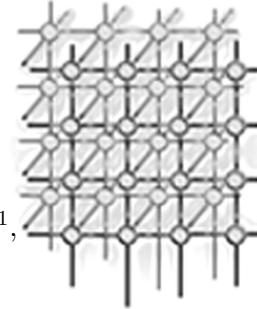

Tackling the Provenance Challenge One Layer at a Time



Carlos Scheidegger¹, David Koop², Emanuele Santos¹, Huy Vo¹,
Steven Callahan¹, Juliana Freire², Cláudio Silva¹

¹ *Scientific Computing and Imaging Institute, University of Utah*

² *School of Computing, University of Utah*

SUMMARY

VisTrails is a new workflow and provenance management system that provides support for scientific data exploration and visualization. Whereas workflows have been traditionally used to automate repetitive tasks, for applications that are exploratory in nature, change is the norm. VisTrails uses a new change-based provenance mechanism which was designed to handle rapidly-evolving workflows. It uniformly and automatically captures provenance information for data products and for the evolution of the workflows used to generate these products. In this paper, we describe how the VisTrails provenance data is organized in layers and present a first approach for querying this data that we developed to tackle the Provenance Challenge queries.

KEY WORDS: visualization, provenance, workflow evolution

1. Introduction

Workflows are emerging as a paradigm for representing and managing complex computations. Workflows capture elaborate processes in a structured way and provide the provenance information necessary for result reproducibility, publication, and sharing among collaborators. Because of the formalism they provide and the automation they support, workflows have the potential to accelerate and transform the information analysis process [6]. Workflows are rapidly replacing primitive shell scripts as evidenced by the release of Apple's Mac OS X Automator [1] and Microsoft's Workflow Foundation [12]. Existing workflow systems, however, fail to provide the necessary infrastructure for exploratory tasks that are common in the scientific process. Although these systems are effective for automating repetitive tasks, they are not suitable for applications that are exploratory in nature, where several different, albeit related workflows need to be created.



To address this limitation, we introduced the notion of *workflow provenance* and designed a novel change-based provenance model that uniformly captures provenance information for the evolution of the workflows and the data products they derive [4, 8]. This model not only ensures that results can be reproduced, but it also simplifies data exploration by allowing scientists to easily navigate through the space of workflows and parameter settings for a given computational task. The information about how different workflow versions are related can be leveraged to streamline the scientific discovery process. For example, to aid users in the creation of new workflows that solve similar or related problems.

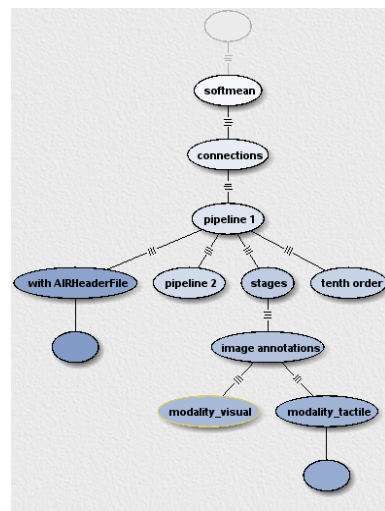
In this paper, we present our first attempt at designing a formalism and interface for querying the change-based model. In Section 2, we detail how provenance is modeled in our framework and describe how the framework is implemented in the VisTrails system. We discuss a new language designed for querying workflow provenance in Section 3 and show how it leverages our layered provenance model to answer *all* the queries defined for the First Provenance Challenge [13]. Our implementation of the challenge queries and their results are presented in Section 4. We conclude in Section 5 with a discussion of similarities and differences between our approach and those of the other participants.

