

Provenance in Scientific Workflow Systems

Susan Davidson,
Sarah Cohen-Boulakia, Anat Eyal
University of Pennsylvania
{susan, sarahcb, anate}@cis.upenn.edu

Bertram Ludäscher, Timothy McPhillips
Shawn Bowers, Manish Kumar Anand
University of California, Davis
{ludaesch, tmcpillips, sbowers, maanand}@ucdavis.edu

Juliana Freire
University of Utah
juliana@cs.utah.edu

Abstract

The automated tracking and storage of provenance information promises to be a major advantage of scientific workflow systems. We discuss issues related to data and workflow provenance, and present techniques for focusing user attention on meaningful provenance through “user views,” for managing the provenance of nested scientific data, and for using information about the evolution of a workflow specification to understand the difference in the provenance of similar data products.

1 Introduction

Scientific workflow management systems (e.g., myGrid/Taverna [18], Kepler [6], VisTrails [13], and Chimera [12]) have become increasingly popular as a way of specifying and executing data-intensive analyses. In such systems, a workflow can be graphically designed by chaining together tasks (e.g., for aligning biological sequences or building phylogenetic trees), where each task may take input data from previous tasks, parameter settings, and data coming from external data sources. In general, a workflow specification can be thought of as a graph, where nodes represent *modules* of an analysis and edges capture the *flow of data* between these modules.

For example, consider the workflow specification in Fig. 1(a), which describes a common analysis in molecular biology: *Inference of phylogenetic (i.e., evolutionary) relationships between biological sequences*. This workflow first accepts a set of sequences selected by the user from a database (such as GenBank), and supplies the data to module M1. M1 performs a multiple alignment of the sequences, and M2 refines this alignment. The product of M2 is then used to search for the most parsimonious phylogenetic tree relating the aligned sequences. M3, M4, and M5 comprise a loop sampling the search space: M3 provides a random number seed to M4, which uses the seed together with the refined alignment from M2 to create a set of phylogenetic trees. M5 determines if the search space has been adequately sampled. Finally, M6 computes the consensus of the trees output from the loop. The dotted boxes M7, M8 and M9 represent the fact that *composite modules* may be used to create the workflow. That is, M7 is itself a workflow representing the alignment process, which consists of modules M1 and M2; M8 is a workflow representing the initial phylogenetic tree construction process, which consists of modules M3, M4, and M5; and M9 is a composite module representing the entire process of creating the consensus tree, which consists of modules M3, M4, M5 and M6.

The result of executing a scientific workflow is called a *run*. As a workflow executes, data flows between module *invocations* (or *steps*). For example, a run of the phylogenetics workflow is shown in Fig. 1(b). Nodes

