Preserving privacy in shared provenance data

Paolo Missier, Jeremy Bryans, Roxana Danger and Vasa Curcin

TECHNICAL REPORT SERIES

No. CS-TR-1366 January 2013

TECHNICAL REPORT SERIES

No. CS-TR-1366 January, 2013

Preserving privacy in shared provenance data

P. Missier, J. Bryans, R. Dangern and V. Curcin

Abstract

Provenance management still lacks robust models for sharing provenance data between multiple parties while keeping parts of it private to the owner. This limits the potential for provenance dissemination, which is a critical step in enabling data sharing amongst partners with limited a priori mutual trust. In turn, this has a negative impact on data-intensive science and its associated research publication repositories, on audit tasks, as well as on increasingly common collaborative dynamic coalitions scenarios. We propose a method for preserving privacy by creating abstractions over provenance graphs, we apply it to provenance sharing, and illustrate it on a health care case study.

Bibliographical details

MISSIER, P., BRYANS, J., DANGER, R., CURCIN, V.

Preserving privacy in shared provenance data
[By] P. Missier, J. Bryans, R. Danger, V. Curcin

Newcastle upon Tyne: Newcastle University: Computing Science, 2013.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1366)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1366

Abstract

Provenance management still lacks robust models for sharing provenance data between multiple parties while keeping parts of it private to the owner. This limits the potential for provenance dissemination, which is a critical step in enabling data sharing amongst partners with limited a priori mutual trust. In turn, this has a negative impact on data-intensive science and its associated research publication repositories, on audit tasks, as well as on increasingly common collaborative dynamic coalitions scenarios. We propose a method for preserving privacy by creating abstractions over provenance graphs, we apply it to provenance sharing, and illustrate it on a health care case study.

About the authors

Dr. Paolo Missier is a Lecturer in Information and Knowledge Management with the School of Computing Science, Newcastle University, UK. His current research interests include models and architectures for the management and exploitation of data provenance, specifically extensions of e-infrastructures for scientific provenance. He is the PI of the EPSRC-funded project "Trusted Dynamic Coalitions", investigating provenance-based policies for information exchanges amongst partners in the presence of limited trust. Paolo contributes to two Provenance-focused Working Groups: he is a member of the W3C Working Group on Provenance on the Web, where co-edited the Provenance Conceptual Data Model; and he is co-lead of the Provenance Working Group of the NSF-funded DataONE project. He serves on a voluntary basis in both capacities. Paolo holds a Ph.D. in Computer Science from the University of Manchester, UK (2007), a M.Sc. in Computer Science from University of Houston, Tx., USA (1993) and a B.Sc. and M.Sc. in Computer Science from Universita' di Udine, Italy (1990).

Dr Roxana Danger is currently a Research Associate at the Computing Department of Imperial College London. She works on provenance APIs and data mining tools for EU-FP7 project TRANSFoRm (http://www.transformproject.eu/). She was previously at the Department of Computer Systems and Computation, Universidad Politecnica de Valencia, Spain. She developed an information extraction system for Protein-Protein interactions which combined Machine Learning and Description Logics. She obtained her PhD at University Jaume I, Castellon, Spain, with a project aimed at extracting and analysing semantic data from archaeology site excavation reports. Previously, she worked at University of Oriente in Santiago de Cuba as a full time professor.

Dr Vasa Curcin (VC) is a Research Fellow in the Social Computing Group and London e-Science Centre in the Department of Computing at Imperial College London. Dr Curcin is the Scientific Manager of the EU FP7 TRANSFoRm project, and the main author of its provenance framework. He is also the technical lead of the NIHR CLAHRC Northwest London project, and the designer of the WISH process improvement toolkit in use by over 600 users in London hospitals. He is one of the principal authors of the E-Science Discovery Net workflow system. His research interests are in the areas of process model transformations from abstract specifications and business models, to executable workflows and associated provenance traces.

Suggested keywords

PROVENANCE
PRIVACY
DATA SHARING

Preserving privacy in shared provenance data

Paolo Missier
School of Computing Science
Newcastle University, UK
Paolo.Missier@ncl.ac.uk

Jeremy Bryans
School of Computing Science
Newcastle University, UK
Jeremy.Bryans@ncl.ac.uk

Roxana Danger
Department of Computing
Imperial College, London, UK
r.danger@imperial.ac.uk

Vasa Curcin
Department of Computing
Imperial College, London, UK
vasa.curcin@imperial.ac.uk

ABSTRACT

Provenance management still lacks robust models for sharing provenance data between multiple parties while keeping parts of it private to the owner. This limits the potential for provenance dissemination, which is a critical step in enabling data sharing amongst partners with limited a priori mutual trust. In turn, this has a negative impact on data-intensive science and its associated research publication repositories, on audit tasks, as well as on increasingly common collaborative dynamic coalitions scenarios. We propose a method for preserving privacy by creating abstractions over provenance graphs, we apply it to provenance sharing, and illustrate it on a health care case study.

1. INTRODUCTION

The provenance of data is a form of structured metadata that records the activities involved in the production of the data. In addition to activities, a provenance trace may include input or intermediate data products, as well as the agents, humans as well as software systems, who were responsible for carrying out those activities. The history of the data is captured by dependencies amongst these elements. In multi-party collaborations settings that involve data sharing, as well as in third party auditing of data and processes, there is a broad expectation that provenance can be used to help data consumers form judgments regarding the reliability of the underlying data. In scientific computing, for instance, the increasing complexity of the code that is backing up research findings, together with the lack of formal software engineering processes in producing that code, is partially to blame for a large number of paper withdrawals [7] and a general *credibility crisis* [18], as evidenced for instance by major scandals such as ClimateGate [13]. Reproducibility of research is hailed as the next big challenge for computational sciences that will increase the confidence in the research process and open it up to detailed scrutiny [17]. In this setting, repeatability is underpinned by the availability of provenance, input data, and executable code. The motivating example presented below combines data sharing protocols in e-health with auditing requirements, in the context of patient selection for clinical trials.

In a different context, the notion of *dynamic coalitions* [6] has recently come to denote ad hoc collaborative partnerships that are created to pursue a common goal, in scenarios such as multi-agency emergency / threat responses. Despite the need to share data of possibly sensitive nature, these coalitions are characterized by a lack of established interaction protocols and trust amongst the partners. In this case, supplementing data exchanges with the exchange of the associated provenance metadata may, in some cases, compensate for the lack of trust.

