

*Department of Computer & Information Science*

*Departmental Papers (CIS)*

---

*University of Pennsylvania*

*Year 2006*

---

Towards a Model of Provenance and User  
Views in Scientific Workflows

Shirley Cohen\*

Sarah Cohen-Boulakia†

Susan B. Davidson‡

\*University of Pennsylvania

†University of Pennsylvania

‡University of Pennsylvania, susan@cis.upenn.edu

Postprint version. Published in *Lecture Notes in Computer Science*, Volume 4076, July 2006, pages 264-279.

Publisher URL: <http://dx.doi.org/10.1007/11799511>

This paper is posted at ScholarlyCommons.

[http://repository.upenn.edu/cis\\_papers/287](http://repository.upenn.edu/cis_papers/287)

# Towards a Model of Provenance and User Views in Scientific Workflows

Shirley Cohen, Sarah Cohen-Boulakia, and Susan Davidson

Department of Computer and Information Science  
University of Pennsylvania, USA  
{shirleyc,sarahcb,susan}@seas.upenn.edu

**Abstract.** Scientific experiments are becoming increasingly large and complex, with a commensurate increase in the amount and complexity of data generated. Data, both intermediate and final results, is derived by chaining and nesting together multiple database searches and analytical tools. In many cases, the means by which the data are produced is not known, making the data difficult to interpret and the experiment impossible to reproduce. Provenance in scientific workflows is thus of paramount importance.

In this paper, we provide a formal model of provenance for scientific workflows which is general (i.e. can be used with existing workflow systems, such as Kepler, myGrid and Chimera) and sufficiently expressive to answer the provenance queries we encountered in a number of case studies. Interestingly, our model not only takes into account the chained and nested structure of scientific workflows, but allows asks for provenance at different levels of abstraction (*user views*).

## 1 Introduction

Fueled by technologies capable of producing massive amounts of data, scientists are faced with an explosion of information which must be rapidly analyzed and combined with other data to form hypotheses and create knowledge. Scientific analyses are thus becoming increasingly large and complex, with a commensurate increase in the amount and complexity of data generated.

To address this problem, over the past several years a number of scientific workflow systems have been developed to support scientists in the analysis of their data. Such systems differ from business-oriented workflow systems in the focus on data – e.g. sequences, phylogenetic trees, proteins – and its transformation into hypotheses and knowledge [23]. Examples of scientific workflow systems include myGrid/Taverna [19], Kepler [5], Chimera [12] and DiscoveryNet [22] (see [30]). Still other interesting examples of workflow systems include MHOLline [25], HKIS-Amadea [9], and AdaptFlow [14]. Some integration solutions also include workflows to add value to warehoused data. For example, the GUS [11] system allows users to import data of interest, run bioinformatics tools over that data, and store the results obtained; pipelines are expressed using Perl.

Scientific workflows are specified using a variety of graph-based models. Nodes in the workflow specification represent *step classes* (alternatively called tasks, actors, processes, boxes) and edges capture the flow of data between step classes. In many workflow systems (e.g. Kepler and myGrid), a step class may itself be a workflow. An execution of a workflow generates a partial order of *steps*, each of which has a set of *input* and *output* data objects. Each step is an instance of a step class, and the input-output flow of data and class associated with each step must conform to the workflow specification (see for example [16]).

In workflow systems, data, both intermediate and final results, is thus derived by chaining and nesting together multiple database searches and analytical tools. In many cases, the means by which the data are produced is not known, making the data difficult to interpret and the experiment impossible to reproduce. Provenance in scientific workflows is thus of paramount and increasing importance, as evidenced by recent specialized workshops [2] and surveys [23] dedicated to the subject of provenance of scientific information.

Many systems using scientific workflows provide a way to keep track of the origins of data. For example, the GUS schema contains about twenty tables dedicated to provenance information. Some scientific workflow systems, such as myGrid [28], record various kinds of metadata related to provenance. Recently, Kepler has developed a logging mechanism for tracking information and dependencies between components of the data flow [4]. Nevertheless, no formal model of provenance for workflow systems has to our knowledge been developed which precisely defines the meaning of provenance taking into account the nested structure of step classes and the data produced.

Formal models of provenance do exist within the database community (see for example [6, 3, 27]). However, these models reason over restricted forms of algebraic queries and give very fine-grained reasoning; for example, a tuple in a result gets its value from a particular set of tuples in the input (*where* provenance) and is there because of a (possibly bigger) set of input tuples (*why* provenance). More recently, [7] considers the problem of copying data between databases, and describes an approach in which these actions can be automatically recorded in a convenient, queryable form with acceptable overhead. However, the problem of tracking provenance in scientific workflow systems raises new challenges. First, since the operators in workflows are black boxes (step classes), fine grained reasoning cannot be performed. The most that can be assumed is that steps are *deterministic*, i.e. that given the same set of input the output will be the same. This input must include not only data but also user input (e.g. the selection of results based on visual inspection), parameter settings (e.g. the kind of matrix used in a Blast tool), and any other input used by the step (e.g. a randomize number used in a bootstrap). Second, scientific workflow systems frequently provide a notion of *user views* which determines whether or not a user can zoom into a step class to see a sub-workflow. User views therefore affect the granularity at which provenance is reasoned about.