Copyright 2007 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

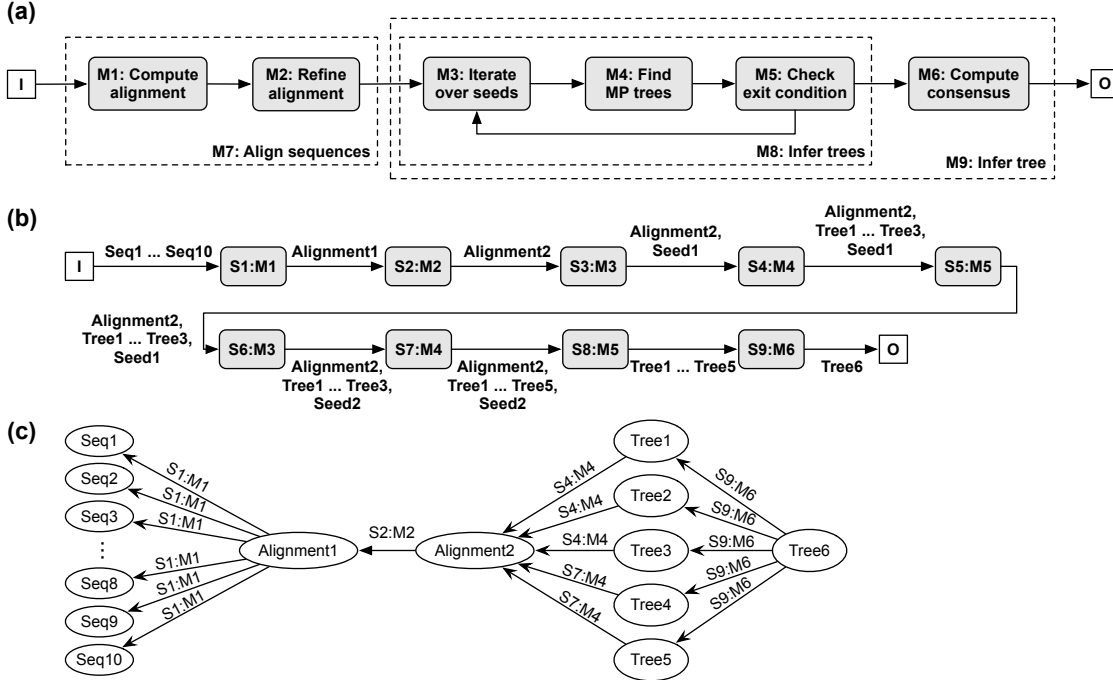


Figure 1: Phylogenetics workflow specification, run, and data dependency graph.

in this run graph represent steps that are labeled by a unique step identifier and a corresponding module name (e.g., S1:M1). Edges in this graph denote the flow of data between steps, and are labeled accordingly (e.g., data objects Seq_1, \dots, Seq_{10} flow from input I to the first step S1). Note that loops in the workflow specification are always unrolled in the run graph, e.g., two steps S4 and S7 of M4 are shown in the run of Fig. 1(b).

A given workflow may be executed multiple times in the context of a single project, generating a large amount of final and intermediate data products of interest to the user [9]. When such analyses are carried out by hand or automated using general-purpose scripting languages, the means by which results are produced are typically not recorded automatically, and often not even recorded manually. Managing such provenance information is a major challenge for scientists, and the lack of tools for capturing such information makes the results of data-intensive analyses difficult to interpret, to report accurately, and to reproduce reliably. Scientific workflow systems, however, are ideally positioned to record critical provenance information that can authoritatively document the lineage of analytical results. Thus, the ability to capture, query, and manage provenance information promises to be a major advantage of using scientific workflow systems. Provenance support in scientific workflows is consequently of paramount and increasing importance, and the growing interest in this topic is evidenced by recent workshops [4, 17] and surveys [5, 19] in this area.

Data provenance in workflows is captured as a set of dependencies between data objects. Fig. 1(c) graphically illustrates a subset of the dependencies between data objects for the workflow run shown in Fig. 1(b). In such data-dependency graphs, nodes denote data objects (e.g., Tree4) and dependency edges are annotated with the step that produced the data. For example, the dependency edge from Alignment2 to Alignment1 is annotated with S2:M2 to indicate that Alignment2 was produced from Alignment1 as a result of this step.

Many scientific-workflow systems (e.g., myGrid/Taverna) capture provenance information implicitly in an event log. For example, these logs record events related to the start and end of particular steps in the run and corresponding data read and write events. Using the (logical) order of events, dependencies between data objects processed or created during the run can be inferred¹. Thus, determining data dependencies in scientific

¹The complexity of the inference procedure and type of log events required depends on the specific model of computation used to

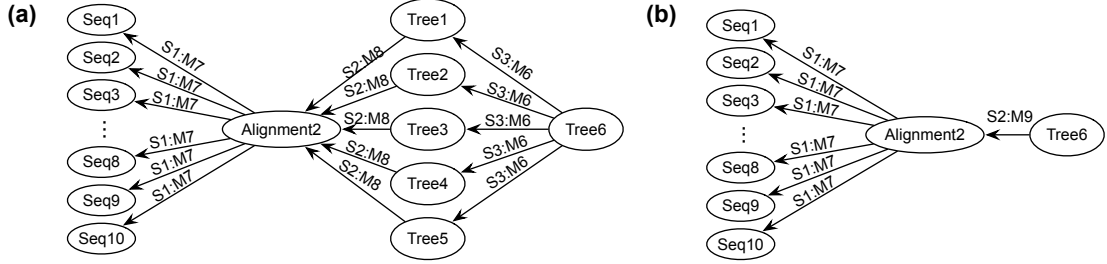


Figure 2: Provenance of `Tree6` in Joe's (a) and Mary's (b) user views.

workflow systems generally is performed using dynamic analysis, *i.e.*, modules are treated as “black boxes” and dependency information is captured as a workflow executes. In contrast, determining provenance information from database views (or queries) can be performed using static analysis techniques [10]. In this case, database queries can be viewed as “white box” modules consisting of algebraic operators (*e.g.*, σ , π , \bowtie). An intermediate type of provenance can also be considered in which black-box modules are given additional annotations specifying input and output constraints, thus making them “grey boxes” [6]. These additional specifications could then be used to reconstruct provenance dependencies using static analysis techniques, without requiring runtime provenance recording.