For provenance to play a role in each of these scenarios, two main obstacles must be overcome. Firstly, full disclosure of provenance data may not be possible, e.g. due to Intellectual Property restrictions connected to individual components or data protection regulations, which introduces the need for hiding certain parts of the provenance traces. Secondly, just as data is generated and disseminated in a distributed fashion across a network of partners, so is its associated provenance, which is naturally fragmented and requires composition as it propagates along the network. The work described in this paper addresses the combination of these two common problems, namely how to compose and disseminate fragments of provenance generated by multiple partners, while enforcing privacy requirements defined by each of those partners.

1.1 Motivating scenario

Consider the example of a provenance trace of a clinical trial study design task, shown in Fig. 1. In a typical clinical research study, multiple partners collaborate on the design, by providing recruitment criteria and protocol details, including finely grained parameters such as desired patients' age, body-mass index (BMI), existing conditions and treatments, all parameterised using specialised ontologies. These eligibility queries are sent to the hospitals and GP surgeries to find suitable patients. Even the smallest variation in the query formulation may have a major impact on the number of recruited patients and determine the success or failure of the study.

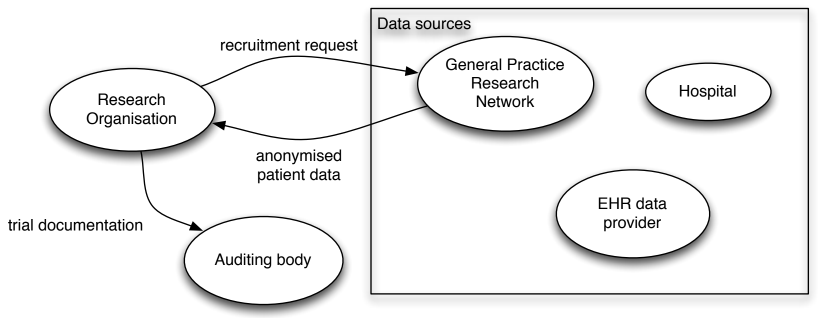


Figure 1: The *Trial Eligibility* scenario

When the request is sent from the research organization (RO) to the data source (DS), additional metadata needs to be provided to convince the recipient that the researcher has been properly authenticated, and a part of the provenance log is dispatched, abstracting over the study protocol and criteria evolution details. Once the study has been completed, various governing bodies (GB) may decide to run an audit of the process, to see if the researchers complied with the national and international regulations pertaining to trial conduct. For this purpose, GB requires RO to include details of the individual criteria and protocols that constitute the study. However, only some parts of the provenance data are passed on to the auditor, with the intermediate steps of the eligibility criteria remaining internal to the research organization. This provenance, possibly combined with other forms of knowledge that are available to GB, represents a form of evidence that GB can leverage to verify regulatory adherence.

Data provenance may also need to be composed and propagated along the network of peers. In our example, this occurs, when a DS sends the response to the RO, together with the provenance information about the database the result was extracted from, for example dates of data collection, geographic location, and socio-demographic profiles of the patient population. Part of this provenance data will need to be passed on to the GB for audit. For example, $DS1$ sends data d_1 to RO along with its provenance p_1, and RO then produces some study results d_2 based on d_1, along with its own provenance, p_2. Logically, p_1 becomes part of the provenance of d_2, because it includes the history of d_1 and d_1 was used to produce d_2. Thus, p_1 and p_2 are composed to produce an overall provenance trace that spans the lifetime of both d_2 and d_1. It is this compound provenance that RO should send to GB, if d_2 is requested as part of the audit.

1.2 Provenance trace of a clinical study

We now introduce the provenance graph example that shall be used to demonstrate our approach. The EU FP7 TRANSFoRm project (`transformproject.eu`) defines a set of modular informatics components, including provenance, data integration, and security solution, that allow interoperability between electronic health records, clinical trial data, and knowledge bases for decision support. The provenance infrastructure is model-based and uses the business process model to define abstract patterns for each client tool that are then submitted to the provenance server, and then assembled into concrete provenance graphs. Fig. 2 depicts one such provenance graph that is generated by the Query Formulation workbench, that is part of the TRANSFoRm Trial Recruitment software.

In the graph, we can see a clinical study (cS) that was created by a user who got authenticated with some security certificate. The study has associated with it a study protocol, $cSP1$, and two eligibility criteria ($cSEC2$ and $cSEC4$), both of which were created and then subsequently edited. Broadly speaking, the provenance of cS consists of all the paths in the graph in which cS appears. In this work we will rely on the rather comprehensive definition of data provenance proposed by the W3C [2], which in Sec. 2 we simplify for the purpose of our initial proof-of-concept implementation. Essentially, we view the provenance information as a graph of relations involving data, data transformation activities, and agents who are responsible for those activities and for the data.

Using the convention that the direction of the edges point “towards the past”, the graph includes two main types of relations: activity a *used* e, where e is a piece of data (an *entity*), and conversely, entity e *wasGeneratedBy* a, where a is an *activity* (a process). Thus, one can traverse the graph starting with any entity, e.g. $cSEC2$, and tracing its provenance through activities (indicated by square boxes, like $SECE1$) and their own data dependencies ($cSEC1$, cS) all the way up to the security information used by the overall process. The graph in the example also relates activities to *agents*, for example lS *wasAssociatedWith* R. Note that a dependency graph does not describe a business process, rather it captures the unfolding of one specific instance of such a process. The process specification is not captured in the trace.

1.3 Assumptions and contributions

In the scenario just described, data producers and consumers pursue potentially conflicting goals. On one side, consumers have an interest to acquire rich provenance that they can query and mine. In contrast, producers may not be prepared to disclose all details of its internal processes, as we have seen in our example. The resulting tension must be resolved if useful information exchange is to take place. For producers, this translates into a goal of maximizing the amount

