Querying and Managing Provenance through User Views in Scientific Workflows

Olivier Biton [#1], Sarah Cohen-Boulakia [#2], Susan B. Davidson [#3], Carmem S. Hara [*4]

[#] *University of Pennsylvania, Philadelphia, USA*
{[1]biton, [2]sarahcb, [3]susan}@cis.upenn.edu

[*] *Universidade Federal do Paraná, Brazil*
[4]carmem@inf.ufpr.br

***Abstract*— Workflow systems have become increasingly popular for managing experiments where many bioinformatics tasks are chained together. Due to the large amount of data generated by these experiments and the need for reproducible results, provenance has become of paramount importance. Workflow systems are therefore starting to provide support for querying provenance. However, the amount of provenance information may be overwhelming, so there is a need for abstraction mechanisms to help users focus on the most relevant information. The technique we pursue is that of "user views." Since bioinformatics tasks may themselves be complex sub-workflows, a user view determines what level of sub-workflow the user can see, and thus what data and tasks are visible in provenance queries.**

In this paper, we formalize the notion of user views, demonstrate how they can be used in provenance queries, and give an algorithm for generating a user view based on which tasks are relevant for the user. We then describe our prototype and give performance results. Although presented in the context of scientific workflows, the technique applies to other data-oriented workflows.

I. Introduction

Workflow management systems (e.g. [1], [2], [3]) have become increasingly popular as a way of specifying and implementing large-scale in-silico experiments. In such systems, a workflow can be graphically designed by chaining together bioinformatics tasks (e.g. aligning sequences, building a phylogenetic tree). Such workflows typically use data coming from various databases and involve several tools, all requiring inputs in a particular format. Scientific workflows may therefore contain a large number of formatting tasks which are unimportant in terms of the scientific goal of the workflow, and which make the workflow appear extremely complex.

As an example, consider the workflow specification (a.k.a workflow definition or schema) in Figure 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

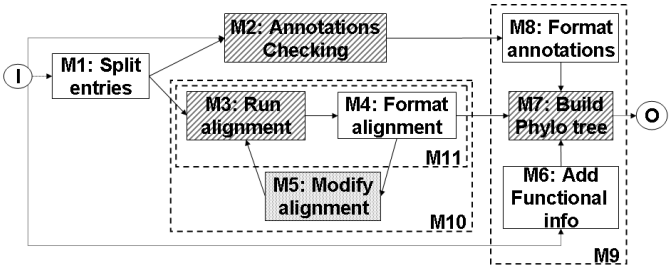

Fig. 1. Phylogenomic workflow

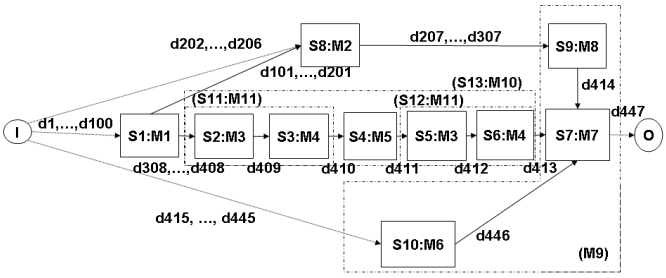

Fig. 2. Phylogenomic workflow run

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.

Workflows may be executed several times a month, resulting in vast amounts of intermediate and final data objects. Figure 2 shows an execution of the workflow in Figure 1. Each box (e.g. S1:M1) is called a *step* and represents the execution of a module; in the figure, it is labeled both with a unique id

for the step (e.g. S1) as well as the module of which it is an execution (e.g. M1). Edges in this execution are labeled with data ids (e.g. d1, d202, d447) representing the actual data that is used/created during the execution. In the workflow execution (a.k.a workflow run) of Figure 2, one hundred sequences are taken as initial input (d1 to d100), minor modifications are done on the annotations (d202 to d206), and thirty additional annotations are used (d415 to d445).

In order to understand and reproduce the results of an experiment, scientists must be able to determine what sequence of steps and input data were used to produce data objects, i.e. ask *provenance* queries, such as: *What are all the data objects/ sequence of steps which have been used to produce this tree?*

However, since a workflow execution may contain many steps and data objects, the amount of provenance information can be overwhelming. For example, the provenance of the final data object d447 in Figure 2 would include every data object (d1,...,d447) and every step (S1,...,S10). There is therefore a need for abstraction mechanisms to present the most *relevant* provenance information to the user.

A technique that is used in systems such as Kepler [1] and Taverna [3] is that of *composite* modules, in which a module is itself a smaller workflow. Composite modules are an important mechanism for abstraction, privacy, and reuse [4] between workflows. The idea in this paper is to use composite modules as an abstraction mechanism, driven by user-input on what is relevant for provenance.

Intuitively, we want to allow users to group modules together to get a workflow in which composite modules represent some relevant task. For example, suppose user Joe believes "Annotations checking" (M2), "Run alignment" (M3) and "Build Phylo tree" (M7) (shaded boxes in Figure 1) to be relevant. Then he might group M6, M7 and M8 together in composite module M9 (shown as a dotted box), which then takes on the meaning of relevant module M7, e.g. building a phylogenetic tree. Similarly, he might group M3, M4 and M5 together in composite module M10, which takes on the meaning "Run alignment".

Note that users may differ in their interests: While Joe is not interested in the alignment modification step (M5), another user, Mary, may be (M5 is lightly shaded in Figure 1). Mary would therefore not include M5 in her composite module representing "Run alignment"; M11 includes only M3 and M4, leaving M5 visible. She may, however, agree on composite module M9. Joe and Mary will therefore have different *user views* defining the level of granularity at which they wish to view the workflow. Using the composite modules in the user view, an *induced* workflow can be created (see Figure 3).

Since M1 is not relevant to Joe, he may now wish to group it with M2, M9 or M10. However, these groupings would dramatically modify the perceived dataflow between relevant modules. For example, by grouping M1 with M2 in a composite module M12, there would exist an edge from M12 to M10 in the view, due to the edge from M1 to M3 in the workflow specification. That is, it would appear that "Annotation checking" (M2) must be performed before "Run alignment" (M3), when in fact there is no precedence or dataflow between those modules. We must therefore restrict what groupings can occur so as to preserve the precedence between relevant modules and hence the perceived data provenance.