2. The VisTrails Change-Based Provenance Mechanism

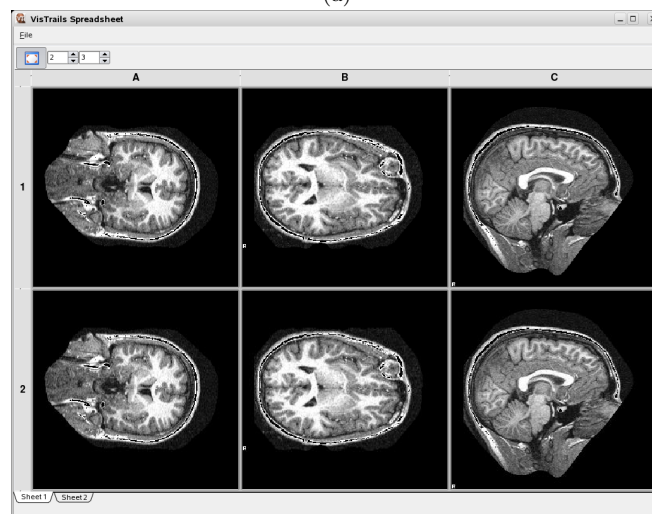
With VisTrails, we introduced a new model for capturing and representing provenance—VisTrails captures changes users make to a workflow. This model has important benefits. The changes are automatically captured as users iteratively refine their workflows, providing detailed information about the trial-and-error process followed in an exploratory task. To make this paper self-contained, before describing the change-based provenance model, we give a brief overview of the components of VisTrails that are directly related to the provenance model. For more details on the system architecture and implementation the reader is referred to [3, 5].

The VisTrails System. Although the initial motivation for developing VisTrails was to provide support for data exploration through visualization (the name is derived from the phrase “visualization trails”), the system provides infrastructure for managing metadata and processes involved in the creation of data products in general, not just visualizations. VisTrails provides infrastructure that allows users to visually create, maintain, and explore large ensembles of workflows. The high-level architecture of the system is shown in Figure 2. Below, we briefly describe its key components.

The *Workflow Builder* provides a visual interface for creating and revising workflows (see Figure 4). As a user modifies a workflow, her actions are captured by the History Manager and saved in the *VisTrails Repository*. These actions include adding or deleting a module and setting the value of a parameter, and they capture detailed information about how a set of related workflows evolves over time. Users may interact with workflows by invoking them directly or by importing them into the *Visualization Spreadsheet*. The spreadsheet allows the results of different workflows to be compared side-by-side (see Figure 1a). In addition, the spreadsheet helps a user explore different parameter settings for a given workflow. The



(a)



(b)

Figure 1: The VisTrails Version Tree (a) captures the process followed to construct the Challenge workflows and the VisTrails Spreadsheet (b) allows the workflow outputs to be compared side by side.

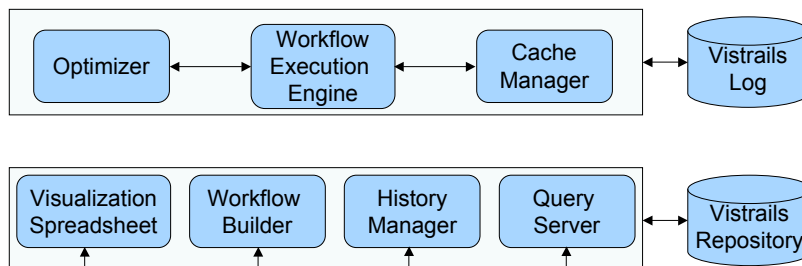


Figure 2: Simplified Architecture of VisTrails.

spreadsheet layout makes efficient use of screen space, and the row/column groupings help users explore the workflow parameter space [5].

Individual workflows are executed by the *Workflow Execution Engine*. Run-time information (e.g., which modules were executed, when, where and by whom) is saved in the *VisTrails Log*. Users may also query the saved information using the *Query Server*. More details about the query capabilities of VisTrails are presented in Section 3.

2.1. Change-Based Provenance

By recording the modifications users apply to a workflow, VisTrails maintains *detailed provenance of the exploration process*. This information not only ensures reproducibility, but it also allows scientists to easily navigate through the space of workflows and parameter settings used in a given exploration task. In particular, this gives them the ability to return to previous versions of a workflow and compare their results.