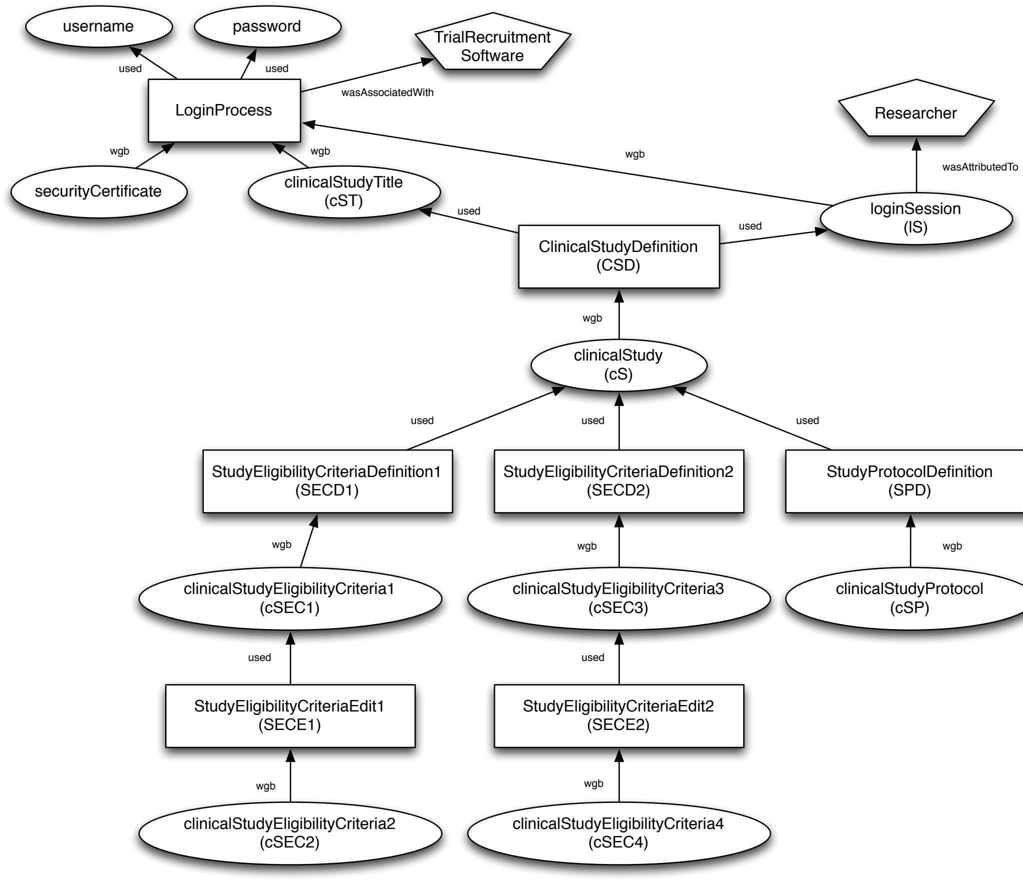

username
password
TrialRecruitment Software
used
used
wasAssociatedWith
LoginProcess
Researcher
wgb
wgb
wgb
wasAttributedTo
securityCertificate
clinicalStudyTitle (cST)
used
loginSession (lS)
used
ClinicalStudyDefinition (CSD)
wgb
clinicalStudy (cS)
used
used
used
StudyEligibilityCriteriaDefinition1 (SECD1)
StudyEligibilityCriteriaDefinition2 (SECD2)
StudyProtocolDefinition (SPD)
wgb
wgb
wgb
clinicalStudyEligibilityCriteria1 (cSEC1)
clinicalStudyEligibilityCriteria3 (cSEC3)
clinicalStudyProtocol (cSP)
used
used
StudyEligibilityCriteriaEdit1 (SECE1)
StudyEligibilityCriteriaEdit2 (SECE2)
wgb
wgb
clinicalStudyEligibilityCriteria2 (cSEC2)
clinicalStudyEligibilityCriteria4 (cSEC4)

Figure 2: Example provenance graph for defining a clinical study in TRANSFoRm

of provenance disclosed to consumers, consistent with their privacy constraints. In this work we provide data producers with an algorithm that lets them perform this trade-off by computing a new provenance graph where sensitive information is replaced by some abstraction. The specification of sensitve information is captured in the form of a privacy policy, consisting of a set of nodes which are not to be disclosed, which is the input to the algorithm. We assume that none of the parties acts maliciously, and that the provenance associated with the data is genuine, i.e., it has not been tampered with. We show how abstracted views of dependency graphs can be computed in a way that preserves properties of the dependency relations, and how the composition of provenance graphs from different producers can itself be abstracted and delivered to a consumer in a way that complies with each of those producers' different policies relative to that consumer.

Concretely, the paper offers the following technical contributions. In Sec. 2, we introduce a definition of abstraction over dependency graphs that preserves the consistency of the graph, defined by certain properties of the dependency relations. We then define a graph abstraction function that takes a user's policy and edits an input provenance graph, resulting in an abstracted view of the graph. This is the algorithm used by a content producer such as DS. We then describe (Sec. 2.5) an early implementation using VDM [1, 14], which is compatible with a core subset of the W3C PROV provenance model [2]. The simple privacy policy used here, consisting of just a set of nodes, provides a foundation for experimenting with more expressive policy languages, which will be needed to make the algorithm useful in practice. Extensions to the policy language are left for future work, however.

In Sec. 3 we address the problem of composing multiple provenance graphs in combination with multiple abstractions, and multi-party provenance propagation. Each partner may receive data from other partners, and use it to generate more data and thus more provenance. It combines the latter with provenance of the data it has received, and shares the result with other partners, using policies of the form introduced in the first part of the paper. We assume that each producer may specify a different policy for each of the partners that its provenance may be distributed to. The abstraction function is used to compute different abstract views over dependency graphs according to the different producer policies.

1.4 Related work

Several strands of research are relevant to our work. The first, largely motivated by the need to preserve privacy in social network data, extends the well-known data anyonomization framework developed for relational data, to graph data structures [19, 3, 15]. Here the main problem, also summarised in a 2008 survey [20], is to ensure that various forms of anonymization are robust to attacks from adversaries who can potentially leverage their partial information about fragments of the graph, to infer additional knowledge. Although in our setting the problem of "de-abstracting" provenance graphs is not our primary concern, this body of research will indeed be a useful reference for future work. Closer to us is a strand of research that investigates the problem of preserving the privacy of functions used in workflows, when a large number of input/output pairs for those functions is revealed through the provenance traces of multiple executions of the workflow. While this work on *module privacy* [11, 10, 9] applies anonymization techniques specifically to provenance graphs, it is centred around a workflow-specific form of provenance and is concerned with protecting the semantics of workflow modules, and thus is still peripheral to our interest.

Also centred on workflow provenance is the Zoom system [4] for computing views over provenance graphs that contain traces of dataflow execution (a "run"). Views, which are effectively a form of abstraction, are computed based on the user's indication of which workflow modules (tasks) are relevant, or perhaps based on which modules a user is allowed to see. This is similar to our work in two ways. Firstly, a view is an answer to a provenance query, which accounts for users preferences and privileges. While provenance sharing policies are not a part of the model and thus are not mentioned explicitly, it is easy to imagine our policies providing input to the view generator. Secondly, Zoom too has a notion of consistency, i.e., views must be valid provenance graphs. The main difference with our work is that in Zoom, provenance views are really a by-product of user views over the workflow whose execution the provenance graph represents, whereas our approach is based on a "PROV-lite" provenance model that makes no assumptions on the structure of the process that generates the graph.