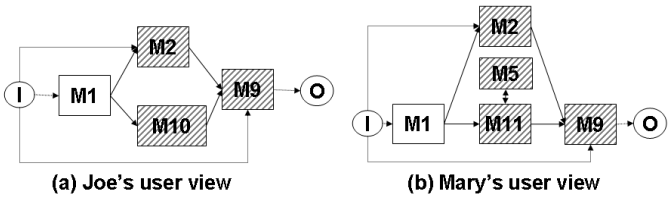

Fig. 3. Induced phylogenomic workflows

In this paper, we have two goals. First, we want to help users construct relevant user views. For example, Joe would be presented with M1,.., M8 and indicate that he finds M2, M3 and M7 to be relevant. Based on this input, the user view in Figure 3(a) would be created.

Second, we want to design a system in which the answer to a provenance query depends on the level at which the user can see the workflow. For example, based on the execution in Figure 2, the answer to a query by Mary on what data objects were used to produce d413 would include the data passed between executions of M11 and M5, d410 and d411. However, this data would not be visible to Joe since it is internal to the execution of M10. Thus the answer to a provenance query depends on the user view. Our approach should also be able to switch between user views. While several workflow systems are able to answer provenance queries [5], none take user views into account.

Contributions. In this paper we propose a model for querying and reasoning about provenance through user views (Section 2). We then define properties of a "good" user view, and present an algorithm which takes as input a workflow specification and a set of relevant modules, and constructs a good user view (Section 3).

Based on this model, we have built a *provenance reasoning system* which assists in the construction of good user views, stores provenance in an Oracle warehouse, and provides a user interface for querying and visualizing provenance with respect to a user view (Section 4).

To evaluate our provenance reasoning system, we have created an extensive suite of simulated scientific workflow specifications based on patterns observed in 30 actual workflows collected from scientists. Measurements include the cost of querying the warehouse, while varying the kind of workflow, run, and user view (Section 5).

It should be noted that although our approach is illustrated using scientific workflows, it is *generic* in the sense that it can be used by any workflow system which provides the required information.

II. Workflow Model and Provenance

Workflow specification. A *workflow specification* defines the order in which modules can be executed and indicates

dataflow. More formally, it is a directed graph, $G_w(N, E)$, in which nodes are uniquely labeled modules. Two special nodes, *input* (I) and *output* (O), are source and sink nodes, respectively, and indicate the beginning and end of the workflow. Every node of G_w must be on some path from *input* to *output*.

User view. A *user view* U of a workflow specification is a partition of its nodes N (excluding *input* and *output*), that is a set $\{M_1, ..., M_n\}$ such that $\emptyset \neq M_i \subseteq N$, M_i and M_j are disjoint for $i \neq j$, and $M_1 \cup M_2 \cup ... \cup M_n = N$. Each M_i is called a *composite* module. The *size* of U, $|U|$, is the number of composite modules it contains. For example, the size of Joe's user view is 4 while that of Mary is 5.

A user view $U = \{M_1, ..., M_n\}$ of a workflow specification G_w *induces* a "higher level" workflow specification, $U(G_w)$, in which there is a node for each M_i (labeled with a new composite module name), *input* and *output* nodes, and an edge $M_i \longrightarrow M_j$ whenever there is an edge in G_w between a module in M_i and a module in M_j (similarly for edges $input \longrightarrow M_i$ and $M_i \longrightarrow output$). The induced specification for Joe's and Mary's user views are shown in Figures 3 (a) and (b), respectively.

Workflow run. An execution of a workflow specification is called a *workflow run*. It generates a partial order of *steps*, each of which has a set of *input* and *output* data objects. More formally, it is a directed acyclic graph, G_r, in which nodes are labeled with unique step-ids as well as the modules of which they are executions. Module labels are not necessarily unique due to cycles in the workflow specification, that is, loops in the specification are unrolled in the execution. For example, in Figure 2 there are two executions of M3, S2 and S5, since the loop between M3 and M5 was executed twice. Edges are labeled with a unique edge label indicating the ids of the data output by the source step and input to the target step. Nodes *input* and *output* indicate the beginning and end of the execution, respectively; every node must be on some path from *input* to *output*.

Provenance. Each data object in the workflow dataspace is produced either as a result of a step of a workflow run, or is input by the user. We call the *provenance* of a data object the sequence of modules and input data objects on which it depends [5], [6]. If the data is a parameter or was input to the workflow execution by a user, its provenance is whatever metadata information is recorded, e.g. who input the data and the time at which the input occurred. Following other work (e.g. [7], [1]), we assume data is never overwritten or updated in place. Each data object therefore has a unique identifier and is produced by at most one step.

We assume that each workflow run generates a log of events, which tells what module a step is an instance of, what data objects and parameters were input to that step, and what data objects were output from that step. For example, the log could include the start time of each step, the module of which it was an instance, and read (write) events that indicate which step read (wrote) which data objects at what time.

Using this log information, we can determine the *immediate provenance* for a data object as the step which produced it and the input set of data objects. The *deep provenance* for a data object is recursively defined as all the steps and input set of data objects that were transitively used to produce it.

For example, the immediate provenance of the data object d413 in the workflow run of Figure 2 is the step with id S6, which is an instance of the module M4, and its input set of data objects {d412}. The deep provenance of d413 includes all the steps and their inputs that transitively produced it, and would include (among other things) the step with id S2, which is an instance of the module M3, and its input set of data objects {d308,...,d408}.

Composite executions. The execution of consecutive steps within the same composite module causes a virtual execution of the composite step, shown in Figure 2 by dotted boxes. These virtual executions can be constructed from the log information as well as containment information between modules and composite modules. For example, we would construct an execution of M10 with the id S13 in Figure 2, which takes as input the set of data objects {d308,...,d408} and produces as output {d413}. Similarly, we would construct two executions of M11, the first of which (id S11) takes as input {d308,...,d408} and produces as output {d410}, and the second of which (id S12) takes as input {d411} and produces {d413} (see [8] for details).