As shown in Figure 1a, VisTrails stores the trail of changes to workflows as a version tree—a vistrail, where each node corresponds to a *version* of a workflow and each directed edge corresponds to a change which transforms one workflow into another. Thus, each workflow is represented as a sequence of changes. This is similar to the versioning mechanism used in DARCS [18]. Powerful operations, like the ability to support collaborative data exploration in a distributed and disconnected fashion, are enabled by this representation and greatly simplify the scientific discovery process [5].

The change-based mechanism uniformly captures provenance of both the workflow evolution and associated data products. This in contrast to previous approaches, which have addressed only the problem of data provenance (see e.g., [16, 2, 20, 17]). Although data provenance is necessary to allow for reproducibility, it fails to capture useful information about the relationship among the different workflows used in an exploratory task. Currently, this information is often recorded manually, forcing users to save and manage a large number of files with different workflow specifications as well as data products. This process not only

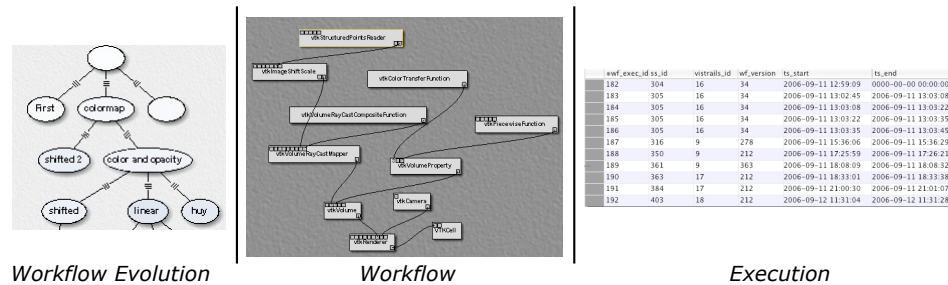


Figure 3: The Layered Provenance Model.

is time-consuming and error-prone, but it also greatly limits a scientist's ability to further explore the data.

Another advantage of the change-based mechanism is the compactness of its representation. Instead of individually storing a set of related workflows, it stores only the changes that are applied to workflows. This representation eliminates redundancy and leads to substantial space savings. Finally, by representing the provenance information in a structured way, the system allows the workflow provenance to be queried and mined. As we discuss in Section 3, this provenance structure enables the construction of an intuitive interface that allows scientists to both understand and interact with the provenance.

2.2. Organizing Provenance into Layers

As illustrated in Figure 3, the VisTrails provenance information is organized into three layers: workflow evolution, workflow, and execution. The *workflow evolution layer* captures the relationships among the series of workflows created in an exploratory task; the *workflow layer* consists of specifications of individual workflows; and the *execution layer* stores run-time information about the execution of workflow modules (e.g., execution time, machine name, date, etc.). The information for the first two layers is naturally captured by the change-based provenance mechanism. Run-time information is captured by the *Workflow Execution Engine* and stored in the *VisTrails Log* (see Figure 2). This structure clearly reflects the relationships among the different provenance components. The version tree stores a set of related workflow definitions, and each workflow definition is connected to a set of execution instances.

Structuring the provenance information into layers leads to a *normalized representation* that avoids the storage of redundant information. This is in contrast to other provenance mechanisms, where the information about the workflow specification is also saved in the execution log, i.e., a module name, parameter, and values are saved for each invocation of a given module [16, 2].

The layered provenance also makes the framework extensible. It allows layers to be replaced and new layers to be added. Since different execution engines may record different kinds of



execution provenance, the execution layer can be replaced with that of the execution engine. For example, for systems like Pegasus [7] that plan and schedule the execution of scientific workflows on a grid, it is useful to save the scheduling information as part of the execution provenance. In addition, to support higher level, semantic queries, it may be useful to add other layers of application-specific metadata and ontologies such as those in Taverna [14].

While the automatic capture of provenance information simplifies scientific discovery, it is also important to allow scientists to manually add their own provenance information. These pieces of additional information are referred to as *annotations*. Annotations are allowed at all levels of our layered provenance model. We should note that allowing annotations at the different layers was a requirement for answering the Challenge queries. The next section discusses our implementation of annotations.

