability to query a computational task's provenance also enables knowledge reuse. By querying a set of tasks and their provenance, users not only identify suitable tasks and reuse them but also compare and understand differences between different tasks. Provenance information is often associated with data products (such as images or graphs), so this data helps users pose structured queries over unstructured data as well.

A common feature across many approaches to querying provenance is that their solutions are closely tied to the storage models used. Hence, they require users to write queries in languages such as SQL,[16] Prolog,[20] and SPARQL.[10,11] Although such general languages are useful to those already familiar with their syntax, they weren't designed specifically for provenance, which means simple queries can be awkward and complex to write. Figure 5 compares three representations of a single query in the First Provenance Challenge that asked for tasks using a specific module (`Align Warp`) with given parameters executed on a Monday. The VisTrails approach uses a language specifically designed to query workflows and their provenance, whereas REDUX and myGrid use native languages for their storage choices. Because the VisTrails language abstracts details about physical storage, it leads to much more concise queries.

However, even queries that use a language designed for provenance are likely to be too complicated for many users because provenance contains structural information represented as a graph. Thus, text-based query interfaces effectively require a subgraph query to be encoded as text. The VisTrails query-by-example (QBE) interface (see Figure 6) addresses this problem by letting users quickly construct expressive queries using the same familiar interface they use to build workflow.[21] The query's results are also displayed visually.

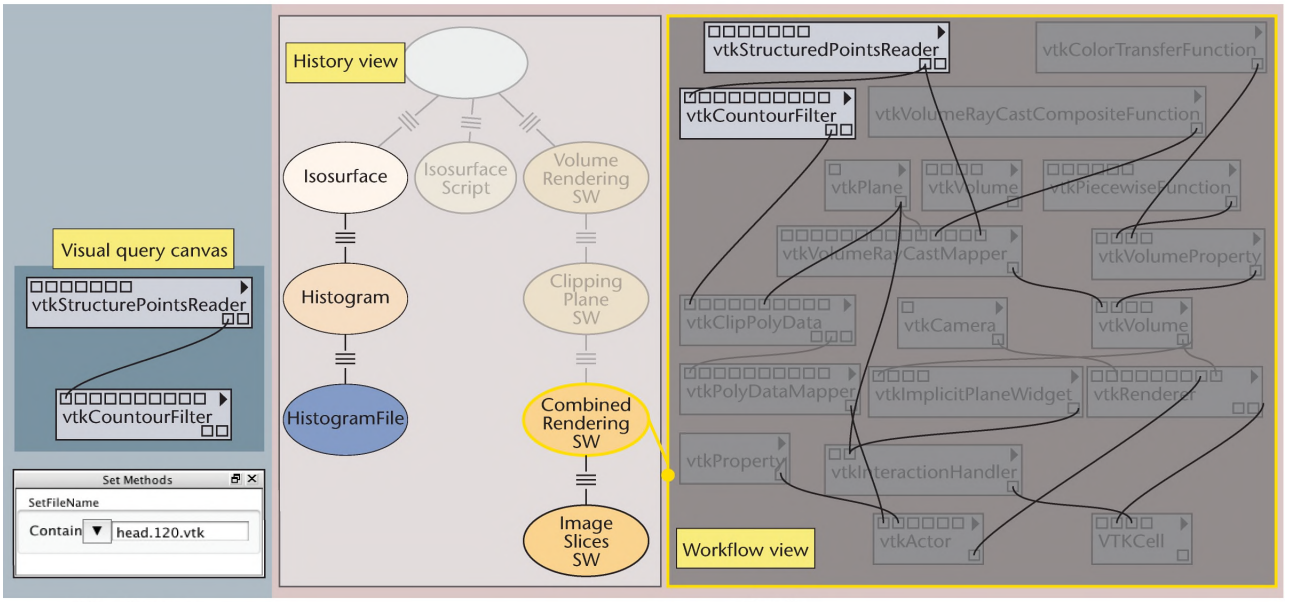


Figure 6. Querying provenance by example. We can specify a query visually in the same interface used to construct workflows. Besides the structure defined by the set of modules and connections, we can also specify conditions for parameter values.

Some provenance models use Semantic Web technology both to represent and query provenance information.[10,11,15,22] Semantic Web languages such as RDF and OWL provide a natural way to model provenance graphs and the ability to represent complex knowledge, such as annotations and metadata. This technology has the potential to simplify interoperability across different provenance models, but it's an open issue whether this technology scales to handle large provenance stores.