The aim of this paper is to present a formal model of provenance in workflow systems which takes into account the chained and nested structure of scientific

workflows as well as user views. The model has been formulated by interviewing numerous scientists in several domains (e.g. genomic research, and phylogenetic tree construction and analysis) and analyzing what several important scientific workflow systems are currently doing. The model is abstract, *i.e.* it details the minimum information that must be provided by a workflow system in order to perform the types of reasoning about provenance that scientists wish to perform. It is generic in the sense that it can be used by any workflow systems providing this minimum information.

This paper is organized as follows. We first present one of the use cases collected (Section 2) from our interviews of scientists, whose data provenance requirements are representative of those of other studies. We then introduce our model of provenance (Section 3) and in Section 4 show how it can be used to express the provenance queries of Section 2. In Section 5 we show the connection to nested transactions, and discuss whether or not the required provenance information is provided by the logging mechanisms of Kepler, myGrid and Chimera. Finally, Section 6 concludes the paper.

## 2 Tree Inference Use Case

Systematic biologists are attempting to develop a comprehensive history of life's origins by studying the phylogenetic relationships of the millions of earth species. Assembling these species and placing them on the "tree of life" requires increased amounts of information about each one as well as sophisticated analytical tools to build an understanding of the relationships among species. At present, the infrastructure used to manage the flow of phylogenetic data lacks the querying capabilities needed to address many important scientific challenges.

As an example, consider a typical tree inference workflow depicted in Figure 1. This workflow is composed of four main step classes (S1 to S4); the last step class is nested and composed of four step classes (S4a to S4d).

The Download Sequence step class (S1) is responsible for obtaining a set of chosen DNA sequences from GenBank. Note that the input to this step class is a user-driven event. The second step class, Create Alignment (S2), takes in the raw sequences and runs an alignment program, such as ClustalW [15], to generate a multiple sequence alignment. The third step class, Refine Alignment (S3), is where the biologist verifies and improves the quality of the multiple sequence alignment by manually adjusting gaps inserted by the alignment program.

The fourth step class, Infer Tree (S4), takes the edited alignment and produces from it a phylogenetic tree. Note that this step class contains multiple substeps within it. The first substep class, Compute Trees (S4a), runs a tree inference program like PAUP [24] or Phylip [21] and generates a set of unrooted trees from the alignment. The second substep class, Create Consensus Tree (S4b), computes a consensus tree from the set of unrooted trees. The third substep class, Bootstrap Tree (S4c), calculates a confidence score for each node of the consensus tree. The last substep class, Root Tree (S4d), consists of rooting the consensus tree by selecting a site as an outgroup. The output from this

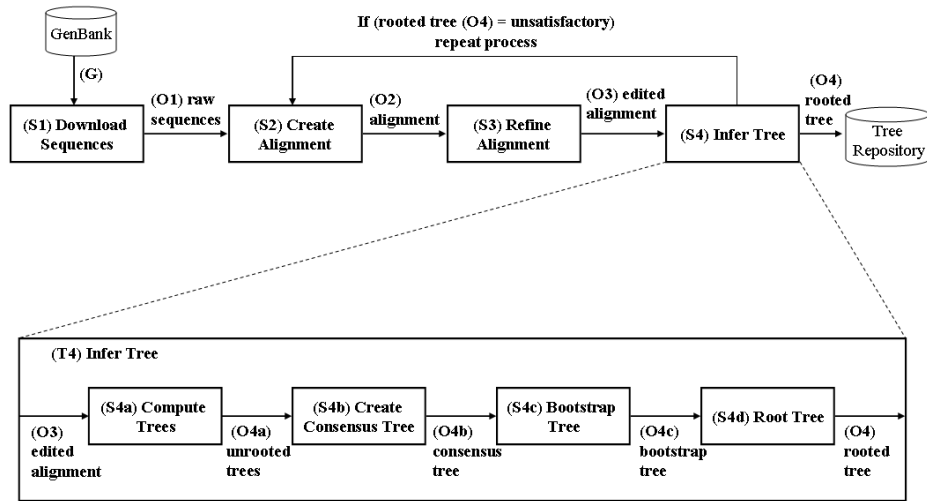


Fig. 1. Tree inference use case

substep class, a rooted tree, is saved if it is considered biologically meaningful. Otherwise, the alignment and inference step classes are repeated until a suitable rooted tree is derived.

A typical lab executes this scenario several times a year, resulting in vast amounts of intermediate and final data products. However, with current workflow technology this scenario is carried out without the ability to ask questions about how a phylogenetic tree came to be and what alignment and sequences it originated from. A biologist wishes to not only review the current state of a phylogenetic analysis that is in progress, but also guide it to some desired future state; such as refining the parameters to Clustal to produce a more precise alignment or foreseeing (based on historical results) that Phylip may produce fewer trees than PAUP\*. The biologist also wishes to know which sequences were dropped by the alignment program and consequently were not used to infer the rooted tree. A related goal is to be able to assess the quality and impact of a data product such as a rooted tree by reviewing both the DNA sequences and alignment used to produce it, and understanding which subsequent workflow executions used the same alignment as input to a tree inference step. In related studies, increased knowledge of data provenance will allow the biologist to reuse intermediate products, such as the many unrooted trees which can be quite time-consuming to generate. To address these needs we collected some data provenance queries that describe in words the semantics of the queries we are interested in:

1. What direct data products did this tree originate from?
2. What are all the data products which have been used to produce this tree?
3. What step produced this tree?
4. What sequence of steps produced this tree?

