On User Views in Scientific Workflow Systems

Susan Davidson
University of Pennsylvania
Email: susan@cis.upenn.edu

Yi Chen, Peng Sun
Arizona State University
Email: {yi, psun5}@asu.edu

Sarah Cohen-Boulakia
Universite Paris-Sud
Email: cohen@lri.fr

***Abstract*—An increasing number of scientific workflow systems are providing support for the automated tracking and storage of provenance information. However, the amount of provenance information recorded can become very large, even for a single execution of a workflow – [6] estimates a ten-fold blowup of the size of the original input data. There is therefore a need to provide ways of allowing users to focus their attention on meaningful provenance information in provenance queries. We highlight recent work in this area on *user views*, showing how they can be efficiently computed given user input on relevance, or and how pre-existing views can be corrected to provide accurate provenance information. We also discuss how to search a repository of workflow specifications and their views, returning workflows at an appropriate level of complexity with respect to a hierarchy of views.**

I. Introduction

Scientific workflow management systems (*e.g.*, myGrid/Taverna [11], Kepler [5], VisTrails [9], and Chimera [8]) have become increasingly popular as a way of specifying and executing data-intensive analyses. To ensure reproducibility of results and track the large amount of final and intermediate data products that are produced in a workflow execution, many of these systems are beginning to provide support for managing and querying provenance information.

However, the amount of provenance information recorded even for a single execution of a workflow can be extremely large; [6] estimates a ten-fold blowup of the size of the original input data. While databases are adept at storing and efficiently answering queries over large amounts of information, users are not adept at *assimilating* large amounts of information. It is therefore important to develop techniques to minimize the cognitive overload resulting from provenance queries, providing provenance information that is relevant to users.

As an example, consider the workflow specification (a.k.a workflow definition or schema) in Fig. 1 (a)[1], which describes a common analysis in molecular biology: *Phylogenomic inference of protein biological function*. This workflow first takes in a set of entries selected by the user from a database (such as GenBank), and formats these entries to extract a set of sequences, and, possibly, a set of annotations (M1). An alignment is then created (M3), and the result formatted (M4). The user may also be interested in rectifying the alignment (M5). M3 to M5 are repeated until the biologist is satisfied with the result obtained. The user may also inspect the annotations provided by GenBank (M2) and generate a set of curated annotations; new user input is needed for this. The annotations are then formatted (M8) to be taken as input to the phylogenetic tree reconstruction task (M7). Other annotations are also considered: M6 takes in annotations from the user's lab and formats them to be taken as input to M7. From the annotations produced by M8 (and possibly M6) together with the alignment produced by M4, M7 provides a phylogenetic tree labeled with functional annotations. Note that a number of these tasks or *modules* (e.g. M1, M4, M8) involve formatting and are not central to the scientific goal of the experiment, and that edges represent the precedence and potential *dataflow* between modules during an execution.

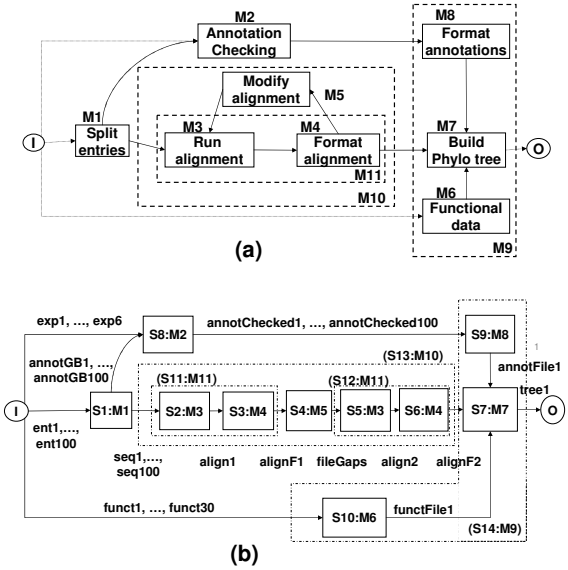

Fig. 1. Phylogenetic workflow specification (a) and run (b)

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 phylogenomics workflow is shown in Fig. 1(b). Nodes represent steps that are labeled by a unique step identifier and a corresponding module name (*e.g.*, S1:M1). Edges denote the flow of data between steps, and are labeled accordingly (*e.g.*, data objects `ent1`,...,`ent10` 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 S3 and S6 of M4 are shown in the run of Fig. 1(b).