Since user views are defined in terms of composite modules, they restrict what provenance information can be seen in an execution by hiding internal steps as well as the data passed between internal steps. For example, the immediate provenance of d413 seen by Joe would be S13 and its input, {d308,...,d408}, since composite module M10 is in his user view, whereas that seen by Mary would be S12 and its input, {d411}, since M11 is in her user view. The deep provenance of d413 as seen by Mary would include the first execution of M11, S11, and its input {d308,...,d408}. However, Joe would not see the data d411, nor would he be aware of the looping inside of S13, i.e. the two executions of M3.

III. Constructing user views

We now turn to the question of constructing user views using a bottom-up approach. Such an algorithm will take as input a workflow specification and a set of relevant modules, and produce as output a user view. But what are the properties of a "good" user view?

From discussions with our scientist collaborators, a user view is (intuitively) good if (i) the user sees a composite module for each relevant module. The composite module takes on the meaning of the relevant module it contains, hence in the induced workflow the paths (dataflow) between relevant modules (as represented by their composite modules) should be preserved. No path should be (ii) added or (iii) removed from the original workflow. However, the need to preserve paths between relevant modules may mean that it is impossible to include every non-relevant module in the specification in some composite module which represents a relevant module. That is, we may need to create one (or more) composite

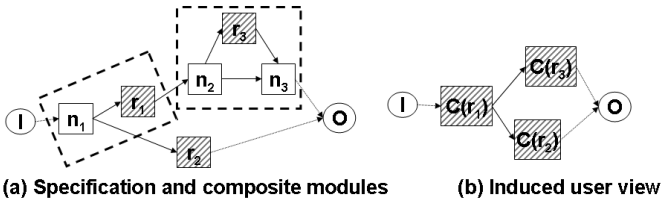

Fig. 4. Counter-example: Properties 2 & 3

modules that do not contain a relevant module. Since such composite modules have no meaning in terms of a relevant module, (iv) there should be as few as possible.

Observations (i) to (iv) are respectively formalized by Properties 1 to 3 and a minimality condition, as follows:

Property 1: Given a workflow specification G_w and a set $R \subseteq N$ of relevant modules, a user view U is *well-formed* iff every composite module in U contains at most one element of R.

Given a well-formed user view U for (G_w, R) and $n \in N$, we use $C(n)$ to denote the composite module in U which contains n. For simplicity, we extend the notation to include $C(input) = input$ and $C(output) = output$. Furthermore, we use the term `nr-path` to denote a path in G_w (or in view $U(G_w)$) which contains no relevant intermediate module $r \in R$ (or relevant composite module $C(r)$). As an example, in the workflow specification of Figure 1, there exists an `nr-path` from $input$ to M2, but not from $input$ to M7, since all paths connecting these two modules contain an intermediate node in R (M2, M3).

Property 2: A user view U *preserves dataflow* iff every edge in G_w that induces an edge on an `nr-path` from $C(r)$ to $C(r')$ in $U(G_w)$ lies on an `nr-path` from r to r' in G_w. Here, r, r' are nodes in $R \cup \{input, output\}$.

Property 3: A user view U is *complete w.r.t dataflow* iff for every edge e on an `nr-path` from r to r' in G_w that induces an edge e' in $U(G_w)$, e' lies on an `nr-path` from $C(r)$ to $C(r')$. Here, r, r' are nodes in $R \cup \{input, output\}$.

In other words, every `nr-path` from $C(r)$ to $C(r')$ in $U(G_w)$ must be the residue of an `nr-path` from r to r' in G_w, and each `nr-path` in G_w must have a residue in $U(G_w)$.

As an example, consider the workflow and user view U shown in Figure 4. While U is well-formed, it does not preserve dataflow: The edge (n_1, r_2) induces the edge (`nr-path`) $(C(r_1), C(r_2))$ in $U(G_w)$, but there is no path from r_1 to r_2 in G_w. This gives the impression that r_1 produces data necessary for r_2. U is also not complete w.r.t. dataflow as the edge (r_1, n_2) is on an `nr-path` from r_1 to O while the induced edge $(C(r_1), C(r_3))$ in $U(G_w)$ is not. This gives the impression that the output of r_1 cannot flow directly to O (without going through r_3).

Algorithm RelevUserViewBuilder, given in Figure 5, takes as input a workflow specification G_w and a set of relevant modules R, and produces a user view $U = \{M_1, ..., M_n\}$. Such U not only preserves Properties 1-3, but it is *minimal*:

Algorithm RelevUserViewBuilder

Input: workflow spec. $G_w(N, E)$, relevant modules R
Output: user view U

/* **Step 1: Create relevant composite modules** */
1. $U = \emptyset$;
2. initialize $n \in (N - R)$ as unmarked;
3. **for all** $r \in R$ **do**
4. $in(r)$= $\{n \in (N - R) \mid \texttt{rSucc}(n) = \{r\}\}$;
5. mark each n in $in(r)$;
6. **for all** $r \in R$ **do**
7. $out(r)$ = $\{n \in (N - R) \mid \texttt{rPred}(n) = \{r\}$ and n is unmarked$\}$;
8. mark each n in $out(r)$;
9. **for all** $r \in R$ **do**
10. $M = in(r) \cup out(r) \cup \{r\}$; insert M in U;

/* **Step 2: Create non-relevant composite modules** */
11. $NRC = \emptyset$;
12. **for all** unmarked $n \in (N - R)$ **do**
13. **if** $\exists M \in NRC$ s.t. $\texttt{rPredM}(M) = \texttt{rPred}(n)$ and $\texttt{rSuccM}(M) = \texttt{rSucc}(n)$ **then**
14. insert n to M;
15. **else**
16. $M = \{n\}$; insert M to NRC;

/* **Step 3: Make the user view minimal**/
17. **repeat until** no further change to NRC
18. **for all** $M_1 \in NRC$
19. **for all** $M_2 \in NRC$
20. $M = M_1 \cup M_2$;
21. $V^- = \{n \in M \mid n$ has an incoming edge from some $n' \notin M\}$;
22. $V^+ = \{n \in M \mid n$ has an outgoing edge to some $n' \notin M\}$;
23. **if** $\forall n \in V^+$: $\texttt{rPred}(n) = \texttt{rPredM}(M)$ and $\forall n \in V^-$: $\texttt{rSucc}(n) = \texttt{rSuccM}(M)$ **then**
24. remove M_1 and M_2 from NRC;
25. insert M to NRC;
26. **return** $(U \cup NRC)$;

Fig. 5. RelevUserViewBuilder

No two M_i, M_j can be removed from U and replaced by $M_i \cup M_j$ to yield a solution that preserves Properties 1-3. Using this algorithm, both Joe's and Mary's user views would be constructed automatically.