5. What parameters and steps produced this tree?
6. What alignments in the space of stored data objects were used as inputs to steps in subsequent workflows?
7. What trees in the data space were inferred using the same sequence of steps, parameters, and input data?
8. What steps require user input data?

It should be noted that there is a strong connection between questions about data provenance and general questions about workflow execution, and the biologist is interested in discovering useful facts about both. Some general workflow queries are shown below:

- What steps in this workflow did not complete or execute?
- What steps ran concurrently in the same workflow instance?

In this paper, we will concentrate on the first set of queries, the *data provenance queries*. However, a longer-term goal is to allow biologists to interactively explore other aspects of a workflow execution without needing to become an expert in the logging mechanisms of the system.

### 3 Model of provenance

Provenance is defined over a workflow execution as a function which takes as input the identifier of a data object and returns the sequence of steps and input data objects on which it depends. All data that is produced by some step is called *calculated* data. In contrast, *user* or *parameter* data is injected into the data space of the workflow execution by a user; its provenance is whatever information is recorded about how it was input, e.g. the user who input the data and the time at which the input occurred. We call this *Info(d)*.

**Definition 1.** *The provenance of a data object  $d$  ( $Prov(d)$ ) is given as:*

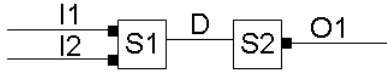
$$Prov(d) = \begin{cases} (sid, \{d_1 : Prov(d_1), \dots, d_n : Prov(d_n)\}) & d \text{ is calculated data} \\ Info(d) & d \text{ otherwise} \end{cases}$$

where *sid* is the id of the step that produced  $d$  as output, and  $d_i$  is the id of data input to the step.

As an example, consider the simple toy workflow in Fig. 2, in which  $S_1$  takes as input  $\{I_1, I_2\}$ , produces as output  $\{D\}$ , which is taken as input to  $S_2$ , which produces as output  $\{O_1\}$ . Then

$$Prov(O_1) = (S_2, \{D : (S_1, \{I_1 : Info(I_1), I_2 : Info(I_2)\})\})$$

Note that *Prov(d)* gives complete information about how the data object came to be (*deep* provenance). We could also have defined *Prov(d)* to consider just the immediate provenance or *n*-deep provenance of a data product.



**Fig. 2.** Example of workflow

Throughout this section, we will use Datalog to define the minimal information needed from the workflow system (base predicates) as well as the reasoning we can perform over that information. We choose Datalog since it is a simple, declarative language which easily expresses recursive queries; it is thus a natural model for graph data and queries which entail finding paths. Using known translations to the relational model, the provenance system described in this paper could then be implemented using a relational database system which provides support for transitive closure (e.g. Oracle or DB2). (See [26] for a description of Datalog and its translation to the relational model and [7] for a discussion of how to optimize performance.)

Another option would have been to use an object-oriented data model as suggested by the definition of  $Prov(d)$ . While this model avoids the problem of having to flatten nested sets of data, it does not naturally capture transitive closure. Furthermore, our model of provenance does not need any of the object features it supports (such as inheritance or polymorphism). We therefore opt for a simpler and more declarative model.

### 3.1 Minimal information to reason about provenance

To be able to reason about provenance, we make a number of assumptions about the information provided by the workflow system:

- **Provenance information for user or parameter data is provided.** That is,  $Info(d)$  is available.
- **Each output data object has a unique id.** The notion of unique ids for output objects is ubiquitous in proposals for scientific workflow. For example, in [27], data is never overwritten or updated in place, and each version of data has a different id; in [5], each token has a unique id although two tokens may correspond to the same data object.
- **The system maintains information about steps and the ordering of input/output operations to steps.** In order to reason about provenance, some sort of logging must be performed by the system. We will discuss how to achieve this in Section 5.

We therefore model the minimal information that must be provided to a provenance reasoning system as the following base predicates, where  $did$  is the id of a data object,  $annot$  is the provenance information of user or parameter data ( $Info(did)$ ),  $sid$  is the id of a step, and  $ts$  is an integer that captures the partial order of input and output events to a step:

$info(did, annot)$   
 $input(sid, did, ts)$   
 $output(sid, did, ts)$

To allow users to see the value of data objects and obtain information about the step class of which  $sid$  is an execution, we use  $csid$  as the id of a step class and add:

$value(did, v)$   
 $instanceOf(sid, csid)$   
 $infoClass(csid, info)$

Using these base predicates, we can express  $Prov(d)$  for calculated data using the following Datalog rule:

$$prov(did, sid, iid) : \neg input(sid, iid, tsi) \wedge output(sid, did, tso) \wedge tsi \leq tso$$

Note that our definition of provenance includes both the step and input data to that step. However, it will also be useful to talk about the set of data objects on which calculated data depends ( $dProv$ ), either directly or indirectly, as well as the set of steps on which were used in calculating the data ( $sProv$ ):

$dProv(did, iid) : \neg prov(did, -, iid)$   
 $dProv(did, iid) : \neg prov(did, -, x) \wedge dProv(x, iid)$

$sProv(did, sid) : \neg prov(did, sid, -)$   
 $sProv(did, sid) : \neg prov(did, -, x) \wedge sProv(x, sid)$

Returning to the toy example of Fig. 2, since  $prov(D, S1, I1)$ ,  $prov(D, S1, I2)$ , and  $prov(O1, S2, D)$  are true, we can infer  $dProv(O1, D)$ ,  $dProv(O1, I1)$ ,  $dProv(O1, I2)$ . We can also infer  $sProv(O1, S2)$ ,  $sProv(D, S1)$  and  $sProv(O1, S1)$ .