[1]The reader should ignore the dotted boxes for now.

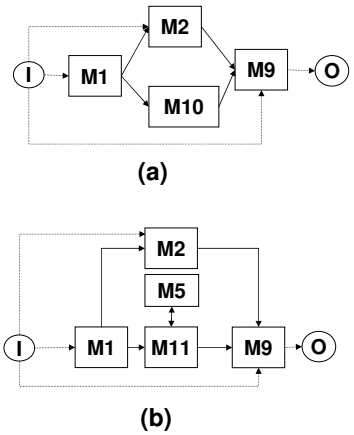

Fig. 2. Joe's (a) and Mary's (b) user views.

Data provenance in workflows is typically captured as a set of dependencies between data objects [7]. Essentially, the graph of Fig. 1(b) becomes one in which the nodes are data; each edge is labeled with the module execution which produced the data at its start and taking as one of its inputs the data at its end. Note that module names are repeated for every input-output pair. Thus, a query of the provenance of the final data product `tree1` would return a graph similar to that in Fig. 1(b), which (even for this simple example) is quite large.

In this paper, we discuss a technique called *user views* which uses composite modules, i.e. modules which may themselves contain subworkflows, to hide portions of a workflow run and thus simplify the workflow specification as well as provenance information. Section II shows how users can indicate which modules are relevant within a specification, and have a user view automatically created around those relevant modules. Section III discusses how to refine pre-defined views to ones which correctly portray the provenance relationships between the input and output of composite modules. Section IV shows how a database of specifications and their views can be searched using keyword queries, returning workflows at an appropriate level of complexity with respect to a hierarchy of views.

II. User Views

As illustrated in Fig. 1 (b), a workflow run may comprise many steps and intermediate data objects, and therefore the amount of information provided in response to a provenance query can be overwhelming. 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].

As an example, for the workflow in Fig. 1 (a), user Joe might indicate that M2: *Annotation Checking*, M3: *Run Alignment*, and M7: *Build Phylo Tree* are relevant to him. In this case, composite modules M9 and M10 would automatically be constructed (indicated by dotted boxes labeled M9 and M10 in Fig. Fig. 1(a)), and Joe's user view would be {M1, M2, M9, M10} as shown in Fig. 2(a). When answering provenance queries with respect to a user view, only data passed between modules in the user view would be visible. Data and module executions internal to a composite module in the view would be hidden; this corresponds to hiding the module executions and data shown within the dotted boxes M9 and M10 in Fig. 1(b). Thus, the provenance for `tree1` presented according to Joe's user view would no longer include `annotFile1`, `functFile1` (both pieces of data are hidden inside M9), or `align1`, `alignF1`, `fileGaps`, `align2` (hidden inside M10).

Views are individualized according the user's interests. For example, another user, Mary, may be interested in modules M2, M3 and M7 (like Joe) but additionally interested in M5: *Modify alignment*. Mary's user view would therefore be constructed as {M1, M2, M5, M9, M11} (shown in Fig. 2(b)), and her view for the provenance of `tree1` would expose `alignF1` and `fileGaps` (unlike Joe's view) while hiding `annotFile1`, `functFile1`, `align1`, and `align2`.

More formally, a *user view* is a partition of the workflow modules. It induces a "higher level" workflow in which nodes represent composite modules in the partition (*e.g.*, M9 and M10) and edges are induced by dataflow between modules in different composite modules (*e.g.*, an edge between M10 and M9 is induced by the edge from M4 to M7 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 system [3], [2], user 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; and (2) no data dependencies (either direct or indirect) are introduced or removed between relevant modules. In this way, the meaning of the original workflow specification is preserved, and only relevant provenance information is provided to the user.