Provenance-Enabled Systems

Now let's review a selection of different provenance-enabled systems. We don't intend to provide a comprehensive guide to all existing systems—rather, we aim to describe a representative subset. Table 1 summarizes the various systems' features.

Workflow-Based Systems

Taverna is a workflow system used in the myGrid project (www.mygrid.org.uk), whose goal is to leverage Semantic Web technologies and ontologies available for bioinformatics to simplify data analysis processes. In Taverna, the workflow engine is responsible for capturing provenance. It stores any prospective provenance information as Scufl specifications (an XML dialect) and retrospective provenance as RDF triples in a MySQL database.

Karma (www.extreme.indiana.edu/karma) was developed to support dynamic workflows in weather forecasting simulations, where the execution path can change rapidly due to external events.[14] Karma collects retrospective provenance in the form of a workflow trace; it also explicitly models data products' derivation history. Although Karma is a workflow-based system, individual services that compose a workflow publish their own provenance to minimize performance overheads at the workflow-engine level (just as in process-based capture). Karma records the published provenance messages in a central database server: this information is exchanged between services and the server as XML, but it's translated into relational tuples before final storage. Karma also stores prospective provenance using the Business Process Execution Language.

The *Kepler* workflow system stores prospective provenance in the Modeling Markup Language (MoML) format,[1] which is an XML dialect for representing workflows; it captures retrospective provenance by using a variation of MoML that omits irrelevant information, such as the coordinates for workflow modules drawn on the screen. Currently, it stores provenance in files; query support is under development.

Pegasus (http://pegasus.isi.edu) is a workflow-mapping engine that, while taking into account resource efficiency and dynamic availability, au-

tomatically maps high-level workflow specifications into executable plans that run on distributed infrastructures such as the TeraGrid.[11] Although Pegasus models prospective provenance using OWL, it captures retrospective provenance via the Virtual Data System (VDS; a precursor of Swift) and then stores it in a relational database. Queries that span prospective and retrospective provenance must combine two different query languages: SPARQL and SQL.

REDUX extends the Windows Workflow Foundation engine to transparently capture the workflow execution trace. As discussed earlier, it uses a layered provenance model to normalize data and avoid redundancy. REDUX stores provenance data (both prospective and retrospective) in a relational database's set of tables that can be queried with SQL. The system can also return an executable workflow as the result of a provenance query (for example, a query that requests all the steps used to derive a particular data product).

Swift (www.ci.uchicago.edu/swift) builds on and includes technology previously distributed as the GriPhyN VDS.[23] The system combines a scripting language (SwiftScript) with a powerful runtime system for the concise specification and reliable execution of large, loosely coupled computations. Swift specifies these computations as scripts, which the runtime system translates into an executable workflow. A launcher program invokes the workflow's tasks, monitors the execution process, and records provenance information, including the executable name, arguments, start time, duration, machine information, and exit status. Similar to VDS, Swift captures the relationships among data, programs, and computations and uses this information for data and program discovery as well as for workflow scheduling and optimization.

VisTrails is a workflow and provenance management system designed to support exploratory computational tasks. An important goal of the VisTrails project is to build intuitive interfaces for users to query and reuse provenance information. Besides its QBE interface (which is built on top of its specialized provenance query language), VisTrails provides a visual interface to compare workflows side by side[12] and a mechanism for refining workflows by analogy—users can modify workflows by example without having to directly edit their definitions.[21] VisTrails internally represents prospective provenance as Python objects that can be serialized into XML and relations; it stores retrospective provenance in a relational database.

OS-Based Systems

PASS (www.eecs.harvard.edu/syrah/pass) operates at the level of a shared storage system: it automatically records information about which programs are executed, their inputs, and any new files created as output. The capture mechanism consists of a set of Linux kernel modules that transparently record provenance—it doesn't require any changes to computational tasks. PASS also constructs a provenance graph stored as a set of tables in Berkeley DB. Users can pose provenance queries using *nq*, a proprietary tool that supports recursive searches over the provenance graph. As discussed earlier, the fine granularity of PASS's capture mechanism often leads to very large volumes of provenance information; another limitation of this approach is that it's restricted to local filesystems. It can't, for example, track files in a grid environment.