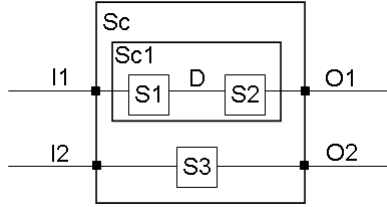
### 3.2 Composite steps

In many workflow systems, a step class may itself be a workflow. We call such step classes *composite*, and their executions *composite steps*; step classes that do not contain workflows will be called *base*, and their executions *base steps*. Typically, each input to a composite step class is input to one or more of its substep classes, and the output of a substep class is either input to another substep class or becomes the output of the composite step class.

There are several reasons why composite step classes are used in workflows. First, users may wish to focus on a certain level of abstraction and ignore lower levels of detail. Second, they may represent levels of “authorization”; users without the appropriate clearance level would not be allowed to see the lower level executions of a step class.

**Definition 2.** *Given a workflow specification, the user view of a user (or class of users)  $U$ ,  $UserView(U)$ , is the set of lowest level step classes that  $U$  is entitled to see.*

Note that a user view cannot contain two step classes such that one is contained in the other. We assume that the user view is *valid*, i.e. that each of the highest level step classes in the workflow specification is either in the view, or that at some lower level all of its contained substeps are in the user view. For example, consider Fig. 3. In this workflow,  $S_C$  directly contains  $S_{C1}$  and transitively contains step classes  $S_1$  and  $S_2$ . The composite step class at the highest level,  $S_C$ , has input set  $\{I_1, I_2\}$  and output set  $\{O_1, O_2\}$ . Within  $S_C$  there is a composite step class  $S_{C1}$  which takes  $\{I_1\}$  as input and produces  $\{O_1\}$  as output;  $S_C$  also contains step class  $S_3$  which takes  $\{I_2\}$  as input and produces  $\{O_2\}$  as output. Within  $S_{C1}$  there is a step class  $S_1$  which takes  $\{I_1\}$  as input and produces  $\{D\}$  as output;  $\{D\}$  is then input to step class  $S_2$ , which produces  $\{O_1\}$  as output.



**Fig. 3.** Example of composite Step

Three examples of user classes for this workflow are:

- $UserView(U_1) = \{S_C\}$  (the “black box” user class)
- $UserView(U_2) = \{S_{C1}, S_3\}$
- $UserView(U_3) = \{S_1, S_2, S_3\}$  (the “admin” user class)

However, the user view  $\{S_{C1}\}$  is not valid since  $S_3$  is missing.

A partial ordering  $<_u$  on user views can be defined using the containment of step classes.

**Definition 3.** *Given two user views  $U_1$  and  $U_2$ , we say that  $U_2$  is a finer level than  $U_1$  (or  $U_1$  is a higher level than  $U_2$ ),  $U_1 <_u U_2$ , iff*

*$\forall s_2 \in UserView(U_2) \exists s_1 \in UserView(U_1)$  such that  $s_1 = s_2$  or  $s_1$  contains  $s_2$  either directly or transitively.*

For example,  $U_1 <_u U_2$ ,  $U_2 <_u U_3$  and  $U_1 <_u U_3$ .

To answer questions of provenance, we must take the user view into account and reason about the input and output to steps which are instances of step classes that are in the user view. That is, we must know the connection between the specification and the execution of a workflow, as well as the containment relationship between step classes. We therefore assume that the workflow system provides the following information:

- **The user view of each class of users.** A variety of techniques could be used to capture this information. For example, the GUI in Kepler allows users to zoom in on steps. We can imagine capturing this information by taking each composite class, zooming in to the appropriate level, and taking the union of the resulting classes.
- **The input and output to each step, whether composite or base.**

Thus we use the following as our base predicates, where *sid* is the id of a step (either base or composite), *did* is the id of a data object, *ts* captures the partial order of input and output events to a step, *cid* is the id of a step class (either base or composite), and *ccid* is the id of a composite step class.

*Cinput*(*sid, did, ts*)  
*Coutput*(*sid, did, ts*)  
*immContains*(*ccid, cid*)  
*userView*(*u, cid*)

*Cinput* (*Coutput*) is *input* (*output*) extended to composite steps. The relation *contains*(*ccid, cid*), denoting the complete containment relation between step classes, can be trivially computed as the transitive closure of the immediately contains relation, *immContains*. Furthermore, the following constraint on *userView* expresses the fact that *cid* is the lowest level that *u* is entitled to see: If *contains*(*ccid, cid*) and *userView*(*u, ccid*) holds, then *userView*(*u, cid*) does not hold.