3. Querying Provenance

The VisTrails provenance query language (vtPQL) is designed to take advantage of the structure of our layered provenance. Queries are broken into pieces that relate to versions, workflows, and executions. Each part of the query is tagged by a qualifier that specifies which layer it refers to. On the vistrail (**vt**) level, the search criteria for all workflow versions is specified; on the workflow (**wf**) level, users can search based on specific module or parameter information; on the execution (**log**) level, users can further refine queries to, for example, include only workflows that have run on specific architectures or during a given year. This piecewise query design simplifies both query specification and execution. Each level of the query is a simple SQL-like expression with some additional functions, predicates, and attributes (described below). These expressions allow standard set operators, boolean comparisons, boolean operators, and a regular expression matching function.

We have identified basic operations that are useful for common querying tasks over workflows, and that further simplify the query syntax. The operations used in the Challenge queries are described below. They take as input a workflow or set of workflows **x**.

- **upstream(x)** returns all modules that precede **x** in the workflow;
- **inputs(x)** returns all modules that immediately precede the given module(s) in the workflow; and
- **executed(x)** returns true if the module(s) have been executed (there is an entry in the VisTrails Log) or false otherwise.

We have also defined a set of predicates and attributes which return specific information about versions, modules, and executions. For example: **annotation(key)** returns the value associated with the given key. As discussed earlier, annotations are available at each layer, and therefore, the language supports searching for annotations at each of these levels. There are other predicates that are only available for specific layers: **vt:user** refers to the user who created a given workflow version; **wf:parameter(name)** refers to the module parameter specified by the given name; and **log:date** refers to the date a log entry was created.

The operational semantics for a vtPQL query corresponds to a nested loop over the objects returned from each layer. For example, the expression

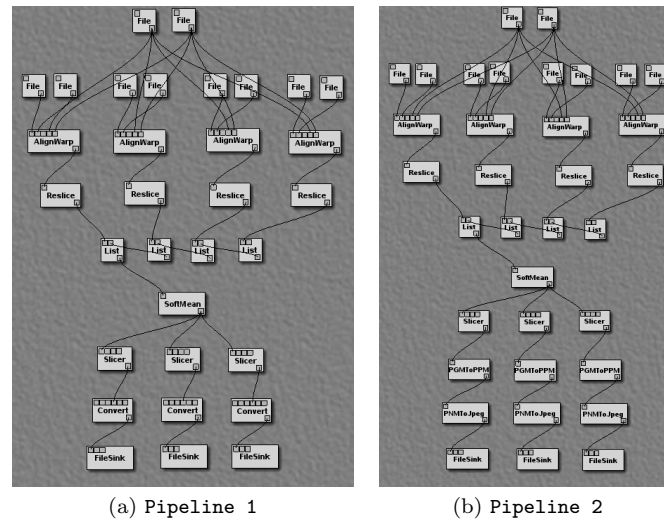


Figure 4: Challenge workflows. Pipeline 1 corresponds to the original workflow; and Pipeline 2 is the result of replacing the *convert* procedures in Pipeline 1 by the sequence *pgmtoppm*, *pnmtjpeg*. These workflows correspond to the nodes labeled *pipeline 1* and *pipeline 2* in Figure 1a.

```
wf{*}: x where x.module = AlignWarp and log{x}: y where y.date = '2006-09-15'
```

searches over all workflow versions (`wf{*}`) for workflows `x` that contain a module named `AlignWarp` and that have associated execution log entries `y` for the given date, i.e., workflows that were run on '2006-09-15'.

As we further develop this language, we plan to continue to emphasize both its simplicity and power. By using layered provenance, we are able to leverage the distinctions between the levels and break queries into more manageable and understandable pieces. By adding constructs for specific functionality (like the upstream operation), we are able to further simplify queries while retaining great flexibility.

