Hiding Data and Structure in Workflow Provenance

Susan Davidson, Zhuowei Bao, and Sudeepa Roy

Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA, USA
{susan,zhuowei,sudeepa}@cis.upenn.edu

Abstract. In this paper we discuss the use of *views* to address the problem of providing useful answers to provenance queries while ensuring that privacy concerns are met. In particular, we propose a *hierarchical* workflow model, based on context-free graph grammars, in which *fine-grained* dependencies between the inputs and outputs of a module are explicitly specified. Using this model, we examine how privacy concerns surrounding data, module function, and workflow structure can be addressed.

1 Introduction

Provenance in scientific workflows is of increasing interest, as evidenced by several recent workshops, tutorials, and surveys on the topic [5,6,18,23]. A number of tools for capturing provenance have been developed in workflow systems such as myGrid/Taverna [19], Kepler [7] and VisTrails [13], and a standard for provenance representation called the Open Provenance Model (OPM) [17] has been designed. By maintaining information about the sequence of module executions (processing steps) used to produce a data item, as well as the parameter settings and intermediate data items passed between module executions, the validity and reliability of data can be better understood and results can be made reproducible.

A repository that includes workflow specifications, executions and provenance information – *provenance-aware workflow information* – is clearly useful in many ways. For example, scientists who wish to perform new analyses may search by keyword to find specifications of interest to reuse or modify. They may also search executions associated with a specification to understand the meaning of the workflow, or to correct/debug an erroneous specification. Finding erroneous or suspect data, a user may then wish to ask structural *provenance queries* to determine what downstream data might have been affected, or to understand how the process failed that led to creating the data.

However, authors/owners of workflows may wish to keep some of this provenance information private. For example, intermediate *data* within an execution may contain sensitive information, such as the social security number, a medical record, or financial information about an individual. Although users with the appropriate level of access may be allowed to see such confidential data, making it available to all users through a workflow repository, even for scientific

S. Kikuchi et al. (Eds.): DNIS 2011, LNCS 7108, pp. 41–48, 2011.

purposes, is an unacceptable breach of privacy. Beyond data privacy, a *module* itself may be proprietary, and hiding its description may not be enough: users without the appropriate level of access should not be able to *infer* its behavior if they are allowed to see the inputs and outputs of the module. Finally, details of how certain modules in the workflow are connected may be proprietary, and therefore showing how data is passed between modules may reveal too much of the *structure* of the workflow. **There is thus an inherent tradeoff between the utility of the information shown in response to a search/query and the privacy guarantees that authors/owners desire.**

One technique that can be used to hide details of a workflow is to create *composite modules* which encapsulate subworkflows. Composite modules can be combined to create *views* of a workflow and its associated executions, showing users a subset of provenance information and hiding the rest within unexpanded composite module executions. Originally proposed in [4] as a technique for focusing user attention on relevant provenance, views can also be used to hide private information, which may include the intermediate data and modules within a composite module as well as the dependencies between the inputs and outputs of the composite module.

In this paper, we examine the use of views to implement workflow provenance privacy. We start in Sec. 2 by describing a model for workflow specifications, executions, and views. We continue in Sec. 3 by describing initial results on module and structural privacy, and discuss the connection to views. We close by pointing to future directions for research.

2 Workflow Model

Our workflow model has several components: specifications, runs, execution graphs, port dependencies, and provenance graphs. A workflow specification describes the design of a workflow, while a workflow run (together with information about the data and processes) describes a particular execution of the given specification. Following [3], a specification is given by a context-free graph grammar and the runs corresponding to the specification are given by the graphs in the language generated by that grammar. Port dependencies are used in the definition of data provenance graphs, and model fine-grained dependencies between the inputs and outputs of a module. Rather than giving full details of the model, we illustrate via an example (see [2] for a more formal treatment).