It will also be convenient to talk about steps (whether base or composite) that are allowed to be seen by a particular user:

$userInstance(u, sid) : \neg instanceOf(sid, cid) \wedge userView(u, cid)$

Using these predicates, we calculate provenance as a function of the user view as follows:

$userProv(u, did, sid, idid) : \neg Cinput(sid, idid, tsi) \wedge Coutput(sid, did, tso) \wedge tsi \leq tso \wedge userInstance(u, sid)$

We can also redefine the data (step) provenance with respect to a user view, *userDProv*(*u, did, iid*) (*userSProv*(*u, did, sid*)) using *userProv* instead of *prov*. (Details are omitted.)

### 3.3 Reasoning with user views

We now explore properties of provenance as a function of user view. In particular, when a user views the execution at a finer level he may see data objects that are not visible at a higher level which are the output of hidden substeps. Reasoning about provenance at a finer level will also allow a more precise view of the provenance of a data object.

For example, in the workflow of Figure 3, from user views  $U_1$  and  $U_2$  the data object  $D$  is not visible as a data object on which  $O_1$  or  $O_2$  depends. Furthermore, at user view  $U_1$  both  $I_1$  and  $I_2$  are seen as data objects on which  $O_1$  depends, while at user views  $U_2$  and  $U_3$  only  $I_1$  is included.

The observation about what data objects  $d$  are visible within a user view  $u$  can be formalized as follows:

$$\begin{aligned} invisible(d, u) &: \neg output(-, d, -) \wedge \neg visible(d, u) \\ visible(d, u) &: \neg userProv(u, d, -, -) \\ visible(d, u) &: \neg userProv(u, -, -, d) \end{aligned}$$

For example, consider the workflow of Figure 3 and the user view  $U_2$ . Then  $userProv(U_2, O1, SC1, I1)$  and  $userProv(U_2, O2, S3, I2)$  hold, meaning that we can infer  $visible(O1, U_2)$ ,  $visible(I1, U_2)$ ,  $visible(O2, U_2)$ , and  $visible(I2, U_2)$ . Furthermore, since  $output(S1, D, -)$  holds but not  $visible(D, U_2)$ ,  $invisible(D, U_2)$  holds. Similarly, we could show that  $invisible(D, U_1)$  holds.

To formalize the second observation, given data object  $d$  and two user views  $u_1$  and  $u_2$ , let  $DProv(u_1, u_2, d)$  be the set of all data objects that  $d$  depends on either directly or indirectly as seen in user view  $u_2$  that are visible in  $u_1$ . More precisely, it is the set of data objects  $X$  in  $ans(X)$  below (where parameter  $\$U1$  is set to  $u_1$ ,  $\$U2$  is set to  $u_2$  and  $\$D$  is set to  $d$ ):

$$ans(X) : \neg userDProv(\$U2, \$D, X) \wedge visible(X, \$U1)$$

As an illustration, consider the workflow of Fig. 3 with  $\$U1=U_1$ ,  $\$U2=U_3$  and  $\$D=O1$ . Then  $userDProv(U_3, O1, D)$ ,  $userDProv(U_3, O1, I1)$  and  $visible(O1, U_1)$  hold, but  $visible(D, U_1)$  does not hold. Thus  $DProv(U_1, U_3, O1)=\{I1\}$ .

The observation about the refinement of data provenance as a function of user view can now be stated as follows:

**Lemma 1.** *Given a data object  $did$  and two user views  $u_1$  and  $u_2$ , such that  $u_1 <_u u_2$  and  $did$  is visible in  $u_1$ . Then*

$$DProv(u_1, u_1, did) \supseteq DProv(u_1, u_2, did).$$

Returning to our example, recall that  $U_1 <_u U_3$ . It can be easily checked that  $DProv(U_1, U_1, O1)=\{I1, I2\}$  and thus  $DProv(U_1, U_1, O1) \supseteq DProv(U_1, U_3, O1)$ .

### 3.4 Discussion

Much of the information (base predicates) that we are assuming are easily obtainable from either the workflow specification (*immContains*, *userView*, *info*, *infoClass*), or from low-level logging/execution knowledge (*input*, *output* and *instanceOf*). However, many workflow systems do not keep intermediate data products, that is  $value(did, v)$  may not be available for all  $did$ . In this case, the workflow system may be able to provide only partial information about provenance, i.e. the  $did$  of data objects.

The remaining predicates, *Cinput* and *Coutput*, are the topic of Section 5.

Is it reasonable to require that the value of all intermediate data objects be kept? An increasing number of optimization and compression techniques to efficiently record provenance information have been proposed in the database

community. In particular, [7] exploits the hierarchical structure of data to optimize provenance storage, and gives experimental results to show that provenance can be tracked and managed efficiently. In the context of scientific workflows, which are run many times and generate a large number of intermediate results, the nesting of composite steps and use of user views also gives the ability to limit the results. However, the results are kept around only if they are visible in some user view. By specifying appropriate user views, the system can therefore limit the promises made to users about provenance information.

## 4 Querying provenance

We now turn to the queries about provenance introduced in Section 2, and show that they can be answered using the predicates developed in Section 3. Note that these queries concern data (1,2,5) and step (3,4) provenance and use immediate (1,3) as well as deep (2,4,5) provenance information.