The functions used in the algorithm are defined as follows:
For $n \in N$ and $M \in U$:
(i) $\texttt{rPred}(n)$= $\{r \in (R \cup \{input\}) \mid$ there is an `nr-path` from r to n $\}$
(ii) $\texttt{rSucc}(n)$= $\{r \in (R \cup \{output\}) \mid$ there is an `nr-path` from n to r $\}$
(iii) $\texttt{rPredM}(M)$= $\bigcup_{n \in M} \texttt{rPred}(n)$
(iv) $\texttt{rSuccM}(M)$= $\bigcup_{n \in M} \texttt{rSucc}(n)$

The algorithm has three steps. In the first step (Lines 1 to 10), for each relevant module $r \in R$, a composite module $C(r)$ is created, which includes non-relevant modules that are connected to r by an `nr-path` and for which r is their only relevant successor ($in(r)$) or predecessor ($out(r)$). Since a module n can be both in $in(r)$ and $out(r')$, $r \neq r'$, the algorithm "marks" n when it is included in $in(r)$ (Line 5), and does not consider it when computing $out(r')$ (Line 7). As an

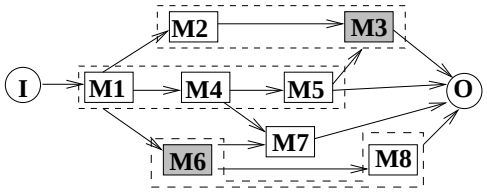

Fig. 6. Example of workflow spec.

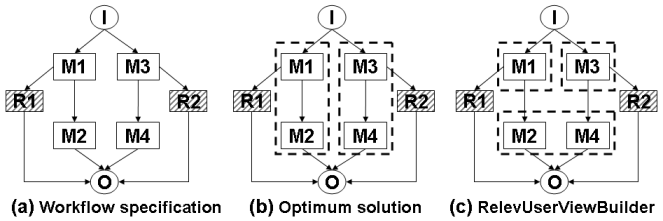

Fig. 7. Example of non minimum solution

example, consider the workflow specification of Figure 6. The algorithm constructs $\{M2, M3\}$, and $\{M6, M8\}$ as relevant composite modules, since $in(M3) = \{M2\}$, and $out(M6) = \{M8\}$. Observe that $M1$ is not in $in(M3)$ because from $M1$ there are `nr-path`s to $M3, M6$, and $output$. Similarly, $M7$ is not an element of $out(M6)$ because it can be reached from both $input$ and $M6$ through an `nr-path` .

Step 2 of the algorithm is responsible for grouping non-relevant modules that remained unmarked after the first step. More specifically, a non-relevant composite module is created for modules that have exactly the same set of relevant modules as predecessors ($\texttt{rPred}(n)$) and successors ($\texttt{rSucc}(n)$), considering only those connected by `nr-path`s (Lines 11 to 16). Consider again the workflow specification of Figure 6. This step of the algorithm constructs $\{M4, M5\}$, $\{M1\}$, and $\{M7\}$, since $\texttt{rPred}(M4) = \texttt{rPred}(M5) = \{input\}$, $\texttt{rSucc}(M4) = \texttt{rSucc}(M5) = \{M3, output\}$, $\texttt{rPred}(M1) = \{input\}$, $\texttt{rSucc}(M1) = \{M3, M6, output\}$, $\texttt{rPred}(M7) = \{input, M6\}$, and $\texttt{rSucc}(M7) = \{output\}$.

The last step of the algorithm checks whether non-relevant composite modules can be merged. Two modules $M1$ and $M2$ can be merged into a new module M iff this does not create `nr-path`s in the view that do not exist in the original specification. This condition is checked by comparing the relevant successors of the entry points (V^-) with the successors of M, and the relevant predecessors of the exit points (V^+) with the relevant predecessors of M (Line 23). If these two sets are equal then the property is preserved. Consider again our running example of Figure 6. The algorithm merges $\{M1\}$ with $\{M4, M5\}$ to create M, since $V^-(M) = \{M1\}$, $V^+(M) = \{M1, M4, M5\}$, $\texttt{rPredM}(M) = \texttt{rPred}(M1) = \texttt{rPred}(M4) = \texttt{rPred}(M5) = \{input\}$, and $\texttt{rSuccM}(M) = \texttt{rSucc}(M1) = \{M3, M6, output\}$. However, observe that M cannot be merged with $M7$. This is because $\texttt{rPred}(M7)$ includes $M6$ from which $M1$ (an element of V^+) cannot be reached. Thus, this merging would induce an `nr-path` in the view from $\{M6, M8\}$ to $\{M2, M3\}$ through M; however, in the original specification there is no `nr-path` from $M6$ to $M3$.

Theorem 1: RelevUserViewBuilder preserves Properties 1-3 and produces a minimal user view.
(The proof can be found in [8].)

RelevUserViewBuilder is clearly a polynomial-time algorithm ($O(|N|^2 + |E|)$), and is very fast in practice (see Section 5.1). However, RelevUserViewBuilder does not guarantee a *minimum* solution, i.e. one with the smallest size. For example, the minimum solution shown in Figure 7(b) has size 4 and does not combine modules with same $rPred/rSucc$; in contrast, the solution produced by RelevUserViewBuilder (Figure 7(c)) has size 5. It is an open question as to whether there exists a polynomial time algorithm which preserves Properties 1-3 and guarantees a minimum solution.