An interesting theoretical question is whether there are efficient algorithms for constructing a user view which obeys conditions (1) and (2) above and which are as small as possible, i.e. in which the number of composite modules is *minimized*. We call such a view *optimum*. It turns out that whether or not the optimum user view can be constructed depends on the graphical structure of the workflow specification [4]: For specifications that are general graphs, regardless of the number of distinct modules in the input workflow and the structure of interaction between them, the number of composite modules can be exponentially large in the number of relevant modules in an optimum user view for the specification. However, for *series-parallel* workflow graphs [14] there is a simple, linear time algorithm for constructing an optimum user view for a given specification [4]. A study of scientific workflow specifications collected from our collaborators as well as those found at `myexperiment.org` has shown that over 80% of scientific workflow specifications are series-parallel graphs,

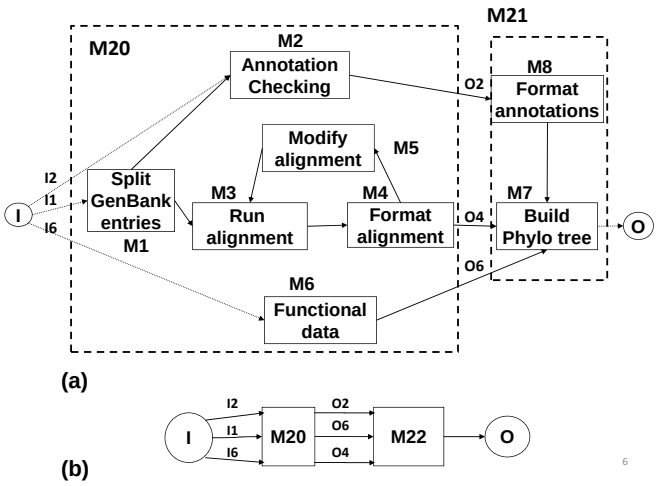

Fig. 3. Unsound User View (a) and Projected View (b)

and those that are not can be easily transformed by adding control points.

III. Correcting Composite Modules for Provenance

The previous section focused on how to create views give user input on what modules were relevant. However, composite modules are frequently used for purposes of modularization, abstraction and reuse when specifying workflows, and therefore workflow views may already exist.

However, unless a view is carefully designed, it may not preserve the dataflow between modules in a workflow, and thus can be misleading and lead to incorrect provenance analysis. For example, consider the view defined in Fig. 3(a), and suppose a user would like to determine the provenance of output O2 of module M20 in the projected view in Fig. 3(b). Based on the abstracted provenance graph, she would believe that inputs I2, I1 and I6 are all involved. However, there is no path between I6 and O2 in the original workflow; only I1 and I2 are in the provenance of O2.

Ideally, a view should preserve all the data dependencies between composite tasks in the workflow, without adding or removing dependencies. We call such a view *sound* with respect to provenance. In our example, the view in Figure 3(b) indicates a data dependency path between I6 and O2, which does not exist in the original workflow in Figure 3(a), and thus unsound. Although it would seem natural to design views which are sound, our survey of workflow designs in a well-curated workflow repository [1] revealed several unsound views. The goal of the WOLVES system [13] is therefore to diagnose and correct unsound views.

We prove that a view is sound if every composite task in the view is sound. Two alternatives can be pursued for correcting an unsound task: Splitting it into multiple smaller tasks, or merging it with other tasks. Note that splitting composite tasks refines the initial view to a lower level and provides more provenance information. In contrast, merging tasks loses information, as tasks that are important to the user may be invisible after the merge. Therefore, in WOLVES we focus on techniques that resolve an unsound view by splitting unsound composite tasks rather than merging them. For example, we could split M20 into three composite tasks: {M1}, {M2}, {M6}, {M3, M4, M5}.