In what follows, we assume that the user view is input as parameter  $\$U$  and the data object as parameter  $\$D$ . Examples are given in terms of data object O4 in the *Tree inference* workflow of Figure 1.

1. **Which data objects have been directly used to produce this result?**  
 $ans(X) : -userProv(\$U, \$D, -, X)$

If the input user view contains step S4, then the immediate provenance of O4 given by  $ans(X)$  above is {O3}. However, if the input user view contains steps S4a-d, then  $ans(X)$  is {O4c}.

2. **What are all the data objects which have been used to produce this result?**  
 $ans(X) : -userDProv(\$U, \$D, X)$

$ans$  returns {O1,O2,O3} if the input user view contains step S4, and {O1,O2,O3,O4a,O4b,O4c} if it contains steps S4a-d.

3. **What step class produced this data product?**  
 $ans(X) : -userProv(\$U, \$D, X, -)$

If the input user view contains step S4, then  $ans(X)$  is {S4}. However, if the input user view contains steps S4a-d, then  $ans(X)$  is {S4d}.

4. **What sequence of steps produced this data product?**  
 $ans(X) : -userSProv(\$U, \$D, X)$   $ans$  returns {S1, S2, S3, S4} if the input user view contains step S4, and {S1, S2, S3, S4a,S4b,S4c,S4d} if it contains steps S4a-d.

5. **What parameters and steps produced this data product?**

The intent of this query is to know the input to each step that led to the data product. Note that to distinguish parameters from other input data we need additional information from the workflow system, e.g. the predicates

$parameter(d)$ ,  $userInput(d)$  and  $calculated(d)$ , which could then be used by our system in a straightforward way.  $ans(X, Y) : -userProv(\$U, \$D, X, Y)$ ,  $parameter(Y)$   
 $ans(X, Y) : -userProv(\$U, Z, X, Y), ans(., Z)$  Assuming the input user view contains step S4,  $ans$  returns  $\{(S1,G), (S2,O1), (S3,O2), (S4,O3)\}$ . To answer the original query, this set would be filtered for the second component to be a parameter resulting in the empty set (all inputs are calculated data in this example).

Details of queries 6-8 can be found in [10].

## 5 Obtaining *Cinput* and *Coutput* from Logs

Up to this point, we have assumed that *Cinput* and *Coutput* are available to define the provenance of a data object. We now argue that this information is achievable using standard nested transaction logging mechanisms, and discuss how to obtain this information in Kepler, MyGrid and Chimera.

*Logging of nested transactions.* Using ideas from nested transactions [18], the log of the workflow system would contain the events – start ( $s$ ), read ( $r$ ), write ( $w$ ), and commit ( $c$ ) – not just of base transactions but of transactions within which they are nested. For example, the following could be the log of the (composite) transaction  $T_1$  which contains subtransaction  $T_2$ , which in turn contains the (base) transaction  $T_3$ :

$$s(T_1), s(T_2), r(d_1), w(d_2), s(T_3), r(d_2), r(d_3), w(d_3), c(T_3), w(o_1), c(T_2), c(T_1)$$

In this case,  $input(sid, did, ts)$  is computed as the data read and the order in which it was read. For example,  $input(T_3, d_2, 5)$  could be true. With composite transactions, the output would be calculated as all the data that is output by some subtransaction and not input to another subtransaction; the input of a composite transaction is defined analogously. For example,  $input(T_1, d_1, 2)$  and  $input(T_1, d_3, 5)$  would be true but  $input(T_1, d_2, 6)$  would not be true.

Note that we can compute *Cinput* and *Coutput* from the log events of nested transactions since it contains the notion of execution of composite steps as well as base steps.

*Kepler.* In Kepler, a workflow consists of a collection of nodes called *Actors* (corresponding to step classes) which communicate through input and output *ports*. Communication occurs through the passing of *tokens* (corresponding to data input and output) which are globally unique; tokens are read and written, and each token is written only once. The model of computation of a workflow is defined by a *Director* who mediates communication between actors.

The log associated with this model records the reading and writing of tokens on ports, which are uniquely associated with Actors [4]. Each execution of an actor corresponds to a step in the terminology of this paper.

Conceptually, the first read event on a port associated with an Actor begins the execution (transaction) of that Actor. Subsequent writes by that Actor on this port depend on all its previous reads, where “previous” is captured by an integer called a firing. Since this implies that the state (read tokens) of the Actor gets bigger and bigger as time goes on, the notion of a *clear* event is introduced and recorded in the log, the effect of which is to clear the state of the Actor. Thus any write after the clear event will depend only on the read events since the state was cleared. In terms of transactions, this can be thought of as committing a transaction and beginning a new transaction.<sup>1</sup>

Using the Kepler log, it is certainly possible to capture *input* and *output*. Moreover, Kepler supports composition of Actors, and enables users to zoom in and view finer levels of detail of an Actor. However, since the log records only events of base steps, there is currently no notion of the execution of a composite step. Thus it is not clear how to calculate *Cinput* and *Coutput* for composite steps. The Kepler group is exploring a variety of approaches to work around this problem [17].