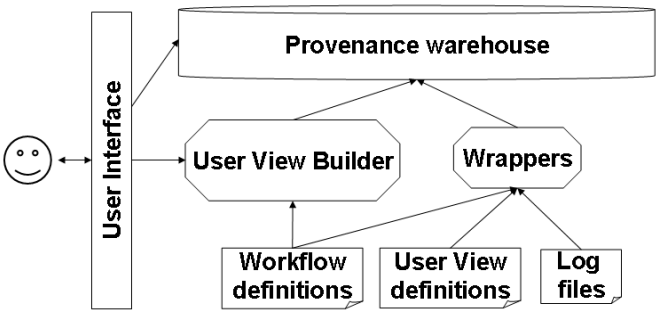

Fig. 8. Architecture

These properties do not state that composite modules must be connected. However, Properties 1-3 guarantee that a relevant composite module will always be a connected partition. This is not true for non-relevant composite modules, where we may wish to hide parallel executions. We have also shown that Properties 1-3 do not introduce loops in the induced workflow other than those that were present in the original specification [8].

IV. ZOOM PROTOTYPE

The goal of our prototype, ZOOM, is to provide users with an interface to query the provenance information provided by a workflow system, as well as to help them construct an appropriate user view.[1]

The architecture of ZOOM is presented in Figure 8. The system designer provides information about workflow specifications and possibly user view definitions, which are converted to tables and stored in the provenance warehouse. Following (or during) a workflow execution, information about the input and output of steps is also extracted from the workflow log and stored in the warehouse. Users interact with the system by building a user view or posing a provenance query. Provenance information is displayed graphically to aid in understanding the results.

UserViewBuilder. UserViewBuilder takes as input the workflow definition, loads it in a graphical environment,

[1]The prototype is available at `http://zoomuserviews.db.cis.upenn.edu/cgi-bin/pmwiki.php`.

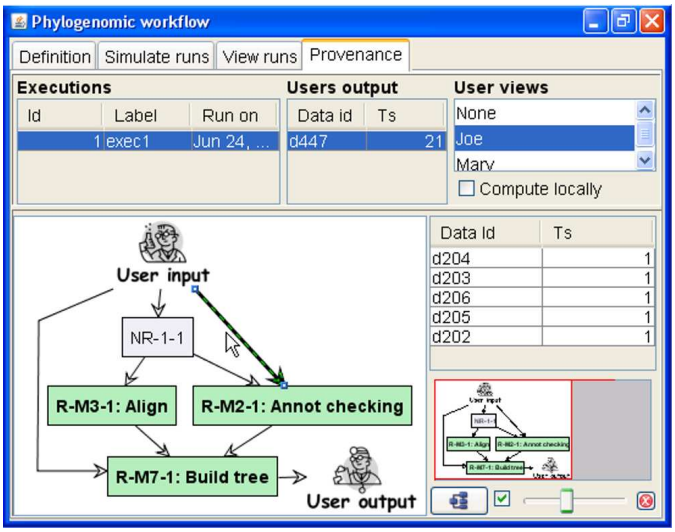

Fig. 9. Graph of provenance information

and allows users to specify which modules are relevant. It produces as output a user view. In our prototype, algorithm RelevUserViewBuilder runs interactively, allowing the user to visualize the new user view each time he flags or unflags a module as relevant.

Querying Provenance. In the prototype, runs are displayed graphically. By selecting a run and clicking on an edge between two steps, the user can see the data set passed between them. To query provenance, the user first selects the data id of interest, and then the requested provenance information is calculated with respect to the user view, and displayed as a graph. When the workflow graphs are large, the user can navigate over the portion of the graph he is interested in. As the user's needs evolve, he may modify (add or remove) the set of modules he considers to be relevant. The provenance graph is then automatically modified for the new user view. For example, the answer to the deep provenance of the final output (data id d447) using Joe's view is shown in Figure 9.

The database used for our prototype is Oracle 10.g. The user interface and wrappers are developed using Java with JDBC. Deep provenance queries are implemented using Oracle's recursive query capabilities (CONNECT BY), extended with stored procedures. Details of the relational schema and the implementation can be found in [8] and in the demo paper [9].

Ongoing work on our prototype includes providing users with forms to express various (canned) provenance queries such as *Return the data objects which have a given data object in their data provenance*.

V. Evaluation

As observed in [10], evaluating any approach to provenance is difficult. Evaluating an approach for querying provenance in scientific workflow systems is particularly challenging since it requires realistic workflows and runs on which to base the experiments. Since scientific workflows reflect expertise and describe precisely how an experiment has been done, they are typically shared in detail with only a small group of collaborators. Very few are made available publicly, although they may be outlined in scientific publications in an experimental methods section.

TABLE I
Classes of Workflows

Class	Pattern (Frequency)	Number of workflows	Avg Size
Class 1	Real workflows	30	12
Class 2 (Linear)	Sequence: 80% Loop: 10% Parallel Process: 10%	10	20
Class 3 (Parallel)	Parallel Process: 20% Parallel Input: 10% Synchronization: 20% Sequence: 50%	10	20
Class 4 (Loop)	Loop: 50% Sequence: 50%	10	20

To evaluate the feasibility of our approach, we therefore collected scientific workflows published in the literature as well as examples of use of several scientific workflow systems, such as myGrid [3] and Kepler [1]. We also obtained detailed information about workflows that our collaborators in biology were running. In this way, a total of thirty scientific workflows were collected together with their runs. From this data, we extracted *patterns* of workflows (e.g., sequence, loop) [11], and inferred statistics on their usage (e.g. the sequence pattern is used four times more than the reflexive loop). Statistics on runs, such as the average number of loop iterations, were also inferred. We then generated simulated workflows by combining patterns according to usage statistics, and produced runs of these simulated workflows that reflect the statistics inferred from real workflow runs. In this way, we were able to generate realistic synthetic workflows and runs. We were also able to gather information about characteristics of user views (e.g. their size) from the collected workflow definitions.