Our goal is to correct an unsound view by splitting its unsound composite tasks to a minimal number of tasks, each of which is sound. However, this problem is NP-hard by reduction from the independent set problem. To efficiently tackle this problem, we propose two optimality criteria: weak local optimality and strong local optimality. A weak local optimal solution is one in which no two tasks in the resulting view can be merged into a sound task, and strong local optimal solution is one in which no set of two or more tasks in the view can be merged. Weak local optimality can be achieved with an $O(n^2)$ algorithm, and strong local optimality with an $O(n^3)$ algorithm, where n is the number of tasks in the workflow. The proposed algorithms are much more efficient than the algorithm which produces an optimal solution. The strongly local optimal algorithm often has comparable processing efficiency to the weakly local optimal algorithm, and produces views that are comparable to the optimal one.

IV. Searching Workflows through Views

An increasing number of workflow specifications and their views are being stored, either as part of a local workflow system or collected to form a community repository (e.g. myexperiment.org [10]). It is therefore important for workflow designers to be able to *search* these repositories and then reuse, include or revise the retrieved workflows to simplify the design of a new workflow.

Techniques for finding workflows of interest are currently limited to keyword searches based on the name of the workflow or tags explicitly associated with the workflow, and the result is a set of workflows shown at an arbitrary level of detail. However, by using a notion of *hierarchical* user views, this rudimentary way of searching for workflows of interest can be significantly improved. In a hierarchical user view, composite modules may themselves contain composite modules, and names can be associated with each atomic or composite module.

For example, suppose that a user would like to make a sauteed dish which uses chicken breast and coconut, and needs a recipe. She would then issue a keyword query Q, "*chicken breast, coconut milk, saute*" on a repository of recipes (workflows) to search for relevant recipes.

Now suppose that Fig. IV is one of the relevant recipes in the repository, where all query keywords have matches. Obviously, returning the entire workflow hierarchy as a query result, i.e. the one in which all the composite modules are exposed, is difficult to understand since too much irrelevant information is exposed to the user. We therefore need to find ways of exposing relevant information in our query results.

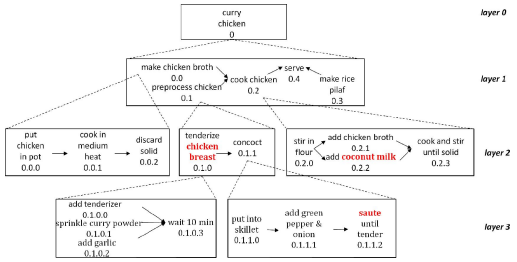

Fig. 4. Recipe Workflow Hierarchy

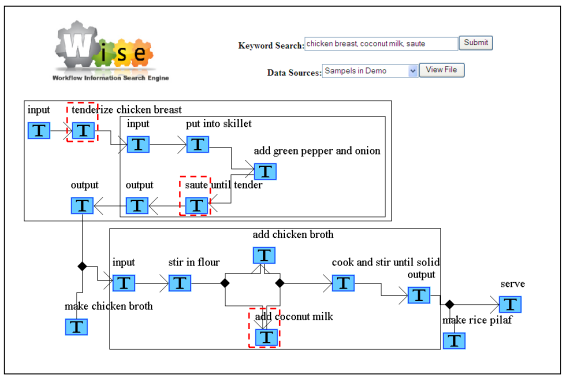

Fig. 5. Query Result for Q:{chicken breast, coconut milk, saute}

The immediate question is how to define search results when users issue keyword queries on a repository of workflow hierarchies. Recall that much research has been done on keyword search on graph-structured data (e.g., relational databases) and tree-structured data (e.g., XML), where a result is defined as a smallest data tree that contains the query keywords. Unfortunately, this definition of a query result is not appropriate for workflow search as the result is not guaranteed to capture the dependencies and dataflow among tasks that contain keyword matches.