Closer to our abstraction model, both in motivation and in its technical approach, is the ProPub system [12], which computes views over provenance graphs that are suitable for publication by meeting certain privacy requirements. In ProPub, users specify edit operations on a graph, such as anonymizing, abstracting, and hiding certain parts of it (here the term "abstracting" is interpreted as "zooming out", much as in [4]). Such operations are specified as logic rules, and are interpreted natively by the Datalog-based prototype implementation. ProPub adopts an "apply–detect–repair" approach, whereby user rules are applied to the graph first, then consistency violations that may occur in the resulting new graph are detected, and a final set of edits are forced on the graph to try and repair such violations. In some cases, this causes nodes that the user wanted removed to be reintroduced, and it is not always possible to satisfy all rules. In contrast, our user policies are more simply a set of nodes to be abstracted (but note that anonymizing and hiding are a particular case of those). However, in return for this simplicity, our algorithm is guaranteed to always produce a valid abstract graph while ensuring that the nodes specified in the policy are removed. This is achieved primarily by augmenting such policy nodes with new nodes, as described in detail in the next section. In order to further clarify the relationship between the two approaches, in Appendix A we show how our algorithm computes an abstraction over the same example graph used in the ProPub paper.

2. ABSTRACTING PROVENANCE GRAPHS

We concentrate on the role of the data and provenance producer, and make concrete the types of sharing policies that a producer applies to each consumer (we use the term *partner* to denote either producers or consumers). Each partner maintains a set of policies, one for each partner to which it sends information. The sharing policy that applies to messages sent by partner X to Y, denoted $pol(X, Y)$, is a set of graph nodes $\{n_1 \dots n_k\}$ which X wants to remove from the graph before transmitting it to Y. In this section we focus on two partners. This is later extended in Sec. 3 to provenance graphs exchanged across mutiple parties. We present an abstraction algorithm that creates a valid new provenance graph in which the desired nodes are abstracted into a single node. In order to preserve the semantics of the provenance graph, other nodes may have been included as well, but we ensure that none of the nodes in the policy set are ever left in the graph. This is consistent with the general goal of maximising the amount of provenance that is shared, consistent with the producer's sharing policy.

2.1 Definitions

We begin with a definition of provenance graphs, which we restrict to a small subset of the node-types and relations available in the full provenance language [2].

DEFINITION 1 (PROVENANCE GRAPH). *A Provenance Graph is a Directed Acyclic Graph $(Vert, Edg)$ such that $Vert = E \cup A$, where E and A are finite disjoint sets of entites (data) and activities, and $Edg = U \cup G$ where $U \subseteq (A \times E)$ and $G \subseteq (E \times A)$.*

We will use e and a to refer to elements of E (entities) and A (activities) respectively, and we will use n (for *node*) to refer to an element of $E \cup A$. The intuition behind Definition 1 is as follows: E is the set of data nodes, and A is the set of activity nodes. U is the *used* relation: if $(a, e) \in U$ then we say that *a used e* (or that e is an input to activity a.) G is the *was-generated-by* relation: if $(e, a) \in G$ then we say that *e was generated by a* (or that e is an output from activity a.) The restriction to Directed Acyclic Graphs in Def. 1 reflects the common intuition made of provenance graphs, that there are no cyclic dependencies amongst causal relations.[1]

DEFINITION 2 (PRECEDENCE AND PATHS). *For any provenance graph, if $(n_2, n_1) \in Edg$, we say that node n_1* directly precedes *node n_2. If $(n_2, n_1) \in Edg^+$, we say that node n_1* indirectly precedes *node n_2. An alternative is the infix notation: $n_1 < n_2$. If for all $i \in 1..m{-}1$ it is the case that $(n_i, n_{i+1}) \in Edg$, then $(n_1, \dots n_m)$ forms a* path.

[1]Note however that more expressive provenance graphs expressible in the PROV model [2], which include relations which are not dependencies amongst entities and activities, may in some cases contain cycles. This work is not concerned with non-causal provenance relations between entities and activities.

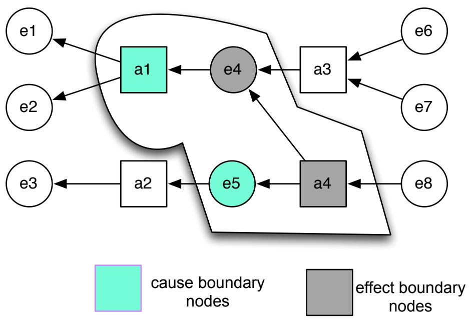


Figure 3: Cause and effect boundary nodes

These dependencies capture temporal order: if $n_1 < n_2$ then n_1 was created earlier (in the case of an entity) or executed earlier (in the case of an activity) in time than n_2[2]. A *path* travels *forwards* in time, from cause to effect.

The *cause boundary nodes* of a set of nodes are the points at which a path through the provenance graph enters the set. The *effect boundary nodes* are the points at which a path through the graph leaves the set.

Definition 3 (Cause boundary nodes). *If $P \subseteq Vert$ is a set of nodes in a provenance graph, then $n \in P$ is a* cause boundary node *provided there exists a node n' such that $n' \notin P \wedge (n, n') \in Edg$. (n, n') is referred to as an in-edge of P.*

Definition 4 (Effect boundary nodes). *If $P \subseteq Vert$ is a set of nodes in a provenance graph, then $n \in P$ is an* effect boundary node *provided there exists a node n' such that $n' \notin P \wedge (n', n) \in Edg$. (n', n) is referred to as an out-edge of P.*

Cause and effect boundary nodes are identified from the point of view of the creation of the provenance trail, and define the points at which paths may enter and leave a set of nodes. For example, the provenance graph in Fig. 3 has been created from left to right. Within the set of nodes $\{a1, e4, e5, a4\}$ the cause boundary nodes are $\{a1, e5\}$ and the effect boundary nodes are $\{e4, a4\}$. Cause boundary nodes and effect boundary nodes are referred to as *boundary nodes*.

An *unbroken* set is a set of nodes in which no effect boundary node precedes a cause boundary node.

Definition 5 (Unbroken sets). *A set of nodes N is* unbroken *provided for all cause boundary nodes $n_i \in N$, there does not exist an effect boundary node $n_o \in N$ such that $n_o < n_i$.*

[2] This is a slight simplification over the PROV model cited earlier, where usage and generation are *events* and a precedence relation is defined over events, rather than over time, to account for possibly inconsistent timestamps across multiple provenance generators. Such complete definition of the precedence relation can be found in [8].