Workflow Specifications. A sample *workflow specification* is given in Fig. 1. The workflow estimates disease susceptibility based on genome-wide SNP array data for an individual as well as information about lifestyle, family history, and physical symptoms, and outputs a prognosis for the patient along with recommended lifestyle changes [25]. In the graph, boxes labeled $M0, \ldots, M16$ indicate *modules* with *input ports* indicated by solid circles and *output ports* indicated by open circles; and the labeled arrows between output and input ports of different modules indicate potential *data flow*. Some of the modules in this workflow are *atomic* ($M5, \ldots, M16$). The rest of the modules ($M0, \ldots, M4$) are *composite*,

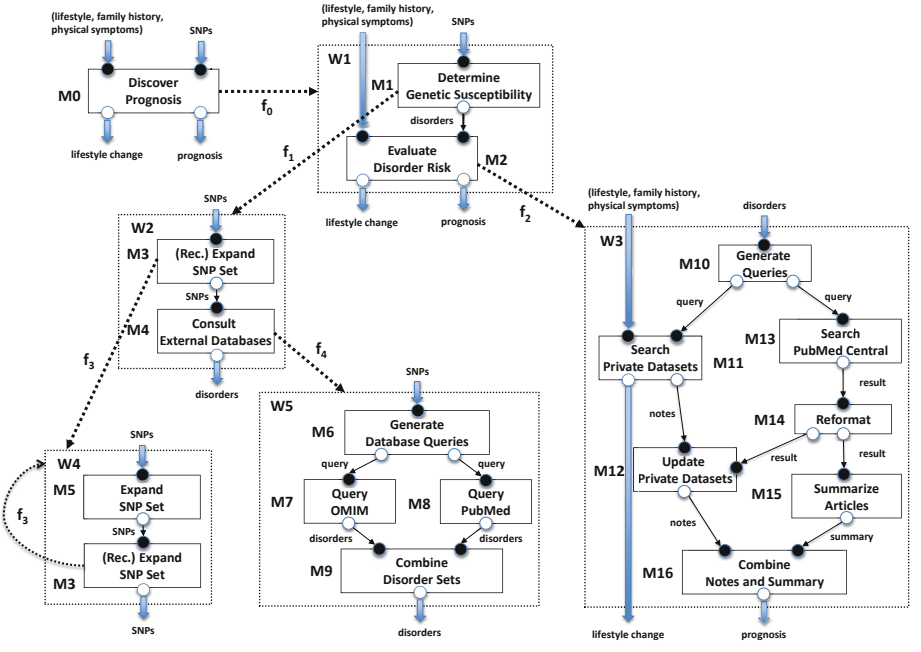

Fig. 1. Disease Susceptibility Workflow Specification

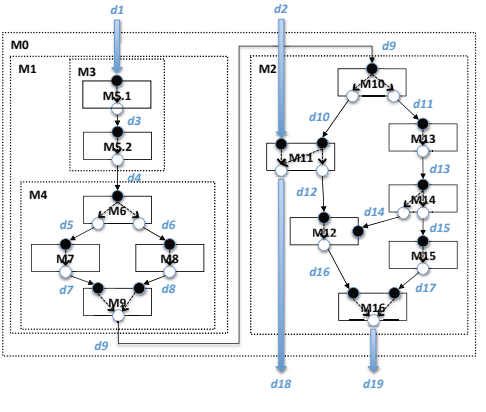

Fig. 2. Sample Workflow Execution

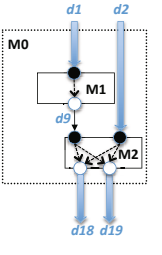

Fig. 3. View of Provenance Graph, $V1$

and their expansion to a subworkflow is shown by dotted edges labeled f_i (the name of the production rule). In particular, the root of the workflow is $M0$, which expands via f_0 to $W1$. The correspondence between inputs/outputs of a composite module and the subworkflow to which it expands is indicated in this figure by reusing names. For example, the initial inputs to the workflow are `(lifestyle,...)` and `SNPs`, and the final outputs are `lifestyle change` and `prognosis`, indicated by double arrows into and out of $M0$, and those names are reused within $W1$. There is also *intermediate* data within subworkflows $W1, \ldots, W5$, e.g. `disorders`, `query`, and `result`.

Note that composite module $M3$ is recursive, indicated by a cycle, and that therefore in an execution the atomic module $M5$ may be executed multiple times. For simplicity, we have dropped from the figure the alternate termination condition for this expansion ($M3 \longrightarrow^{f_5} M5$).