ES3's goal is to extract provenance information from arbitrary applications by monitoring their interactions with the execution environment.[6] These interactions are logged to the ES3 database, which stores the information as provenance graphs, represented in XML. ES3 currently supports a Linux plug-in, which uses system call tracing to capture provenance. As in PASS, ES3 requires no changes to the underlying processes, but provenance capture is restricted to applications that run on ES3-supported environments.

Process-Based Systems

The Provenance-Aware Service Oriented Architecture (PASOA) project (www.pasoa.org) developed a provenance architecture that relies on individual services to record their own provenance.[5] The system doesn't model the notion of a workflow—rather, it captures assertions produced by services that reflect the relationships between represented services and data. The system must infer the complete provenance of a task or data product by combining these assertions and recursively following the relationships they represent. The PASOA architecture distinguishes the notion of *process documentation*—that is, the provenance recorded specifically about a process—from the notion of a *data item's provenance*, which is derived from the process documentation. The PASOA project developed an open source software package called PreServ that lets developers integrate process documentation recording into their applications. PreServ also supports multiple back end storage systems, including files and relational databases; users can pose provenance queries by using its Java-based query API or XQuery.

Table 1. An overview of provenance-enabled systems.

System	Capture mechanism	Prospective provenance	Retrospective provenance	Workflow evolution
REDUX	Workflow-based	Relational	Relational	No
Swift	Workflow-based	SwiftScript	Relational	No
VisTrails	Workflow-based	XML and relational	Relational	Yes
Karma	Workflow- and process-based	Business Process Execution Language	XML	No
Kepler	Workflow-based	MoML	MoML variation	Under development
Taverna	Workflow-based	Scufl	RDF	Under development
Pegasus	Workflow-based	OWL	Relational	No
PASS	OS-based	N/A	Relational	No
ES3	OS-based	N/A	XML	No
PASOA/PreServ	Process-based	N/A	XML	No

Provenance management is a new area, but it's advancing rapidly. Researchers are actively pursuing several directions in this area, including the ability to integrate provenance derived from different systems and enhanced analytical and visualization mechanisms for exploring provenance information. Provenance research is also enabling several new applications, such as science collaboratories, which have the potential to change the way people do science—sharing provenance information at a large scale exposes researchers to techniques and tools to which they wouldn't otherwise have access. By exploring provenance information in a collaboratory, scientists can learn by example, expedite their scientific work, and potentially reduce their time to insight. The "wisdom of the crowds," in the context of scientific exploration, can avoid duplication and encourage continuous, documented, and reproducible scientific progress.[24]

Acknowledgments

This work was partially supported by the US National Science Foundation, the US Department of Energy, and IBM faculty awards.

References

1. R. Bose and J. Frew, "Lineage Retrieval for Scientific Data Processing: A Survey," *ACM Computing Surveys*, vol. 37, no. 1, 2005, pp. 1–28.
2. Y.L. Simmhan, B. Plale, and D. Gannon, "A Survey of Data Provenance in E-Science," *SIGMOD Record*, vol. 34, no. 3, 2005, pp. 31–36.
3. W.C. Tan, "Provenance in Databases: Past, Current, and Future," *IEEE Data Eng. Bulletin*, vol. 30, no. 4, 2007, pp. 3–12.
4. B. Clifford et al., "Tracking Provenance in a Virtual Data Grid," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008, pp. 565–575.
5. P. Groth, *The Origin of Data: Enabling the Determination of Provenance in Multi-Institutional Scientific Systems through the Documentation of Processes*, PhD thesis, Univ. of Southampton, 2007.
6. J. Frew, D. Metzger, and P. Slaughter, "Automatic Capture and Reconstruction of Computational Provenance," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008, pp. 485–496.
7. K.-K. Muniswamy-Reddy, D.A. Holland, and U.B.M.I. Seltzer, "Provenance-Aware Storage Systems," *Proc. USENIX Conf.*, Usenix, 2006, pp. 43–56.
8. I. Altintas, O. Barney, and E. Jaeger-Frank, "Provenance Collection Support in the Kepler Scientific Workflow System," *Proc. Int'l Provenance and Annotation Workshop* (IPAW), LNCS 4145, Springer, 2006, pp. 118–132.
9. S. Cohen, S.C. Boulakia, and S.B. Davidson, "Towards a Model of Provenance and User Views in Scientific Workflows," *Data Integration in the Life Sciences*, LNCS 4075, Springer, 2006, pp. 264–279.
10. J. Golbeck and J. Hendler, "A Semantic Web Approach to the Provenance Challenge," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008, pp. 431–439.
11. J. Kim et al., "Provenance Trails in the Wings/Pegasus System," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008, pp. 587–597.
12. J. Freire et al., "Managing Rapidly-Evolving Scientific Workflows," *Proc. Int'l Provenance and Annotation Workshop* (IPAW), LNCS 4145, Springer, 2006, pp. 10–18.
13. S. Miles et al., "Extracting Causal Graphs from an Open Provenance Data Model," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008, pp. 577–586.
14. Y.L. Simmhan et al., "Karma2: Provenance Management for Data Driven Workflows," to be published in *Int'l J. Web Services Research*, vol. 5, no. 1, 2008.
15. J. Zhao et al., "Mining Taverna's Semantic Web of Provenance," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008, pp. 463–472.
16. R.S. Barga and L.A. Digiampietri, "Automatic Capture and