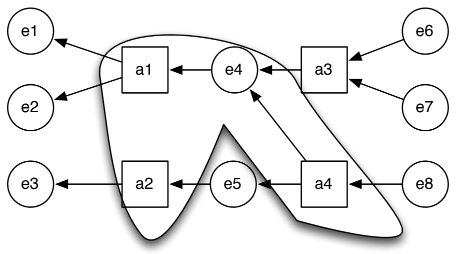


Figure 4: A broken set

In Fig 4, a broken set $\{a1, a2, e4, e3\}$ is identified. Within this set, $a1$ is a cause boundary node and $e4$ is an effect boundary node. Node $a2$ is a cause (because of the relationship $(a2, e3)$,) and an effect boundary node (because of the relationship $(e5, a2)$). Similarly $a4$ is both a cause and an effect boundary node. The set is broken because $a2$ (an effect boundary node) precedes $a4$ (a cause boundary node.) The set depicted in Fig. 3 is an unbroken set.

Definition 6 (a-chunk). *If N is a set of nodes in a provenance graph and N is an unbroken set, then N is an* a-chunk *provided all boundary nodes in N are a-nodes.*

Definition 7 (e-chunk). *If N is a set of nodes in a provenance graph and N is an unbroken set, then N is an* e-chunk *provided all boundary nodes in N are e-nodes.*

We now define a simple abstraction function on provenance graphs. When an abstraction operation is performed on a graph, we use the term *concrete* to refer to the graph before the operation, and *abstract* to refer to the graph after abstraction.

Definition 8 (Step-Abstraction). *For any provenance graph $(Vert, Edg)$, a* step-abstraction *is a provenance graph $(Vert_a, Edg_a)$ in which either a single a-chunk (resp. e-chunk) N is replaced by an abstract a-node (resp. e-node) n, and (i) for all $n_1, n_2 \in N$, (n_1, n_2) is removed from Edg; (ii) for all $(n_1, n_2) \in Edg$ where $n_1 \in N$ and $n_2 \notin N$, (n_1, n_2) is replaced with (n_a, n_2), and (iii) for all $(n_1, n_2) \in Edg$ where $n_1 \notin N$ and $n_2 \in N$, (n_1, n_2) is replaced with (n_1, n_a).*

In the above definition, (*i*) ensures that each edge wholly contained within N is removed altogether, and (*ii*) and (*iii*) that links between nodes inside and outside of N continue to be recorded, with the appropriate change of name. An algorithm (Alg. 1.) to perform a single step-abstraction is given. An abstraction is formed as a sequence of step-abstractions.

Definition 9 (Abstraction). *An* abstraction *of $(Vert, Edg)$ is a new DAG $(Vert_a, Edg_a)$ together with a total abstraction function $Abs : Vert \rightarrow Vert_a$, and its inverse relation $Con \subseteq Vert_a \times Vert$ such that (i) for all n_a in $Vert_a$, $Con(n_a)$ is an a-chunk or an e-chunk, if n_a is an* *abstract node, or a singleton set if not; (ii)* $(n_a, n'_a) \in Edg_a$ *provided* $n_a \neq n'_a$ *and there exists* $(n, n') \in Edg$ *such that* $Abs(n) = n_a$ *and* $Abs(n') = n'_a$ *(iii)* Edg_a^+ *is acyclic.*

The intuition behind abstraction is as follows: (i) Each newly created node n_a is formed either by abstracting an a-chunk or an e-chunk. The concrete set of abstracted nodes is represented by the set $Con(n_a)$. (ii) The abstracted relations lose only the links covered completely by an element of the partition: other links are renamed by the *Abs* function. (iii) There are no cycles. The abstraction does not break the usual precedence rule associated with provenance graphs [3].

When abstracting nodes, it is possible to create a *false dependency*: i.e. to create a path in the abstract graph that does not have a counterpart in the concrete graph. This has an implication for the receiver, who may then make inferences of the abstracted graph which are not justified by the original data.

DEFINITION 10. *A* false dependency *appears in an abstraction if there exists* $(n_1, n_2) \in Edg_a^+$ *such that* $(n_1, n_2) \notin Edg^+$.

THEOREM 1. *Checking that a step abstraction introduces no false dependencies is equivalent to checking that for the abstract node* n_a *created in the step, the chunk formed by* $Con(n_a)$ *is* causally total*, where a chunk is* causally-total *provided for all cause boundary nodes* n_c *and effect boundary nodes* n_e *it is the case that all paths* $nc \rightarrow \cdots \rightarrow ne$ *are in the concrete graph.*

PROOF. Consider a graph $(Vert, Edg)$, and an abstracted counterpart $(Vert_a, Edg_a)$ that contains a path $(n_1 \ldots n_{abs} \ldots n_2) \in Edg_a^+$. Assume $(n_1, n_2) \notin Edg^+$ (i.e. that $(n_1 \ldots n_{abs} \ldots n_n)$ is a false dependency.) There must exist concrete nodes $n_c, n_e \in Con(n_{abs})$, and so the paths (n_1, n_c) and (n_e, n_2) must exist in the concrete graph (since (n_1, n_{abs}) and (n_{abs}, n_2) exist in the abstract graph.) If there is no path $(n_1 \ldots n_2)$ in the graph, then (since (n_1, n_c) and (n_e, n_2) are both in Edg^+), (n_c, n_e) must not be in Edg^+, and so $Con(n_{abs})$ is not causally total.

If, on the other hand, if $Con(n_{abs})$ is causally total, then $(n_c, n_e) \in Edg^+$, and so (by a similar argument to above) $(n_1, n_2) \in Edg^+$. $\square$

2.2 Abstraction at work

The declarative definition of step abstraction (Def. 8) has an operational counterpart in operator *abs*:

$$abs(pg, Remove) \tag{1}$$

which takes a provenance graph *pg* and a set *Remove* of nodes, and computes a new abstracted, valid graph in which a superset of *Remove* is replaced by an abstract node graph, as described above. The pseudo-code for the abstraction operator is shown in Alg. 1. The algorithm consists of two phases. In the first phase, the *Remove* set is extended

[3]But please note footnote 2.