Workflow Executions. The set of all possible *runs* of a specification is modeled as the graph language of the corresponding graph grammar. More precisely, it consists of all simple workflows that can be derived from the start module and contain only atomic module. A *workflow execution* is a run in which each module is given a unique process id and data flows over the edges. One execution of our sample specification is given in Fig. 2, in which we reuse the name of the module as the process id unless the module occurs multiple times in the run, e.g. we use $M5.1$ and $M5.2$ for the two executions of module $M5$. Data items represent instances of the abstract data in the specification, e.g. $d1$ represents the initial input of `SNPs`.

Provenance Graphs. Data provenance in workflows is typically considered to be *coarse-grained* [9], i.e., the data coming out of each output port of a module depends on the data that entered *all* input ports of the module. However, the ability to capture fine-grained dependencies is increasingly important in a number of workflow systems, e.g., Taverna 2 [24] and COMAD-Kepler [22], so we allow the modeling of *fine-grained* provenance. That is, as part of the specification we assume that each atomic module has an associated *port dependency matrix* $\delta(M)$ showing which inputs are connected to which outputs. This is illustrated in our sample execution in Fig. 2 as an edge between input/output ports within a module execution, which we will call a *dependency* edge. For example, in $M11$ the output $d12$ depends only on $d10$ as there is no edge from $d2$ to $d12$. The information contained in an execution allows us to capture *provenance* for data items (such as $d18$ and $d19$), so we will call them *provenance graphs*. Note that the provenance graph for this relatively simple workflow is already complex. Note also that dependency matrix for composite modules in an execution can be inferred from the dependency matrices of atomic modules as *paths* of dependency and dataflow edges between input and output ports for the composite module.

A provenance query such as "What data does $d18$ depend on?" can be answered by finding all data items at the origin of a path of dependency and data flow edges that ends at $d18$. For our example, this would include data items $d1$, $d2, \ldots,$ $d10$, but not $d11, \ldots,$ $d17$. In contrast, $d19$ depends on all data

items ($d1, d3, \ldots,$ $d17$) but not $d2$, since there is no dependency between the first input port and second output port in $M11$.

Views. As noted earlier, certain modules in this execution are *composite*, indicated by boxes containing subworkflow executions (e.g., $M0$, $M1$, $M2$). Controlling the expansion to subworkflows can be used to create *views*, such as the one of our sample workflow execution in Fig. 3. In this view ($V1$), users can only see the expansion of $M0$ and therefore have no access to any intermediate data except for $d9$, and cannot see what modules were executed in the implementation of modules $M1$ and $M2$. For example, the answer to the provenance query "Does the prognosis $d19$ depend on the output of a PubMed search?" (where PubMed search matches modules $M8$ and $M13$) would be "yes" with respect to the full provenance graph of Fig. 2 but "no" with respect to $V1$ since these modules are not visible .

Views may also alter *fine-grained dependencies* between the input and output ports of a module, as illustrated by module $M2$ in Fig. 3. Here, there is a dependency between the first input port and second output port (the given dependency matrix for $M2$ in the view) that does not exist as a path within $M2$ in Fig. 2. In this view, the output of the provenance query "What data does $d19$ depend on?" would therefore include $d2$ and exclude all intermediate data except for $d9$ (i.e., $d1$, $d2$, $d9$).

Finally, we may hide data on edges in a view of an execution (for data privacy) or delete connections between modules in a specification and its executions (for structural privacy).

3 Privacy

Privacy concerns are tied to the workflow components: data, modules, and the structure of a workflow. To illustrate them, consider again the sample workflow in Fig. 1.

Data Privacy. Certain data in a workflow execution may be confidential. For example, the output of $M1$, i.e. the genetic disorder the patient is susceptible to, should not be revealed with high probability, in any execution, to users without the required access privilege. Such data masking is a fairly standard requirement in privacy-aware database systems, and a variety of well known techniques can be applied, e.g. access control [21]. A key question to consider is whether access to aggregated provenance data (e.g. the most frequent genetic disorder) is allowed and, if so, whether some standard notion of privacy like *differential privacy* (see, for instance, [12]) used in statistical databases is appropriate for our application. For example, often random noise is added to the output of a statistical query to achieve differential privacy in statistical databases, but adding random noise to the data values may prohibit repeatability of scientific experiments performed using a workflow.