The use of provenance in workflow systems also differs from that in database systems. Provenance is not only used for interpreting data and providing reproducible results, but also for troubleshooting and optimizing efficiency. Furthermore, the application of a scientific workflow specification to a particular data set may involve tweaking parameter settings for the modules, and running the workflow many times during this tuning process. Thus, for efficiency, it is important to be able to revisit a “checkpoint” in a run, and re-execute the run from that point with new parameter settings, re-using intermediate data results unaffected by the new settings. The same information captured to infer data dependencies for a run can also be used to reset the state of a workflow system to a checkpoint in the past or to optimize the execution of a modified version of a workflow in the future.

While the case for provenance management in scientific workflow systems can easily be made, real-world development and application of such support is challenging. Below we describe how we are addressing three provenance-related challenges: First, we discuss how composite modules can be constructed to provide provenance “views” relevant to a user [3]. Second, we discuss how provenance for complex data (*i.e.*, nested data collections) can be captured efficiently [7]. Third, we discuss how the evolution of workflow specifications can be captured and reasoned about together with data provenance [13].

2 Simplifying provenance information

Because a workflow run may comprise many steps and intermediate data objects, the amount of information provided in response to a provenance query can be overwhelming. Even for the simple example of Fig. 1, the provenance for the final data object `Tree6` is extensive.² A user may therefore wish to indicate which modules in the workflow specification are *relevant*, and have provenance information presented with respect to that user view. To do this, composite modules are used as an abstraction mechanism [3].

For example, user Joe might indicate that the `M2: Refine alignment`, `M4: Find MP trees`, and `M6: Compute consensus` modules are relevant to him. In this case, composite modules `M7` and `M8` would automatically be constructed as shown in Fig. 1(a) (indicated by dotted lines), and Joe's user view would be $\{M7, M8, M6\}$. When answering provenance queries with respect to a user view, only data passed between modules in the user

execute a workflow, *e.g.*, see [8].

²The graph shown in Fig. 1(c) is only partial, and omits the seeds used in `M4` as well as additional notations of `S7:M4` on the edges from `Tree1, ..., Tree3` to `Alignment2`.

view would be visible; data internal to a composite module in the view would be hidden. The provenance for `Tree6` presented according to Joe’s user view is shown in Fig. 2(a). Note that `Alignment1` is no longer visible.

More formally, a *user view* is a partition of the workflow modules [3]. It induces a “higher level” workflow in which nodes represent composite modules in the partition (*e.g.*, M7 and M8) and edges are induced by dataflow between modules in different composite modules (*e.g.*, an edge between M7 and M8 is induced by the edge from M2 to M3 in the original workflow). Provenance information is then seen by a user with respect to the flow of data between modules in his view. In the `Zoom*UserViews` system [2], views are constructed automatically given input on what modules the user finds relevant such that (1) a composite module contains at most one relevant (atomic) module, thus assuming the “meaning” of that module; (2) no data dependencies (either direct or indirect) are introduced or removed between relevant modules; and (3) the view is minimal. In this way, the meaning of the original workflow specification is preserved, and only relevant provenance information is provided to the user.

Note that user views may differ: Another user, Mary, may only be interested in the modules M2: *Refine alignment* and M6: *Compute consensus*. Mary’s user view would therefore be constructed as {M7, M9}, and her view for the provenance of `Tree6` (shown in Fig. 2(b)) would not expose `Tree1 ... Tree5`.