Algorithm 1 Given a Provenance Graph *PG* and set of nodes $\{n_1, \ldots n_n\}$ to combine into a single node, computes the smallest superset of these of nodes which when hidden will result in a sound graph. These are removed and replaced by n_{abs} which has type *a-node*.

```
1: $Remove := \{n_1, \ldots n_n\}$
2: repeat
3:     $Remove' := Remove$
4:     $Remove :=$ EXTEND_TO_A-NODES($Remove$)
5:     $In :=$ CAUSE_SUBSET($Remove$)
6:     $Out :=$ EFFECT_SUBSET($Remove$)
7:     $Remove := \{n | n_i \leq n \leq n_o . n_i \in In \wedge n_o \in Out\}$
8: until $Remove = Remove'$
9: $Vert := Vert \cup n_{abs}$
10: for all $n \in In$ do
11:     for all $n' \notin Remove \bullet used(n, n') \in PG.used$ do
12:         replace $used(n, n')$ with $used(n_{abs}, n')$ in $PG.used$
13:     end for
14: end for
15: for all $n \in Out$ do
16:     for all $n' \notin Remove \bullet wgb(n', n) \in PG.wgb$ do
17:         replace $wgb(n', n)$ with $wgb(n', n_{abs})$ in $PG.wgb$
18:     end for
19: end for
20: $Vert := Vert \setminus Remove$
```

until an a-chunk is obtained[4]. This involves the following steps. Firstly, the a-nodes related to the boundary nodes of *Remove* (line 4) are added to *Remove*. Secondly, the causal and effect boundary nodes for this new set are identified (lines 5 and 6). Finally (line 7), the nodes in the path between any pair of causal and effect boundary nodes are also added. In the second phase, all nodes in *Remove* are replaced with a new, abstract node, and the surrounding nodes are properly connected to it using the correct relation types.

We illustrate the algorithm at work on the provenance graph of Fig. 2, illustrated in Sec. 1.1. Suppose that the graph is generated by the Research Organisation (RO) which sends it to the Data Source (DS), subject to sharing policy $pol(\mathrm{RO}, \mathrm{DS}) = R_1$ where $R_1 = \{CSD, SECD1\}$. Thus, initially $Remove = Remove' = \{CSD, SECD1\}$. EXTEND_TO_A-NODES($CSD, SECD1$) does not change the set, since both are already a-nodes. In lines 5 and 6, the cause and effect boundary subsets of this set are calculated, which are also both $\{CSD, SECD1\}$. The intersection of their precedents and consequents consists of the nodes themselves, together with any nodes which lie on a path between them. In this case the node *cS* is added, so the new set is $\{CSD, cS, SECD1\}$. Since this is not the same as *Remove'* the execution returns to line 2 with the set *Remove* now the set $\{CSD, cS, SECD1\}$.

On the second traversal the extension to a-nodes propagates the set to $\{CSD, cS, SECD1, SECD2, SPD\}$. The effect boundary nodes are $\{SECD1, SECD2, SPD\}$ and the cause boundary node is $\{CSD\}$. The intersection of their precedents and consequents is equal to the *Remove* set, and so the new abstract node *ABS* is added to the set (line 9). Lines 10-14 take all the cause boundary nodes (only *CSD* in this case) and replace the relations $used(CSD, cST)$

[4]The algorithm described here considers a-chunk expansions but it can be easily changed to compute e-chunk expansions.

Algorithm 2 EFFECT_SUBSET; Input: a set of nodes N; Output: a set of nodes $M \subseteq N$ such that M is the subset of N which are effect boundary nodes. Intent: to retain only the effect boundary nodes wrt the DAG ordering in a set. The effect boundary nodes are later in time than the cause boundary nodes

```
1: $M := \emptyset$
2: for all $n \in N$ do
3:     if $\exists n'.n' \notin N \wedge (n, n') \in Edg$ then
4:         $M := M \cup \{n\}$
5:     end if
6: end for
7: output $M$
```

Algorithm 3 CAUSE_SUBSET; Input: a set of nodes N; Output: a set of nodes $M \subseteq N$ such that M is the subset of N which are cause boundary nodes. Intent: to retain only the cause boundary nodes wrt the DAG ordering in a set

```
1: $M := \emptyset$
2: for all $n \in N$ do
3:     if $\exists n'.n' \notin N \wedge (n', n) \in Edg$ then
4:         $M := M \cup \{n\}$
5:     end if
6: end for
7: output $M$
```

and $used(CSD, lS)$ with $used(ABS, cST)$ and $used(ABS, lS)$ respectively. Lines 15-19 go through the same process for the effect boundary nodes, this time replacing the relations $wgb(SECD1, cSEC1)$, $wgb(SECD2, cSEC3)$ and $wgb(SPD, cSP)$ with $wgb(ABS, cSEC1)$, $wgb(ABS, cSEC3)$ and $wgb(ABS, cSP)$ respectively. Finally (line 20) the set *Remove* $(\{CSD, cS, SECD1, SECD2, SPD\})$ is removed. The result is shown in Fig. 5.

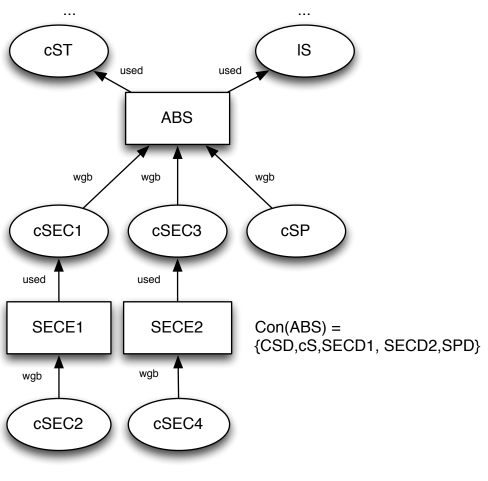


Figure 5: Result of abstracting over $R_1 = \{CSD, SECD1\}$

2.3 Introducing Agents

We have developed the core of our work for the two primary types in a provenance graph – entities and activities. Extending this scope to take account of the richness of the PROV language is part of our future work. In this section we indicate how such extensions can be managed, by extending the definition of dependency graphs to include agents and three important agent-based relations: attribution, association and delegation. Entities may be *attributed to* agents, activities may be *associated with* agents and agents may *delegate to* other agents. These relations are all included in Defn 11, which is an extension of Defn. 1.