In our approach, provenance reasoning is based on the log files provided by workflow systems (e.g., [12], [13]). Our experiments therefore do not measure the cost of provenance tracking which produces the log files; rather, while varying the class of workflow, runs, and user view, we measure the cost of constructing user views and querying the provenance warehouse, as well as determine how evolvable and interactive our approach is.

A. Experimental setup

The experiments were performed on a Dell PowerEdge 1950 with 4GB RAM and 250GB disk space running Linux RedHat Enterprise AS4.4. Oracle 10.g was used as the database system and was allocated 1GB of memory.

Classes of workflows. Each class of workflow exhibits particular pattern frequencies (sequence, loop etc.). Table I describes these classes: Class 1 is the set of real workflows collected, while Classes 2-4 are synthetic workflows. The settings were based on the characteristics of real workflows collected

TABLE II
CLASSES OF RUNS

Kind	User input (range)	Data prod. by step (range)	Loop-iteration (range)	Size (Nodes -Edges)
Small	1-10	1-10	1-3	105-523
Medium	1-50	1-50	1-10	306-6406
Large	1-100	1-100	1-50	1153-41633

but are considerably more complex (e.g., no workflow that we collected had as many loops and involved as many nodes as those in Class 4).

Classes of runs. We also identified parameters that determine the complexity of a workflow run, and used them to generate different classes of runs, as shown in Table II. These parameters were: the size of the run (small, medium, and large); the amount of data given as input by the user (user input), the amount of data generated by a step (data produced); and the number of loop iterations. The "size" parameter of Table II indicates the maximum number of nodes and edges of the workflow runs.

User views. We used algorithm RelevUserViewBuilder to create relevant user views. The choice of relevant modules given as input to the algorithm was done both by hand (using our experience from case studies and advice given by biologists) as well as randomly. In our experiments, we call the user views generated in the former case as UBio and those generated in the latter case as UV. In the latter case, we randomly chose a given percentage of modules in a workflow to be relevant. Relevant modules were selected randomly 10 times for each percentage, and the percentage varied from 0 to 100 by steps of 10.

B. Experiments and results

The goal of our experiments is to evaluate: (i) the performance of the RelevUserViewBuilder algorithm, (ii) the benefit of user views when querying provenance, and (iii) the interactive capability of our system. Benefit is measured in terms of the size of the answer to a provenance query, which should be as concise as possible while providing all provenance information that is relevant to the user.

Experiments on RelevUserViewBuilder. In the following two experiments, we have evaluated RelevUserViewBuilder by running the algorithm on 1000, increasingly large, randomized workflow specifications (100-2000 nodes).

Scalability. In our first experiment, we evaluated the scalability of RelevUserViewBuilder. Each execution of the algorithm took less than 80ms.

Optimality. In the second experiment, we evaluated the "optimality" of RelevUserViewBuilder by increasing the percentage of relevant modules chosen and measuring the number of relevant composite modules created. Recall that a lower bound on the size of a user view is the number of relevant modules, and that such a view is "optimal" in the sense that it only contains relevant composite modules. Our results showed

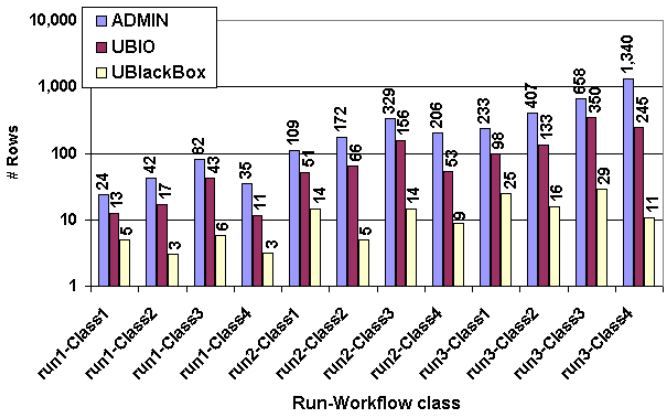

Fig. 10. Size of query result

that adding one relevant class in a workflow creates only one new composite class, meaning that RelevUserViewBuilder algorithm does not frequently construct non-relevant composite modules.

Experiments on querying provenance through user views. Since querying provenance is affected more by the size of the run than by the size of the specification, we used specifications containing about 20 nodes, which is slightly larger than the 12 node average of the real workflows collected, and with more loops than found in practice in order to complicate the runs (most collected workflows were linear). By iterating over the loops many times we were able to generate very large runs (see Table II).

We used two metrics to measure the cost of querying provenance: The time to produce the query result and the size of the query result. Size plays a crucial role since it measures the difficulty of understanding the result provided, and therefore the usability of the system. Using 10 workflows in each of the 4 classes described in Table I, we created 30 runs of each kind in Table II (small, medium and large), generating 3,600 runs in total. This corresponds to what would happen in a large laboratory with 40 workflows, each of which is executed about twice a week.

In the next two experiments described below, we used the most expensive provenance query possible: the deep provenance of the final output of the run. We also considered three types of user views: UAdmin, in which each step class is relevant (no composite modules); UBio, constructed from relevant modules using RelevUserViewBuilder; and UBlackBox, in which the entire workflow is in one composite class.

Conciseness of query result. Figure 10 shows the size of the query result, i.e. the number of tuples returned in the deep provenance of the final output of a run, while varying the type of user view and runs (note that log scale is used for the y-axis). Each bar represents a run of a particular kind (run1, small; run2, medium; run3, large) of a workflow of a particular class (Class1 through Class4, see Table I). In small runs, an average of 24 data items are returned in UAdmin, 13 in UBio, and 5 in UBlackBox. In medium and large runs,

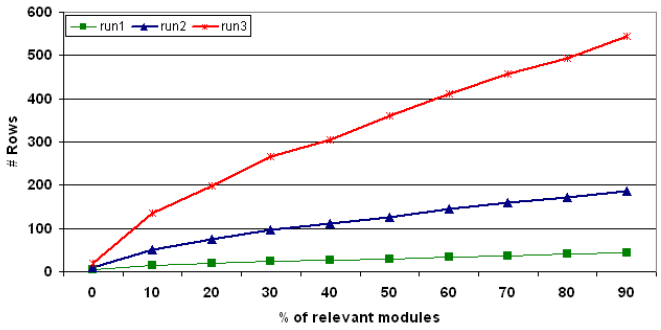

Fig. 11. Effect of view granularity on size of query result

UBio views are even better filters, yielding only 20% of the data returned in UAdmin and an average of 22 times more data than UBlackBox. Interestingly, Class4 workflows (with many loops) benefit enormously from user views since iterations of loops are frequently hidden (up to 90%).