Storage	Query support	Available as open source?
Relational database management system (RDBMS)	SQL	No
RDBMS	SQL	Yes
RDBMS and files	Visual query by example, specialized language	Yes
RDBMS	Proprietary API	Yes
Files; RDBMS planned	Under development	Yes
RDBMS	SPARQL	Yes
RDBMS	SPARQL for metadata and workflow; SQL for execution log	Yes
Berkeley DB	nq (proprietary query tool)	No
XML database	XQuery	No
Filesystem, Berkeley DB	XQuery, Java query API	Yes

Efficient Storage of e-Science Experiment Provenance," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008, pp. 419–429.

17. B. Ludäscher et al., "From Computation Models to Models of Provenance: The RWS Approach," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008, pp. 507–518.
18. T. Oinn et al., "Taverna: Lessons in Creating a Workflow Environment for the Life Sciences," *Concurrency and Computation: Practice & Experience*, vol. 18, no. 10, 2006, pp. 1067–1100.
19. O. Biton et al., "Querying and Managing Provenance through User Views in Scientific Workflows," to be published in *Proc. IEEE Int'l Conf. Data Eng.*, 2008.
20. S. Bowers, T. McPhillips, and B. Ludaescher, "Provenance in Collection-Oriented Scientific Workflows," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008, pp. 519–529.
21. C.E. Scheidegger et al., "Querying and Creating Visualizations by Analogy," *IEEE Trans. Visualization and Computer Graphics*, vol. 13, no. 6, 2007, pp. 1560–1567.
22. J. Futrelle and J. Myers, "Tracking Provenance Semantics in Heterogeneous Execution Systems," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008, pp. 555–564.
23. Y. Zhao et al., "Swift: Fast, Reliable, Loosely Coupled Parallel Computation," *IEEE Int'l Workshop on Sci. Workflows* (SWF), IEEE CS Press, 2007, pp. 199–206.
24. J. Freire and C. Silva, "Towards Enabling Social Analysis of Scientific Data," *CHI Social Data Analysis Workshop*, 2008, (to appear).

Juliana Freire *is an assistant professor at the University of Utah. Her research interests include scientific data management, Web information systems, and information integration. Freire has a PhD in computer science from SUNY at Stony Brook. She is a member of the ACM and the IEEE. Contact her at Juliana@cs.utah.edu.*

David Koop *is a research assistant and graduate student at the University of Utah. His research interests include scientific data management, visualization, and visualization systems. He has an MS in computer science from the University of Wisconsin-Madison. Contact him at dakoop@cs.utah.edu.*

Emanuele Santos *is a research assistant and graduate student at the University of Utah. Her research interests include scientific data management, visualization, and comparative visualization. Santos has an MS in computer science from the Federal University of Ceara in Brazil. Contact her at esantos@cs.utah.edu.*

Cláudio T. Silva *is an associate professor at the University of Utah. His research interests include visualization, geometry processing, graphics, and high-performance computing. Silva has a PhD in computer science from SUNY at Stony Brook. He is a member of the IEEE, the ACM, Eurographics, and Sociedade Brasileira de Matematica. Contact him at csilva@cs.utah.edu.*