Algorithm 4 EXTEND_TO_A-NODES: Input a set of nodes N. Output: The minimal extension of N to include a-nodes at the boundary, except where the boundary the extension is also the boundary of the underlying dependency graph.

```
 1: for all $n \in$ EFFECT_SUBSET($N$) do
 2:     if $n \in$ e-nodes then
 3:         $N := N \cup \{a|wgb(n, a)\}$
 4:     end if
 5: end for
 6: for all $n \in$ CAUSE_SUBSET($N$) do
 7:     if $n \in$ e-nodes then
 8:         $N := N \cup \{a|used(a, n)\}$
 9:     end if
10: end for
```

DEFINITION 11 (EXTENDED PROVENANCE GRAPH). *An Extended Provenance Graph is DAG* $(Vert, Edg)$ *such that* $Vert = E \cup A \cup Ag$, *where* E, A *and* Ag *are finite disjoint sets of data, activities and agents respectively;* $Edg = U \cup G \cup Atr \cup Ass \cup Del$ *where* $U \subseteq (A \times E)$ *and* $G \subseteq (E \times A)$, $Atr \subseteq (Ag \times E)$, $Ass \subseteq (Ag \times A)$ *and* $Del \subseteq (Ag \times Ag)$.

$Del(ag1, ag2)$ means agent *ag2* delegated to *ag1*; the other relations are unambiguous. The abstraction algorithm may be extended to incorporate agents by allowing the sender to identify agents to be removed. The extended procedure is given in Alg. 5 and works as follows:

1. Ignoring agents and their relations, perform algorithm 1 on the concrete graph. A new node n_{abs} will be created.
2. Remove agents which have been identified for removal, and the relevant relations. (Alg. 5 lines 1-6.)
3. For any remaining agent ag and removed node n, replace all relations $atr(ag, n)$ and and $ass(ag, n)$ with $atr(ag, n_{abs})$ and $ass(ag, n_{abs})$ respectively. (Alg. 5 lines 7-12.)
4. Remove orphan agents: remove all agents which do not appear in $rng(del)$ or $dom(atr)$ or $dom(ass)$. (Alg. 5 lines 13-22.)

If an agent is removed from the middle of a delegation chain, the agents higher up the chain may be *orphaned.* We have chosen to explicitly break delegation chains, and so these orphan agents are removed. Another straightforward possibility for encoding is that they are joined to their nearest successor.

2.4 Composing multiple step abstractions

Observe that, since the *abs* operator (1) generates a provenance graph, it should be possible to functionally compose *abs* with itself, i.e., by computing

$$abs(abs(pg, R_1), R_2) \qquad (2)$$

Algorithm 5 REMOVING A CHOSEN SET OF AGENTS: Input: An abstracted provenance graph and the name of the abstracted node n_{abs}, a set of agents AG to remove, and the set $Remove$ of names of nodes that have been removed by Alg. 1. Output: The abstracted provenance graph, with agents AG removed together with their relationships.

```
1: for all $ag \in AG$ do
2:     for all $n \in Vert$ do
3:         remove $\{atr(ag, n), ass(ag, n), del(ag, n)\}$
4:     end for
5: end for
6: remove $AG$ from $Vert$
7: for all $ag \in Vert$ do
8:     for all $n \in Remove$ do
9:         replace $atr(ag, n)$ with $atr(ag, n_{abs})$ in $PG.atr$
10:        replace $ass(ag, n)$ with $ass(ag, n_{abs})$ in $PG.ass$
11:    end for
12: end for
13: $orphans := \emptyset$
14: repeat
15:    $orphans' := orphans$
16:    for all $ag \in Vert$ do
17:        if $ag \notin rng(PG.del) \wedge ag \notin dom(PG.ass) \wedge ag \notin dom(PG.atr)$ then
18:            $orphans := orphans \cup \{ag\}$
19:        end if
20:    end for
21: until $orphans = orphans'$
22: $Vert := Vert \setminus orphans$
```

for two policy sets, R_1, R_2. This is important as it provides users with finely grained control over abstraction. To see this, observe that abs is designed to work best when the nodes to be abstracted out are close to each other in the graph, as nodes that are far apart will result in many additional intermediate nodes to be coalesced in the process[5] ("coarsely grained" abstraction). This is not always the intended effect of abstraction, however. Suppose for instance, continuing with the example from Fig. 5 where the set of nodes in $R_1 = \{CSD, SECD1\}$ was abstracted, that the user wants to also abstract the set $R_2 = \{SECE2, cSEC4\}$. If the union of these nodes was removed from the original graph, this would result in a large abstraction that would include the node $cSEC3$, amongst others.

In contrast, consider the effect of sequential composition. The first step results in the graph $pg' = abs(pg, R_1)$ shown in Fig. 5, where the single abstract node ABS is abstracting over $Con(ABS) = \{CSD, cS, SECD1, SECD2, SPD\}$. One can then apply R_2 to pg' to compute $abs(pg', R_2)$ shown in Fig. 6, with the second abstract node ABS' added. Applying a single abstraction over $R_1 \cup R_2$ would have led to node $cSEC3$ being abstracted out, while the composition of separate abstractions on R_1 and R_2 results in a graph that still contains $cSEC3$. Since there is no overlap between extensions of R_1 and R_2, in this case the two abstractions are independent of one another. This is not always the case, however. For instance, suppose again that $pg' = abs(pg, R_1)$, but now the second policy set to be

[5]In the worst case, abstracting the input and output nodes of a computation simply results in one comprehensive node that summarizes the entire provenance history. On the other hand, if only a single activity node is abstracted, the effect is simply to rename (anonymize) that node, or to remove the value of some of its properties.

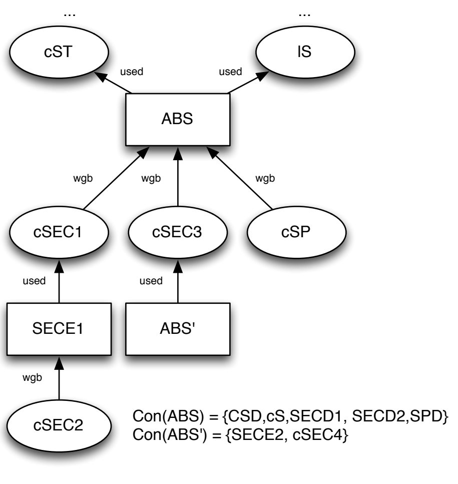


Figure 6: Applying $R_2 = \{SECE2, cSEC4\}$ after R_1

applied is $R_3 = \{SECD2, SECE2\}$. Since $SECD2$ is no longer part of pg', having been abstracted out in the previous step, one cannot simply compute $abs(pg', R_3)$.

The strategy we propose is to replace the node in the policy restriction with the node that abstracts it in the new graph. So, if $SECD2$ is replaced with ABS in R_3, the new policy set $R_3' = \{ABS, SECE2\}$ can be applied to pg', resulting in

$$pg'' = abs(pg, R_3')$$

which contains the new abstract node ABS' abstracting over $Con(ABS') = \{ABS, cSEC3, SECE2\}$, as shown in Fig.7.