4. Challenge: Implementation and Results

Collecting Provenance Data. To generate the provenance data required to answer the Challenge queries, we constructed and ran the two workflows in VisTrails. Using the VisTrails



Query 1	<code>wf{*}: upstream(x) union x where x.module = FileSink and x.parameter('name') = 'atlas-x.gif' and executed(x)</code>
Query 2	<code>wf{*}: (upstream(x) union x) - upstream(y) where x.module = FileSink and x.parameter('name') = 'atlas-x.gif' and executed(x) and y.module = SoftMean</code>
Query 3	<code>wf{*}: upstream(x) union x where x.module = FileSink and x.parameter('name') = 'atlas-x.gif' and x.annotation('stage') in {'3','4','5'} and executed(x)</code>
Query 4	<code>wf{*}: x where x.module = AlignWarp and x.parameter('model') = '12' and (log{x}: y where y.dayOfWeek = 'Monday')</code>
Query 5	<code>wf{*}: upstream(x) where x.module = FileSink and matches(x.parameter('name'), '.*atlas.*gif') and y in upstream(x) and y.module = File and (log{y}: z where z.annotation('globalmaximum') = '4095')</code>
Query 6	<code>wf{*}: upstream(x) union x where x.module = SoftMean and executed(x) and y in upstream(x) and y.module = AlignWarp and y.parameter('model') = '12'</code>
Query 7	(operation is a feature of Workflow Builder—see result in Figure 5b)
Query 8	<code>wf{*}: upstream(x) union x where x.module = AlignWarp and y in inputs(x) and y.annotation('center') = 'UChicago'</code>
Query 9	<code>wf{*}: x where x.module = File and x.annotation('studyModality') in {'speech', 'visual', 'audio'}</code>

Table I. Challenge Queries

plug-in mechanism, we were able to quickly wrap the AIR* and FSL† suites used in the workflows. By wrapping these tools and importing the wrappers into VisTrails, their functions are automatically (and dynamically) made available as modules in the Workflow Builder. The Challenge workflows, shown in Figure 4, were then constructed and executed. The provenance information gathered consisted of the version tree (which contains the specifications for both workflows), and the execution log. In our current prototype, the version tree is saved in an XML file and the log is saved onto a set of MySQL tables.

Executing Queries. Table I shows the nine Challenge queries expressed in our language. We implemented a Python API that supports the basic constructs of vtPQL. Using this API, it was straightforward to write the queries. It is worthy of note that Query 7 was already supported in VisTrails, through the visual difference interface.

Displaying Query Results. In addition to providing information about the results (e.g., workflow identifiers and attributes), VisTrails has the ability to visually display query results by highlighting the workflows and modules (at the vistrail and workflow levels, respectively) that satisfy the query. Due to space limitations, we discuss only the results for two of the Challenge queries. More details on the other queries are available in [21]. The results for Query 4 are shown in Figure 5a. All the workflow versions that match the query conditions are highlighted in the version tree. If the user clicks on a given version, the workflow is displayed and the modules that match the query are highlighted. The user can then click on the individual modules to view execution log records associated with modules. The versions, modules, and parameters returned by a query can be explored interactively by the user. For example, if a query asks for a specific workflow fragment, like the signal processing piece of a complicated workflow, we