Query response time. We tested various strategies to implement the computation of deep provenance through user views, including SQL views, stored procedures, and a variety of indexes to optimize the query (details can be found in [8]). The best results were obtained by the following strategy: first compute UAdmin and then remove information hidden within composite steps of the given user view. Using this strategy, whatever the size of the user view, the response time was dominated by the first step. Even when the result was large, the query time was less than 30 seconds. On average, answering queries for small runs took 23 milliseconds, queries for medium runs took 213 milliseconds, and queries for large runs took 1.1 seconds.

Interactive capability of ZOOM*UserViews. In the next two experiments, we tested the interactive capability of our system to respond to the need for finer provenance information. We did this by measuring the cost of increasing the percentage of relevant modules in a workflow in terms of the query result size and response time. All kinds of runs, all classes of workflows, and randomized user views were tested. The deep provenance of the final output was used as the query.

Effect of view granularity on response time. We analyzed the cost of switching user views while analyzing the provenance of a given data-item, to show how the user can interact with the system when trying to understand provenance. Recall that to compute any user view UV provenance information, we first compute UAdmin and then project out UV information. In our system, when a query is executed on a given workflow run, the UAdmin provenance information is stored in a temporary table, and does not need to be recomputed when switching the user view on the same workflow run. The results therefore showed that on average it takes 13 msec to compute the provenance for a different user view. The maximum computation time was 1 sec for an execution in run4 with 90% of relevant modules. Visualizing the results took longer: On average it took 300 msec to show the provenance graph for a new user view, and the maximum time was 2 sec.

Effect of view granularity on size of query result. Lastly, Figure 11 shows how the size of the query result (i.e. the number of rows) increases as a function of the percentage of relevant modules in all workflows for small (run1), medium (run2) and large runs (run3). Each point represents the average size of the query result over 120,000 provenance queries over runs of each of the four classes of workflows. Increasing the percentage (and therefore number) of relevant modules in a workflow increases the granularity of provenance information, thereby allowing the user to see more provenance information. Although not shown in this figure (which averages over all classes of workflows), for Class4 workflows (loops) the increase in size of the query result is more than linear. In contrast to the experiment in Figure 10, in which relevant modules were selected by hand, by randomly selecting them, loops are much more likely to be seen as the size of composite modules decreases.

VI. Related Work

Provenance and annotations have been studied extensively by the database community [7], [14], [10], [15], [16]. The aim is to determine which tuples were used to generate the answer to a query by exploiting the algebraic form of the query and/or the relational or XML-like form of the data. In contrast, transformations occurring in workflows are external processes (black boxes), and the log files typically provide only object ids. Provenance is more coarse-grained, and the structure of data cannot be reasoned about.

Workflow systems have been developed in many domains, e.g. business processes and e-commerce. Within the scientific community they are used to conduct and manage experiments (e.g. [1], [2], [17], [3]). Many of these systems record information about the processes used to derive intermediate and final data objects from raw data, and can be classified in terms of provenance along three axes: the level at which information is recorded, the ability to create composite tasks, and the ease with which provenance queries can be asked. First, the level at which information is recorded varies from a very low level, e.g. Condor [18] which is job-centric, to higher levels e.g., Taverna, which stores and annotates runs with semantic concepts [3], [13], and Kepler, in which it is possible to track data flows and dependencies among actors, tokens, and objects [1]. Survey of this work can be found in [19]. Although many workflow systems differ in how they record provenance, the majority attempt to provide the scientist with the necessary tools to assess the quality of experimental results and to improve the repeatability of such results [20]. Recent “provenance challenges” have been held to encourage system designers to learn about the capabilities and expressiveness of each others’ systems and work toward interoperable solutions [5]. Second, the ability to create composite tasks (based on the ideas of Statecharts [21]) appears in many workflow systems: Kepler [1] allows composite actors, and Taverna/myGrid [13] uses nested workflow processors/knowledge view models. Third, workflow systems differ in the extent to which they facilitate queries on the provenance information.

Some systems merely provide the user with XML or RDF files containing the workflow definition and log information associated with each run (e.g. [3], [1]). Others, like GridDB [22] or Redux [23], incorporate workflow and data modeling into the same system by using a workflow system on top of a DBMS. For more information about the various layers of a generic architecture dealing with provenance recording and querying, see [24].

Work on modeling and querying information generated by workflow systems has also appeared in the e-business domain. While WQM [25] and BP-QL [26] provide query languages for business processes, they do not address user views.

In [6], we introduced a simple model of provenance that was used in the First Provenance Challenge [5]. We were able to show that our model captured the necessary information for the challenge queries, and that this information could be extracted from the log file of the workflow systems participating in the challenge. However, we did not provide an algorithm for computing (relevant) user views.

The approach presented in the present paper is agnostic as to the workflow system it supports. It can be used in any workflow system that provides basic log-like information, whether or not composition is available, and whether the recorded information is provided as a file or is stored in a DBMS. Our approach only requires a definition of the workflow, and information about the objects consumed and produced by steps in a workflow run.

Finally, there has been work on displaying workflow provenance information (see [19]) and providing support for comparative visualization [27], neither of which provide provenance information at various levels of user views. Furthermore, the problem of constructing relevant user views cannot be simply resolved by masking the graphical display of the workflow since it will not ensure the precedence between relevant composite modules in the possibly complex graph structure of the workflow.