3 Representing provenance for nested data collections

Modules within scientific workflows frequently operate over collections of data to produce new collections of results. When carried out one after the other, these operations can yield increasingly nested data collections, where different modules potentially operate over different nesting levels. The *collection-oriented modeling and design* (COMAD) framework [16] in Kepler models this by permitting data to be grouped explicitly into nested collections similar to the tree structure of XML documents. These trees of data are input, manipulated, and output by collection-aware modules. However, unlike a general XML transformer, a COMAD module generally preserves the structure and content of input data, accessing particular collections and data items of relevance to it, and adding newly computed data and new collections to the data structure it received. COMAD workflow designers declare the *read scope* and *write scope* for each module while composing the workflow specification. A read scope specifies the type of data and collections relevant to a module using an XPath-like expression to match one or more nodes on each invocation; paths may be partially specified using wildcards and predicates. As an example, the read scope for M1 could be given as `Proj/Trial/Seqs`, which would invoke M1 over each collection of sequences in turn. A write scope specifies where a module should add new data and collections to the stream. Data and collections that fall outside a module’s read scope are automatically forwarded by the system to succeeding modules, enabling an “assembly-line” style of data processing.

Similar to other dataflow process networks [15], modules in a COMAD workflow work *concurrently* over items in the data stream. That is, rather than supplying the entire tree to each module in turn, COMAD streams the data through modules as a sequence of tokens. Fig. 3 illustrates the state of a COMAD run of the example workflow shown in Fig. 1 at a particular point in time, and contrasts the logical organization of the data flowing through the workflow in Fig. 3(a) with its tokenized realization at the same point in time in Fig. 3(b). This figure further illustrates the pipelining capabilities of COMAD by including two independent sets of sequences in a single run. This pipeline concurrency is achieved in part by representing nested data collections at runtime as “flat” token streams containing paired opening and closing delimiters to denote collection membership.

Fig. 3 also illustrates how data provenance is captured and represented at runtime. As COMAD modules insert new data and collections into the data stream, they also insert metadata tokens containing explicit data-dependency information. For example, the fact that `Alignment2` was computed from `Alignment1` is stored in the insertion-event metadata token immediately preceding the A2 data token in Fig. 3(b), and displayed as the dashed arrow from A2 to A1 in Fig. 3(a). The products of a COMAD workflow may be saved as an

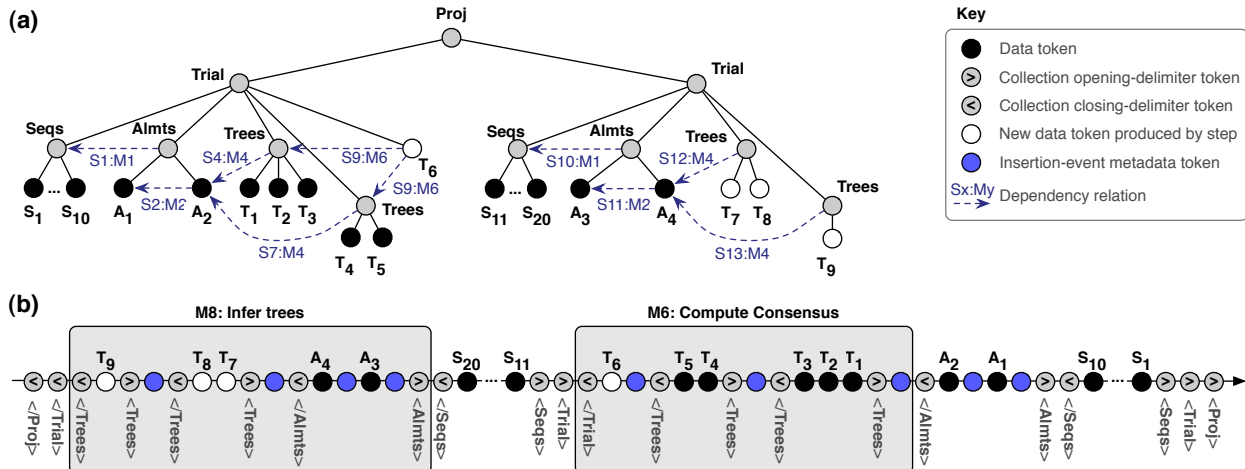


Figure 3: An intermediate state of a COMAD run

XML-formatted trace file, in which provenance records are embedded directly within the file as XML elements annotating data and collection elements. Detailed data dependencies can be inferred from the trace file, *i.e.*, from the embedded provenance annotations together with the nested data collections output by the workflow run. Note that COMAD can minimize the number and size of provenance annotations as described in [7, 9]. For example, when a module inserts a node that is a collection, the provenance information for that node implicitly cascades to all descendant nodes. Similarly, if a node is derived from a collection node, an insertion annotation is created that refers just to the collection identifier rather than the various subnodes.