*myGrid*. In myGrid [19],<sup>2</sup> a workflow is a network of processors and links. A *processor* (corresponding to a step class) is a transformation that accepts a set of input data and produces a set of output data. Several types of processors exist, one of which is the nested processor. Two kinds of links are considered: *data links*, which mediate the flow of data between a data source and sink; and *coordination constraint links*, which control the execution of two processors (roughly speaking, playing the role of a director in Kepler). The log file in myGrid is an XML file which records global execution information: the user of the workflow, the start time, the end time and the set of services invocations performed (each invocation corresponds to a step). Exploiting the nested structure of XML, information is also provided for each service invocation: start time, end time, parameters of the service invocations, input data, and output data. Life Sciences Identifiers (LSIDs) [8] are used to uniquely and persistently identify data resources and their associated metadata.

An interesting aspect of myGrid is the automatic annotation of provenance logs with concepts drawn from the myGrid ontology. The COHSE<sup>3</sup> system performs this task by augmenting documents with links based on the semantic content of those documents. This process allows users to dynamically generate a hypertext collection of provenance documents, data, services, and workflows based on their associated concepts, and to perform reasoning over the ontology (see [28], [29], and [1] for more details).

Using the myGrid log, it is indeed possible to capture *input* and *output*. While the current literature does not focus on the provenance of nested processors, the

---

<sup>1</sup> This is a simplification of the model, which also uses a notion of “firings” to capture the set of read tokens on which a write depends rather than an ordering of events.

<sup>2</sup> We omit here the internal relationships between myGrid, the Scuf language, Taverna and freefluo tools.

<sup>3</sup> Conceptual Open Hypermedia Services Environment

intrinsically nested structure of the myGrid log file seems naturally suitable for capturing nested transactions, thus allowing the calculation of *Cinput* and *Coutput*.

*Chimera*. In Chimera [12],<sup>4</sup> a *transformation* is a program (script file) and an execution of a transformation is a *derivation*, corresponding to a step class and a step, respectively. Data products are represented as abstract typed datasets (virtual data) and as materialized replicas. *Derivations* can be connected to form workflows that consume and produce replicas (input and output data).

The Chimera virtual data schema defines a set of relations used to represent and capture descriptions of how a program can be invoked, and to record its potential and/or actual invocations. Upon execution, workflows automatically create *invocation objects* for each derivation in the workflow, annotated with the information of the runtime process. Invocation objects are an *annotation scheme* for representing provenance information and thereby providing a mechanism for linking input and output data products.

In Chimera, provenance information can be retrieved from the Virtual Data Catalog (VDC) [13] expressed in the Virtual Data Language (VDL). VDL supports both recursive searches and can output all the derivations in the system that produced a particular dataset. VDL interacts with an end-user query system, the Virtual Data Browser (VDB), to interactively access the catalog.

In the current implementation of Chimera, nested transformations are allowed since each transformation can call other transformations. As each derivation has its own provenance information, it should be possible to populate *Cinput* and *Coutput*.

## 6 Conclusion

This paper examines data provenance through the prism of large-scale scientific applications. Motivated by phylogenetic analyses which produce volumes of data, our research extends existing ideas of data provenance to scientific workflows. In this context, we formulate a model for provenance and define notions of *data provenance*, *step provenance*, and *user views* for computing user-oriented queries over workflow executions.

User views are especially helpful for reasoning about data provenance through nested executions. They are essential for defining the level of detail of a provenance query and determining what data must be kept by the system. As such, we devise ways in which a user can effectively query a workflow execution in an intuitive fashion without needing to become an expert in the system's logging facility. We demonstrate the expressiveness of our model by answering a collection of queries supplied by systematic biologists.

Our model is simple and generic enough to capture information that is (or soon will be) available in existing scientific workflow system, and we demonstrate

---

<sup>4</sup> We omit here the internal relationships between GriPhyN, Chimera, Pegasus and Condor.

this with Kepler, myGrid, and Chimera. From this, we show that a scientific workflow system which provides basic execution logging could implement our model and benefit from our approach.

We are currently exploring new ways to improve the expressiveness of our model. First, we will consider the general workflow queries in Section 2 related to partial and concurrent executions. Second, we will augment our model with additional semantics such as object typing to allow finer-grained queries, and explore the use of an object-oriented data model augmented with transitive closure. Third, we wish to experiment with storage models such as that proposed by the Paoa project [20] to improve query performance.

#### **Acknowledgment:**

The authors wish to thank the members of the Kepler group, particularly Bertram Ludäscher, Timothy McPhillips, and Shawn Bowers, for the many fruitful discussions about scientific workflows.