*<http://air.bmap.ucla.edu/AIR5>†<http://www.fmrib.ox.ac.uk/fsl>

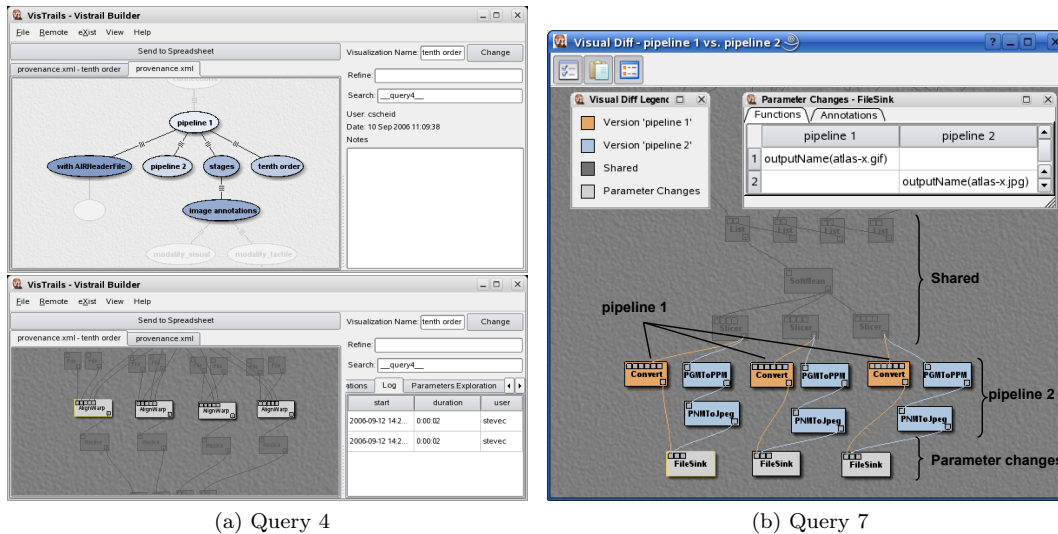


Figure 5: The visual answers for two of the queries

allow the user to extract and run this fragment. Note that this is only possible because of the inherent connectivity between workflow representation and provenance in VisTrails.

As noted in Table I, we use the VisTrails' visual difference interface to answer Query 7. In the version tree, by dragging one workflow over another, VisTrails computes the differences between the workflow specifications. As Figure 5b shows, these differences are displayed visually via color cues: blue and orange colors represent modules present in one workflow but not the other, dark-grey indicates modules that are present in both workflows, and light-grey depicts modules whose parameter values differ.

5. Discussion and Future Work

In this paper, we give an overview of the VisTrails provenance management framework and how it was used to answer the Provenance Challenge queries. There are important features of our provenance approach that set it apart from the approaches used by the other Challenge participants. Instead of requiring users to write complicated SQL or Prolog syntax, we have designed a new, a domain-specific language tailored for querying workflow provenance. This language leads to concise queries that are easy to write and understand. The simplicity, however, does not sacrifice expressiveness: the language can express all of the Challenge queries. In addition, VisTrails provides a way to visually display and explore the results of queries in the same framework used to create the workflows.



The VisTrails system provides a flexible and extensible infrastructure for integrating new tools and libraries, creating workflows, and managing their provenance. Our component-based architecture makes it possible to integrate the VisTrails provenance framework with existing scientific workflow [16, 22, 10] and workflow-based visualization systems [11, 15]. In addition, the query language, while designed according to the layered provenance model, is independent from the provenance representation, allowing it to be used with other provenance data. We plan to further investigate interoperability issues in the future.

While VisTrails automatically captures the provenance data for workflow evolution and executions, it does not capture information about temporary files that may be created during workflow execution, but that are not specified in the workflow. For example, the AIR tools used in the Challenge workflows create and depend on temporary files which are not explicitly defined in the workflows. To properly capture this information in the VisTrails execution log, these tools need wrapped appropriately. In contrast, systems like ES3 [9] and PASS [19], in contrast, which track workflow execution at the operating system level, are able to detect the creation of such files. It should be noted, however, that these systems require tools that depend on specific operating systems, thus, machines need to be properly configured to make use of these systems.

The VisTrails system was released under the GPL license and can be downloaded from <http://www.vistrails.org>.

Acknowledgments. We thank Erik Anderson and Nathan Smith for their contributions to the VisTrails system. This work was partially supported by the National Science Foundation under grants IIS-0513692, IIS-0534628, OCE-0424602, CCF-0401498, CNS-0524096, and OISE-0405402, the Department of Energy, an IBM Faculty Award, and a University of Utah Seed Grant.