The current COMAD implementation includes a prototype subsystem for querying traces. The system provides basic operations for accessing trace nodes, constructing dependency relations, and querying corresponding dependency graphs over the XML trace files. Methods also are provided to reconstruct parameter settings and metadata annotations attributed to data and collection nodes [7].

4 Workflow evolution

Scientific workflows dealing with data exploration and visualization are frequently exploratory in nature, and entail the investigation of parameter spaces and alternative techniques. A large number of related workflows are therefore created in a sequence of iterative refinements of the initial specification, as a user formulates and tests hypotheses. VisTrails [13] captures detailed information about this refinement process: As a user modifies a workflow, it transparently captures the change actions, *e.g.*, the addition or deletion of a module, the modification of a parameter, the addition of a connection between modules,³ akin to a database transaction log. The history of change actions between workflow refinements is referred to as a visual trail, or a *vistrail*.

The change-based representation of workflow evolution is concise and uses substantially less space than the alternative of storing multiple versions of a workflow. The model is also extensible. The underlying algebra of actions can be customized to support change actions at different granularities (*e.g.* composite modules versus atomic modules). In addition, it enables construction of an intuitive interface in which the evolution of a workflow is presented as a tree, allowing scientists to return to a previous version in an intuitive way, to undo bad changes, and be reminded of the actions that led to a particular result.

Vistrails and data provenance interact in a subtle but important way: The vistrail can be used to explain the difference in process between the data provenance of similar data products. Returning to our example,

³Modules are connected by input/output *ports*, which carry the data type and meaning. Static type-checking can be therefore performed to help in debugging.

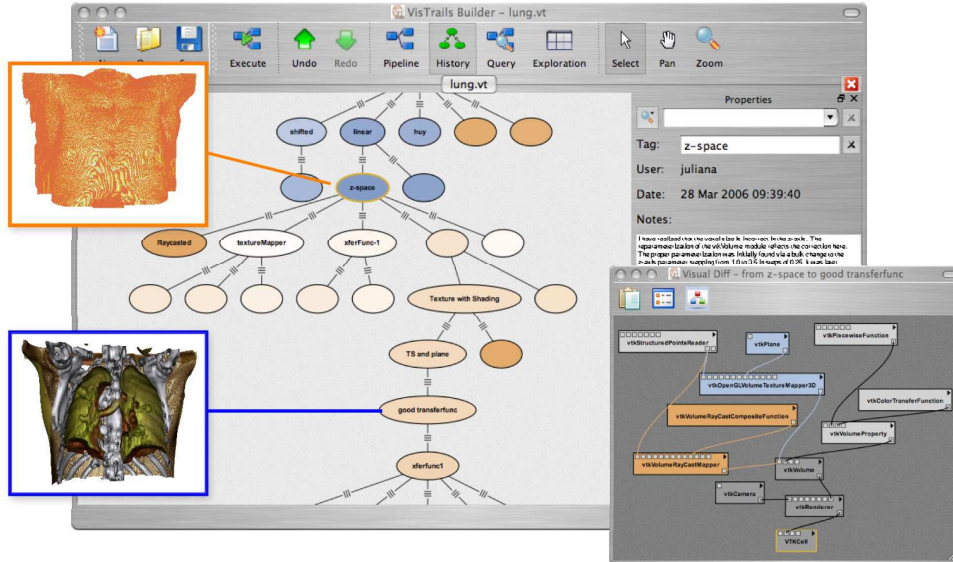


Figure 4: Visual difference interface for a radiation treatment planning example.

suppose that two runs of the workflow in Fig. 1 took as input the same set of sequences, but returned two different final trees. Furthermore, suppose that the specification was modified between the two runs, e.g. that a different alignment algorithm was used in M1, or that three iterations of the loop were performed in M8 due to different seeds being used. Rather than merely examining the data provenance of each tree, the scientist may wish to compare their provenance and better understand *why* the final data differed. However, computing the differences between two workflows by considering their underlying graph structure is impractical; the related decision problem of *subgraph isomorphism (or matching)* is known to be NP-complete [14]. By capturing evolution explicitly in a vistrail, discovering the difference in process is simplified: The two workflow nodes are connected by a path in the vistrail, allowing the difference between two workflows to be efficiently calculated by comparing the sequences of change actions associated with them [11].