## **References**

1. Alpdemir, N., Mukherjee, A., Paton, N. W., Fernandes, A., Watson, P., Glover, K., Greenhalgh, C. Oinn, T., Tipney, H.: Contextualised Workflow Execution in myGrid, *Proc of European Grid Conference*, Springer-Verlag, LNCS **3470**, 444-453, 2005.
2. Berry, D., Buneman, P., Wilde, M., and Ioannidis, Y. editors. e-Science Workshop on Data Provenance and Annotation, *National e-Science Centre, Edinburgh*, 2003.
3. Bhagwat, D., Chiticariu, L., Tan W. C., Vijayvargiya, G.: An Annotation Management System for Relational Databases, *Proc. Conference on Very Large Data Bases (VLDB)*, 900-911, 2004.
4. Bowers, S., McPhillips, T., Ludäscher, B., Cohen, S., Davidson, S.B.: A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows. *To appear in Proc. of IPAW'06 International Provenance and Annotation Workshop*, 2006.
5. Bowers, S., Ludäscher, B.: Actor-Oriented Design of Scientific Workflows, *Proc of ER'05, International Conference on Conceptual Modeling*, 369-384, 2005.
6. Buneman, P., Khanna, S., Tan, W.: Why and Where: A Characterization of Data Provenance, *Proc. of Int. Conf. on Database Theory (ICDT)*, 316-330, 2001.
7. Buneman, P., Chapman, A., Cheney, J.: Provenance Management in Curated Databases, *To appear in Proc. of SIGMOD International Conference on Management of Data*, 2006.
8. Clark, T., Martin, S., Liefeld, T.: Globally distributed object identification for biological knowledgebases. *Briefings in Bioinformatics*, **5(1)** 59-70, 2004.
9. Cohen-Boulakia, S., Lair, S., Stransky, N., Graziani, S., Radvanyi, F., Barillot, E., Froidevaux, C.: Selecting biomedical data sources according to user preferences, *Bioinformatics, Proc. ISMB/ECCB04*, **20**, i86-i93, 2004.
10. Cohen, S., Cohen-Boulakia, S., Davidson, S.: Towards a Model of Provenance in Scientific Workflows, *University of Pennsylvania, Internal Report, #MS-CIS-06-03*, 2006.
11. Davidson, S., Crabtree, J., Brunk, B., Schug, J., Tannen, V., Overton, C., Stoekert, C.: K2/Kleisli and GUS: Experiments in integrated access to genomic data sources *IBM Systems Journal*, 2001.

12. Foster, I., Vockler, J., Woilde, M., Zhao, Y.: Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation, Proc. of the 14th *Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, 2002.
13. Foster, I., Voeckler, J., Wilde, M., Zhao, Y.: The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration Proc of Conference on Innovative Data System Research (CIDR), 2003.
14. Greiner, U., Mller, R., Rahm, E., Ramsch, J., Heller, B., Lffler, M.: AdaptFlow: Protocol-based Medical Treatment Using Adaptive Workflows. *Methods of Information in Medicine*, **44**, 80–88, 2005.
15. Higgins, D. G., Sharp, P. M.: Clustal: A package for performing multiple sequence alignment on a microcomputer. *Gene* 73: 237-244, 1998.
16. Kiepuszewski, B., ter Hofstede, A. H. M., van der Aalst, W. M. P. : Fundamentals of control flow in workflows. *Acta Inf.*, **39(3)**, 143–209, 2003.
17. McPhillips, T., Bowers, S.: An approach for pipelining nested collections in scientific workflows. *SIGMOD Record*, **34(3)**, 12–17, 2005.
18. Moss, J.E.B.: Nested Transactions: An Approach to Reliable Distributed Computing, *Ph.D. dissertation, Dept. of Electrical Engineering and Computer Science, MIT*, April 1981.
19. Oinn, T.M., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, R.T., Carver, K., Glover, Pocock, M.R., Wipat, A., Li, P. : Taverna: a tool for the composition and enactment of bioinformatics workflows, *Bioinformatics, Proc. ISMB/ECCB03*, **20(1)**, 3045–3054, 2003.
20. The Psoa Project Luc Moreau et al. <http://www.psoa.org/>
21. Phylip Programs and Documentation:  
<http://evolution.genetics.washington.edu/phylip/phylip.html>. Swofford
22. Rowe, A., Kalaitzopoulos, D., Osmond, M., Ghanem, M., Guo Y.: The discovery net system for high throughput bioinformatics *Bioinformatics*, **19(1)**, i225–i231, 2004.
23. Simmhan, Y., Plale, B., Gannon, D.: A survey of data provenance in e-science. *SIGMOD Record*, **34(3)**, 31–36, 2005.
24. Swofford D. L: PAUP\*: Phylogenetic Analysis Using Parsimony (\*and other methods). Sinauer Associates, Sunderland, MA, 2000.
25. Targino, R., Cavalcanti, M.C., Mattoso M.: An Environment to Define and Execute In-Silico Workflows Using Web Services. Proc. of *DILS 2005, Data Integration in the Life Sciences*, Springer-Verlag, LNBI **3615**, 288–291, 2005.
26. Ullman, J.D., Widom, J.: A First Course in Database Systems. Prentice-Hall, 1997.
27. Widom, J.: Trio: A System for Integrated Management of Data, Accuracy, and Lineage. *CIDR'05, Conference on Innovative Data Systems Research*, 262–276, 2005.
28. Zhao, J., Wroe, C., Goble, C., Stevens, R., Quan, D. and Greenwood, M.: Using Semantic Web Technologies for Representing e-Science Provenance, *Proc of Semantic Web Conference (ISWC)*, 92-106, 2004.
29. Zhao, J., Goble, C., Stevens, R., Bechhofer, S.: Semantically Linking and Browsing Provenance Logs for e-Science. *Proc of International Conference on Semantics of a Networked World (IC-SNW)*, Springer-Verlag, LNCS **3226**, 157–174, 2004.
30. <http://www.extreme.indiana.edu/swf-survey/>