REFERENCES

1. Apple's Mac OS X Automator. <http://www.apple.com/downloads/macosx/automator>.
2. R. S. Barga and L. A. Digiampietri. Automatic capture and efficient storage of escience experiment provenance. *Concurrency and Computation: Practice and Experience*, 2007.
3. L. Bavoil, S. P. Callahan, P. J. Crossno, J. Freire, C. Scheidegger, C. Silva, and H. Vo. Vistrails: Enabling interactive multiple-view visualizations. In *IEEE Visualization 2005*, pages 135–142, 2005.
4. S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo. Managing the evolution of dataflows with visTrails (*Extended Abstract*). In *IEEE Workshop on Workflow and Data Flow for Scientific Applications (SciFlow)*, 2006.
5. S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegge, and H. T. Vo. Using provenance to streamline data exploration through visualization. Technical Report UUSCI-2006-016, SCI Institute, University of Utah, 2006.
6. E. Deelman and Y. Gil. NSF Workshop on Challenges of Scientific Workflows. Technical report, NSF, 2006. http://vtcpc.isi.edu/wiki/index.php/Main_Page.
7. E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems. *Scientific Programming Journal*, 13(3):219–237, 2005.
8. J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *International Provenance and Annotation Workshop (IPAW)*, LNCS 4145, pages 10–18. Springer Verlag, 2006.
9. J. Frew, D. Metzger, and P. Slaughter. Automatic capture and reconstruction of computational provenance. *Concurrency and Computation: Practice and Experience*, 2007.



10. J. Kim, E. Deelman, Y. Gil, G. Mehta, and V. Ratnakar. Provenance trails in the wings/pegasus system. *Concurrency and Computation: Practice and Experience*, 2007.
11. Kitware. The Visualization Toolkit. <http://www.vtk.org>.
12. MSFT Workflow Foundation. <http://msdn2.microsoft.com/en-us/netframework/aa663328.aspx>.
13. L. Moreau, B. Ludäscher, I. Altintas, R. S. Barga, S. Bowers, S. Callahan, G. Chin Jr., B. Clifford, S. Cohen, S. Cohen-Boulakia, S. Davidson, E. Deelman, L. Digiampietri, I. Foster, J. Freire, J. Frew, J. Futrelle, T. Gibson, Y. Gil, C. Goble, J. Golbeck, P. Groth, D. A. Holland, S. Jiang, J. Kim, D. Koop, A. Krenek, T. McPhillips, G. Mehta, S. Miles, D. Metzger, S. Munroe, J. Myers, B. Plale, N. Podhorszki, V. Ratnakar, E. Santos, C. Scheidegger, K. Schuchardt, M. Seltzer, Y. L. Simmhan, C. Silva, P. Slaughter, E. Stephan, R. Stevens, D. Turi, H. Vo, M. Wilde, J. Zhao, and Y. Zhao. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience*, 2007.
14. T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: lessons in creating a workflow environment for the life sciences: Research articles. *Concurrency and Computation: Practice & Experience*, 18(10):1067–1100, 2006.
15. S. G. Parker and C. R. Johnson. SCIRun: a scientific programming environment for computational steering. In *Supercomputing*, page 52, 1995.
16. N. Podhorszki, B. Ludaescher, I. Altintas, S. Bowers, and T. McPhillips. Recording data provenance for kepler scientific workflows. *Concurrency and Computation: Practice and Experience*, 2007.
17. The EU Provenance Project. <http://twiki.gridprovenance.org/bin/view/Provenance>.
18. D. Roundy. Darcs. <http://abridgegame.org/darcs>.
19. M. Seltzer, D. A. Holland, U. Braun, and K.-K. Muniswamy-Reddy. Pass-ing the provenance challenge. *Concurrency and Computation: Practice and Experience*, 2007.
20. Y. L. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.
21. VisTrails—Provenance Challenge, 2006. <http://twiki.ipaw.info/bin/view/Challenge/VisTrails>.
22. J. Zhao, C. Goble, R. Stevens, and D. Turi. Mining taverna’s semantic web of provenance. *Concurrency and Computation: Practice and Experience*, 2007.