Figure 4 (right) shows the visual difference interface provided by VisTrails. A visual difference is enacted by dragging one node in the history tree onto another, which opens a new window with a difference workflow. Modules unique to the first node are shown in orange, modules unique to the second node in blue, modules that are the same in dark gray, and modules that have different parameter values in light gray. Using this interface, users can correlate differences between two data products with differences between their corresponding specifications.

5 Conclusion

Workflow systems are beginning to implement a “depends-on” model of provenance, either by storing the information explicitly in a database (e.g., VisTrails) or within the data itself (e.g., COMAD). Several techniques have also been proposed to reduce the amount of provenance information either presented to the user (e.g., user views), or stored by the database (e.g., by treating data as collections). Furthermore, since workflow specifications evolve over time, there is a need to understand not only the provenance of a single data item but how the provenance of related data items differ.

Although some workflow systems provide a query interface for interacting with the provenance information, it is still an open problem as to what a provenance query language should provide. For example, we might wish to scope provenance information within a certain specified portion of a workflow, or return all provenance information that satisfies a certain execution pattern. The query language should also allow users to issue high

level queries using concepts that are familiar to them, and present the results in an intuitive manner. Related work in this area has been done in the context of business processing systems, in which runs are monitored by querying logs (*e.g.*, [1]).

References

- [1] C. Beeri, A. Pilberg, T. Milo, and A. Eyal. Monitoring business processes with queries. In *VLDB*, 2007.
- [2] O. Biton, S. Cohen-Boulakia, and S. Davidson. Zoom*UserViews: Querying relevant provenance in workflow systems (demo). In *VLDB*, 2007.
- [3] O. Biton, S. Cohen-Boulakia, S. Davidson, and C. Hara. Querying and managing provenance through user views in scientific workflows. In *ICDE*, 2008 (to appear).
- [4] R. Bose, I. Foster, and L. Moreau. Report on the International Provenance and Annotation Workshop. *SIGMOD Rec.*, 35(3), 2006.
- [5] R. Bose and J. Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comp. Surveys*, 37(1):1–28, 2005.
- [6] S. Bowers and B. Ludäscher. Actor-oriented design of scientific workflows. In *ER*, 2005.
- [7] S. Bowers, T. M. McPhillips, and B. Ludäscher. Provenance in collection-oriented scientific workflows. In *Concurrency and Computation: Practice and Experience*. Wiley, 2007 (in press).
- [8] S. Bowers, T. M. McPhillips, B. Ludäscher, S. Cohen, and S. B. Davidson. A model for user-oriented data provenance in pipelined scientific workflows. In *IPAW*, volume 4145 of *LNCS*, pages 133–147. Springer, 2006.
- [9] S. Bowers, T. M. McPhillips, M. Wu, and B. Ludäscher. Project histories: Managing provenance across collection-oriented scientific workflow runs. In *Data Integration in the Life Sciences*, 2007.
- [10] P. Buneman and W. Tan. Provenance in databases. In *SIGMOD*, pages 1171 – 1173, 2007.
- [11] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo. Using provenance to streamline data exploration through visualization. Technical Report UUSCI-2006-016, SCI Institute–University of Utah, 2006.
- [12] I. Foster, J. Vockler, M. Woilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *SSDBM*, pages 37–46, 2002.
- [13] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *IPAW*, 2006.
- [14] J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182:105–142, 1999.
- [15] E. A. Lee and T. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.
- [16] T. M. McPhillips, S. Bowers, and B. Ludäscher. Collection-oriented scientific workflows for integrating and analyzing biological data. In *Data Integration in the Life Sciences*, 2006.
- [17] L. Moreau and B. Ludäscher, editors. *Concurrency and Computation: Practice and Experience – Special Issue on the First Provenance Challenge*. Wiley, 2007 (in press). (see also <http://twiki.ipaw.info/bin/view/Challenge/>).
- [18] T. Oinn *et al.* Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(1), 2003.
- [19] Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, 2005.