Module Privacy. Module privacy requires that the functionality of a private module – that is, the mapping it defines between inputs and outputs – is not

revealed to users without the required access privilege. Returning to our example, assuming that $M1$ implements a function f_1, module privacy with respect to $M1$ requires that no adversarial user should be able to guess the output f_1(`SNP`, `ethnicity`) with high probability for any `SNP` and `ethnicity` input. From a patient's perspective, this is important because they do not want someone who may happen to have access to their `SNP` and `ethnicity` information to be able to determine what `disorders` they are susceptible to. From the module owner's perspective, they do not want the module to be simulated by competitors who capture all input-output relationships. It is easy to see that if information about all intermediate data is repeatedly given for multiple executions of a workflow on different initial inputs, then partial or complete functionality of modules may be revealed. The approach that we take in [11,10] is to *hide a carefully chosen subset of intermediate data*, thereby limiting the amount of provenance data shown to the user and guaranteeing some desired level of privacy. Since there may be several different subsets of intermediate data whose hiding yields the desired level of privacy, and certain data may be more useful utility-wise to users than other data, this becomes an interesting optimization problem.

Note that there is an interesting connection between data and module privacy: If a module is *public* (i.e. its function is known), then its output can be simulated if the inputs are public. Therefore, hiding the output of a public module may also require hiding some of its input. Furthermore, if a module is *invertible* then its input can be simulated if the outputs are known. Again, hiding the input of a public module may also require hiding some of its outputs.

Structural Privacy. The goal of structural privacy is to keep private the information that some module M contributes to the generation of a data item d, output by another module M'. For instance, in the execution of the workflow $W3$, we may wish to hide the fact that the reformatted data from PubMed Central (module $M13$) contributes to updating the private DB (module $M12$), and hence to the output of module $M12$. One possible approach is to delete edges and vertices from both the visible specification and its execution so as to eliminate all paths from M to M'; for instance, in this example we can delete the edge $M13 \rightarrow M14$. However, by doing so, we may hide additional provenance information that does not need be hidden (e.g. the existence of a path from $M13$ to $M15$). Another approach would be to avoid altering the structure of the workflow and instead find a view in which $M13$ and $M12$ are hidden in a composite module P, so that the reachability of any pair (u, v) in P is no longer externally visible, but in this case we may introduce some new paths that did not exist before. Since there may be many different views of the same workflow, each of which has a different composite module structure and different dependency matrices, we may need to choose the "best" view. Once again one faces a challenging optimization problem: guaranteeing an adequate level of privacy while minimizing unnecessary loss of information or introduction of spurious information. Techniques from preserving the privacy of *social networks* [14,1,20,8,16] may also be useful.

4 Conclusion

We have presented a model of workflows based on context free graph grammars in which fine-grained dependencies between inputs and outputs of an atomic (non-expandable) module can be explicitly specified. Using this model, a view can be defined using several techniques, including: 1) hiding data in an execution; 2) hiding substructure within composite modules, e.g. enabling only a subset of the workflow productions, thereby allowing only some composite modules expansions; 3) hiding data flow edges in the specification. We also discussed privacy concerns in workflow provenance – data, module and structure. Applying a view to an execution yields a subset of the provenance information, in which module executions and intermediate data of non-expandable modules are not visible, and hidden data or data flow are not revealed. Note that hiding data flow edges may introduce false negatives (data that actually is in the provenance of a given data item is not returned in a provenance query) while using composite modules may introduce false positives (data that is not actually in the provenance of a given data item is returned) and/or false negatives, depending on the fine-grained dependency graph associated with the composite module. The utility of a view to a user can be measured by the number of false positives or false negatives introduced in the view used to answer provenance queries.

Our approach of using a view to answer provenance queries while ensuring privacy of the workflow components is quite different from that used in other areas (statistical databases, data mining, social networks) where random noise is added or other randomized mechanisms are applied to guarantee privacy. These approaches do not seem to be directly applicable to our problem; provenance queries are quite different in nature from aggregate queries, and results of scientific experiments performed using a workflow are expected to be repeatable and accurate over different executions. The chief challenge is have a formal analysis of privacy and a utility guarantee of the solutions we provide, which leads to numerous new research directions. In our initial research for module privacy, we used a weaker notion of privacy called ℓ-diversity [15]. In our current work we are studying whether stronger notion of privacy (such as differential privacy) can be applied meaningfully to our application.