VII. Conclusions

This paper presents an abstraction mechanism called *user views* for querying provenance information in workflows. The technique is appropriate for any data-oriented workflow system in which log-like information is collected at run time. We present an algorithm (RelevUserViewBuilder) for constructing a user view based on input from the user on what modules they believe to be relevant. Our algorithm guarantees that the view has one composite class for each relevant class, preserves and is complete w.r.t. the dataflow between relevant modules, and is minimal. We show the interaction between provenance and user views, and argue that user views not only allow users to visualize the workflow specification at a meaningful level, but to see provenance information at an appropriate level of detail. Our approach can be used in conjunction with other composite module construction techniques used in current workflow systems by either marking relevant *composite* modules in the existing workflow specification, or by viewing each composite module as itself being a workflow and marking relevant *atomic* modules contained within it. To evaluate the benefit of provenance through user views, we performed a series of experiments and showed how our approach helps users quickly focus on meaningful answers to provenance queries.

Constructing user views is a very interesting problem, and there are several directions of future work that we are pursuing (see [8]). In particular, it is an open problem as to whether there exists a polynomial time algorithm for producing a "good" user view and guaranteeing a minimum solution. We are currently exploring how to do this for "well-structured" workflows such as those found in business processes (e.g. BPEL [28]).

Acknowledgment

We would like to thank S. Cohen for her help in collecting real scientific workflows, and S. Khanna for his help in wrestling with the open problem of finding a minimum solution.

References

[1] S. Bowers and B. Ludäscher, "Actor-oriented design of scientific workflows," in *Int. Conf. on Concept. Modeling*, 2005, pp. 369–384.

[2] I. Foster, J. Vockler, M. Wilde, and Y. Zhao, "Chimera: A virtual data system for representing, querying, and automating data derivation," in *SSDBM*, 2002, pp. 37–46.

[3] 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*, vol. 20(1), pp. 3045–3054, 2003.

[4] C. B. Medeiros, J. Perez-Alcazar, L. Digiampietri, J. G. Z. Pastorello, A. Santanche, R. S. Torres, E. Madeira, and E. Bacarin, "Woodss and the web: annotating and reusing scientific workflows," *SIGMOD Rec.*, vol. 34, no. 3, pp. 18–23, 2005.

[5] L. Moreau, "The first provenance challenge." 2006, http://twiki.ipaw.info/bin/view/Challenge/.

[6] S. Cohen, S. Cohen-Boulakia, and S. Davidson, "Towards a model of provenance and user views in scientific workflows," in *Data Integration in the Life Sciences*, ser. LNBI, vol. 4075. Springer, 2006, pp. 264–279.

[7] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom, "ULDBs: Databases with Uncertainty and Lineage." in *VLDB*, 2006, pp. 953–964.

[8] O. Biton, S. Cohen-Boulakia, and S. Davidson, "Querying and managing provenance through user views in scientific workflow systems," in *U. of Penn., TR # MS-CIS-07-13*, 2007.

[9] ——, "Zoom*UserViews: querying relevant provenance in workflow systems," in *VLDB (demo)*, 2007.

[10] P. Buneman, A. Chapman, and J. Cheney, "Provenance management in curated databases," in *SIGMOD*, 2006, pp. 539–550.

[11] W. van der Aalst *et al.*, "Workflow patterns initiative." 2007, http://www.workflowpatterns.com/.

[12] 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*, ser. LNCS, vol. 4145. Springer, 2006, pp. 133–147.

[13] J. Zhao, C. Wroe, C. Goble, R. Stevens, D. Quan, and M. Greenwood, "Using semantic web technologies for representing e-science provenance," in *ISWC*, 2004, pp. 92–106.

[14] D. Bhagwat, L. Chiticariu, W. Tan, and G. Vijayvargiya, "An annotation management system for relational databases," in *VLDB*, 2004, pp. 900–911.

[15] P. Buneman, S. Khanna, and W. Tan, "Why and where: A characterization of data provenance." in *ICDT*, 2001, pp. 316–330.

[16] Y. Cui and J. Widom, "Lineage tracing for general data warehouse transformations," in *VLDB*, 2001, pp. 471–480.

[17] R. Müller, U. Greiner, and E. Rahm, "$A_{gent}W_{ork}$: a workflow system supporting rule-based workflow adaptation." *Data Knowl. Eng.*, vol. 51, no. 2, pp. 223–256, 2004.

[18] M. Litzkow, M. Livny, and M. Mutka, “Condor - a hunter of idle workstations,” in *Proc. Conf. of Distributed Computing Systems*, 1988.
[19] Y. Simmhan, B. Plale, and D. Gannon, “A survey of data provenance in e-science.” *SIGMOD Rec.*, vol. 34(3), pp. 31–36, 2005.
[20] R. Bose, I. Foster, and L. Moreau, “Report on the international provenance and annotation workshop.” *SIGMOD Rec.*, vol. 35(3), pp. 51–53, 2006.
[21] D. Harel, “Statecharts: A visual formalism for complex systems.” *Science of Comp. Programming*, vol. 8, pp. 231–274, 1987.
[22] D. Liu and M. Franklin, “GridDB: A data-centric overlay for scientific grids,” in *VLDB*, 2004, pp. 600–611.
[23] R. S. Barga and L. A. Digiampietri, “Automatic generation of workflow provenance.” in *IPAW*, ser. LNCS, vol. 4145. Springer, 2006, pp. 1–9.
[24] S. Miles, P. Groth, M. Branco, and L. Moreau., “The requirements of recording and using provenance in e-science experiments.” *Journal of Grid Computing*, 2006.
[25] V. Christophides, R. Hull, and A. Kumar, “Querying and splicing of xml workflows.” in *Cooperative Inf. Systems*, ser. LNCS, vol. 2172. Springer, 2001, pp. 386–402.
[26] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo, “Querying business processes,” in *VLDB*, 2006, pp. 343–354.
[27] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo, “Managing rapidly-evolving scientific workflows.” in *IPAW*, ser. LNCS, vol. 4145. Springer, 2006, pp. 10–18.
[28] “BPEL. business process execution language for web services.” http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.