For our example, a "good" query result is shown in Fig. IV, which is a *query-driven view* that visualizes keyword matches shown in dashed rectangles together with their dataflow. For example, after `saute (the chicken) until tender`, we `stir (it) in flour` and then `add coconut milk`. Note that the dataflow paths among these tasks are not explicitly shown in the workflow hierarchy in Fig. IV, but are derived. Also, expansion edges that represent irrelevant views are avoided.

Supporting keyword search on workflow hierarchies poses new challenges beyond keyword search on relational and XML data. First we need to define meaningful query results. We propose that a query result should be a *minimal query-driven view* of the workflow hierarchy that contains all keyword matches, which is a graph containing all matches and dataflow edges among them. Second, we must design efficient techniques to generate such query results.

To address these challenges, we have developed WISE [12], a Workflow Information Search Engine (available at http://wise.asu.edu/). WISE allows users to search a repository of workflow hierarchies using simple keywords, and returns concise and informative query results. The query results can be efficiently and dynamically synthesized by exploiting indexes and labeling schemes. To the best of our knowledge, this is the first work that supports keyword search on repositories of workflow hierarchies and returns query results capturing the dataflows among tasks matching keywords.

V. Conclusion

User views, in which composite modules are used to hide portions of a workflow specification or execution, are a useful abstraction for simplifying information. We have shown how they can be used to create individualized views of provenance information by having users indicate which modules are relevant, and how existing user views can be corrected to accurately capture provenance information. We have also discussed how they can be used to simplify the result of a keyword search over a repository of workflow specifications.

We are currently pursuing several other interesting applications of user views, including provenance query languages, and secure views of workflows and their executions.

References

[1] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock. Kepler: An extensible system for design and execution of scientific workflows. In *SSDBM*, pages 423–424, 2004.

[2] O. Biton, S. C. Boulakia, and S. B. Davidson. Zoom*userviews: Querying relevant provenance in workflow systems. In *VLDB*, pages 1366–1369, 2007.

[3] O. Biton, S. C. Boulakia, S. B. Davidson, and C. S. Hara. Querying and managing provenance through user views in scientific workflows. In *ICDE*, pages 1072–1081. IEEE, 2008.

[4] O. Biton, S. B. Davidson, S. Khanna, and S. Roy. Optimizing user views for workflows. In *ICDT '09: Proceedings of the 12th International Conference on Database Theory*, pages 310–323, 2009.

[5] S. Bowers and B. Ludäscher. Actor-oriented design of scientific workflows. In *Int. Conf. on Concept. Modeling*, pages 369–384, 2005.

[6] A. Chapman, H. V. Jagadish, and P. Ramanan. Efficient provenance storage. In *SIGMOD Conference*, pages 993–1006, 2008.

[7] S. B. Davidson, S. C. Boulakia, A. Eyal, B. Ludäscher, T. M. McPhillips, S. Bowers, M. K. Anand, and J. Freire. Provenance in scientific workflow systems. *IEEE Data Eng. Bull.*, 30(4):44–50, 2007.

[8] I. T. Foster, J.-S. Vöckler, M. Wilde, and Y. Zhao. Chimera: Avirtual data system for representing, querying, and automating data derivation. In *SSDBM*, pages 37–46, 2002.

[9] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *IPAW*, volume 4145 of *LNCS*, pages 10–18. Springer, 2006.

[10] myExperiment. http://www.myexperiment.org/workflows.

[11] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. Greenwood, K. Carver, M. G. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(1):3045–3054, 2003.

[12] Q. Shao, P. Sun, and Y. Chen. Wise: a workflow information search engine. In *ICDE*, 2009.

[13] P. Sun, Z. Liu, S. B. Davidson, S. N., and Y. Chen. WOLVES: Achieving Correct Provenance Analysis by Detecting and Resolving Unsound Workflow Views. In *PVLDB*, 2009.

[14] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series parallel digraphs. *SIAM J. Comput.*, 11(2):298–313, 1982.