References

1. Backstrom, L., Dwork, C., Kleinberg, J.M.: Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In: WWW, pp. 181–190 (2007)
2. Bao, Z., Davidson, S., Milo, T.: A Fine-Grained Workflow Model with Provenance-Aware Security Views. In: Proceedings of TaPP (2011)
3. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying business processes. In: Proceedings of the 32nd International Conference on Very Large Data Bases, pp. 343–354 (2006)
4. Biton, O., Boulakia, S.C., Davidson, S.B., Hara, C.S.: Querying and Managing Provenance through User Views in Scientific Workflows. In: ICDE, pp. 1072–1081 (2008)

5. Bose, R., Foster, I., Moreau, L.: Report on the International Provenance and Annotation Workshop. SIGMOD Rec. 35(3) (2006)
6. Bose, R., Frew, J.: Lineage retrieval for scientific data processing: a survey. ACM Comp. Surveys 37(1), 1–28 (2005)
7. Bowers, S., Ludäscher, B.: Actor-oriented design of scientific workflows. In: Int. Conf. on Concept. Modeling, pp. 369–384 (2005)
8. Campan, A., Truta, T.M.: A clustering approach for data and structural anonymity in social networks. In: PinKDD (2008)
9. Davidson, S.B., Boulakia, S.C., Eyal, A., Ludäscher, B., McPhillips, T.M., Bowers, S., Anand, M.K., Freire, J.: Provenance in scientific workflow systems. IEEE Data Eng. Bull. 30(4), 44–50 (2007)
10. Davidson, S.B., Khanna, S., Milo, T., Panigrahi, D., Roy, S.: Provenance views for module privacy. In: Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 175–186 (2011)
11. Davidson, S.B., Khanna, S., Panigrahi, D., Roy, S.: Preserving module privacy in workflow provenance (2010) (manuscript), http://arxiv.org/abs/1005.5543
12. Dwork, C.: Differential Privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006)
13. Freire, J., Silva, C.T., Callahan, S.P., Santos, E., Scheidegger, C.E., Vo, H.T.: Managing Rapidly-Evolving Scientific Workflows. In: Moreau, L., Foster, I. (eds.) IPAW 2006. LNCS, vol. 4145, pp. 10–18. Springer, Heidelberg (2006)
14. Korolova, A., Motwani, R., Nabar, S.U., Xu, Y.: Link privacy in social networks. In: CIKM, pp. 289–298. ACM, New York (2008)
15. Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-diversity: Privacy beyond k-anonymity. ACM Trans. Knowl. Discov. Data 1(1), 3 (2007)
16. Machanavajjhala, A., Korolova, A., Sarma, A.D.: Personalized social recommendations: accurate or private. Proc. VLDB Endow. 4, 440–450 (2011)
17. Moreau, L., Freire, J., Futrelle, J., McGrath, R.E., Myers, J., Paulson, P.: The Open Provenance Model: An overview. In: Freire, J., Koop, D., Moreau, L. (eds.) IPAW 2008. LNCS, vol. 5272, pp. 323–326. Springer, Heidelberg (2008)
18. Moreau, L., Ludäscher, B. (eds.): Concurrency and Computation: Practice and Experience – Special Issue on the First Provenance Challenge. Wiley (2007), http://twiki.ipaw.info/bin/view/Challenge/
19. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, R., Carver, K., Pocock, M.G., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(1), 3045–3054 (2003)
20. Rastogi, V., Hay, M., Miklau, G., Suciu, D.: Relationship privacy: output perturbation for queries with joins. In: PODS, pp. 107–116 (2009)
21. Samarati, P., De Capitani di Vimercati, S., Paraboschi, S.: Access control: principles and solutions. Software—Practice and Experience 33(5), 397–421 (2003)
22. Shawn Bowers, B.L., McPhillips, T.M.: Provenance in collection-oriented scientific workflows. Concurrency and Computation: Practice and Experience 20(5), 519–529 (2008)
23. Simmhan, Y., Plale, B., Gannon, D.: A survey of data provenance in e-science. SIGMOD Rec. 34(3), 31–36 (2005)
24. Sroka, J., Hidders, J., Missier, P., Goble, C.A.: A formal semantics for the Taverna 2 workflow model. J. Comput. Syst. Sci. 76(6), 490–508 (2010)
25. Stoyanovich, J., Pe'er, I.: MutaGeneSys: estimating individual disease susceptibility based on genome-wide SNP array data. Bioinformatics 24(3), 440–442 (2008)