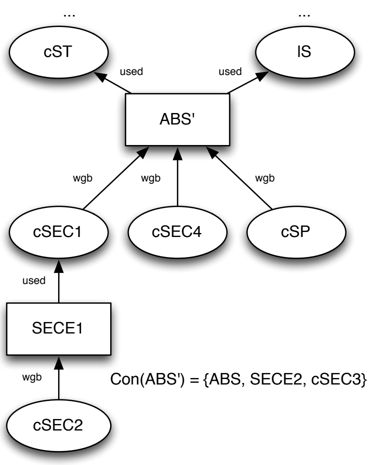


Figure 7: Applying $R_3 = \{SECD2, SECE2\}$ after R_1

These considerations suggest that we can simply use the mapping $Abs : Vert \rightarrow Vert_a$ defined as part of the abstraction process (Def. (9)), to replace each node mentioned in a policy such as R_3, with its corresponding abstract node produced by the previous application of the abs operator. Let Abs_M be the Abs function created from an application of abstraction using policy set M. The replacement function $\rho(.)$ for a policy set R is a comprehension:

$$\rho(R, Abs_M) = \{Abs_M(n) | n \in R\}$$

for any policy M (recall from Def. (9) that $Abs_M(n) = n$ for any non-abstract node n).

In our running example, we have $Abs_{R_1}(n) = ABS$ for $n \in \{CSD, cS, SECD1, SECD2, SPD\}$, and $Abs_{R_1}(n) = n$ for all other nodes. Thus, $\rho(\{SECD2, SECE2\}, Abs_{R_1}) = \{ABS, SECE2\}$. This is the new policy set that should be used for the second application of Abs.

Using $\rho(.)$, we define a new *composable* version $abs_C(pg, R, Abs_M)$ of abs, as:

$$abs_C(pg, R, Abs) = abs(pg, \rho(R, Abs)) \qquad (3)$$

In our example, $abs_C(abs_C(pg, R_1, I), R_3, Abs_{R_1})$ produces the graph in Fig. 7 (I is the identity function, used for the first application of abs to pg). Note that abs_C reduces to abs when applied to non-abstract graphs. As a final note, observe that in our example we have

$$abs(abs(pg, R_1, I), R_2, Abs_{R_1}) = abs_C(abs_C(pg, R_2, I), R_1, Abs_{R_2}).$$

We conjecture, without proof, that abs_C is indeed commutative.

The ability to compose abstractions provides users with a choice, between an abstraction semantics where a policy set is applied as a whole, and one where different partitions of the policy set are applied in sequence, using composition. The former leads to potentially large subsets of the nodes being abstracted out, while the second results in more finely grained control over the set of nodes that are abstracted out. The case of multiple sets of nodes arises naturally when a graph is abstracted according to multiple, independent sharing policies (each possibly consisting of a single set). This scenario is discussed in the next section.

2.5 Prototype implementation

We have encoded the PROV subset discussed as a model within the Vienna Development Method (VDM) specification language [1, 14]. The algorithms are encoded functionally using an executable subset of the modelling language, allowing models of provenance to be readily analysed using the Overture interpreter. VDM benefits from strong tool support and has been used similarly to its use here in the modelling and analysis of access control policies [5]. The open source tool set (Overture[6]) includes a syntax and type checker, an interpreter for executable models, test scripting and coverage analysis facilities, program code generation and pretty-printing. These have the potential to form a platform for tools specifically tailored to the analysis of provenance policies in a PROV-compliant framework. Advanced static analysis for VDM models includes automatic proof obligation generation and automated proof support is under development. The development of automated translation from PROV-N (the relational syntax used by PROV) to VDM models and vice versa is under way.

3. PROVENANCE COMPOSITION AND QUERY

[6]`http://overturetool.org`

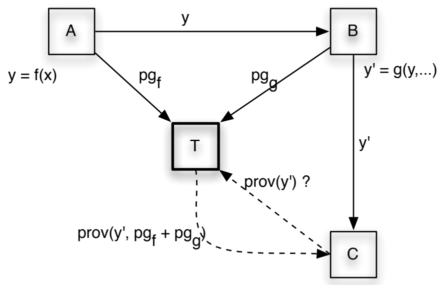


Figure 8: Multi-party provenance exchanges with trusted partner

The provenance abstraction algorithm described in the previous section defines an operator on provenance graphs, that can be used to enforce a sender privacy policy which regulates the exchange of provenance between two parties. In this section we build on this work and discuss a typical usage of abstraction. We outline a model for propagating provenance across a network of partners, under the assumption that *each partner may define a different sender privacy policy relative to each other partner.* We argue for the need for dependency graph composition, we define a simple composition operator that is adequate for our provenance model, and analyse the interplay of such an operator with the abstraction operator.

3.1 Multi-party interaction pattern

Consider the stereotypical interaction pattern of Fig. 8, in which three partners, A, B, and C can send and receive data values and can generate provenance graphs corresponding to the computation of those values. A distinguished mutually trusted partner, T (this may be one of the senders/receivers) is in charge or managing the graphs on behalf of the entire coalition, and of ensuring that each partner will only see views over the provenance graphs, which are consistent with the policies.

In this pattern, we represent data production at A and B using the two functions f and g, respectively, where $y' = g(y, \dots)$ denotes that the new data value y' depends in part on y. Initially, A computes $y = f(x)$ and sends y to B, then B uses y (along with other local data) to compute $y' = g(y, \dots)$. Let pg_f and pg_g denote the provenance graphs associated with the computation of $y = f(x)$ and $y' = g(y)$, respectively, and assume that A and B send their respective graphs to T. At some later time, B sends y' to C. C can then issue a query to T asking for the provenance $prov(y')$ of y'.

As we have assumed throughout this work, the disclosure of provenance information by A and B to C is regulated by privacy policies, which are defined independently for each pair of partners. Let $pol(X, Y)$ denote the privacy policy that defines the abstraction of X's graph that Y is allowed to see. Recall that $abs_C(pg, pol(X, Y))$ denotes the result of abstracting out graph pg according to policy $pol(X, Y)$. We assume that T has unrestricted access to all provenance graphs produced by each of the other partners, and that it has full knowledge of $pol(X, Y)$ for each pair X, Y. In order to respond to a provenance query, such as the one from C, T must (i) *compose* all provenance graphs that may be relevant to the query (pg_f and pg_g), and (ii) compute an abstraction that conforms with all the policies, i.e., $pol(A, C)$ and $pol(B, C)$ in the example[7].

3.2 Composition and abstraction

Consider the two graphs pg_f and pg_g introduced earlier. In this simple example, we expect such graphs (Fig. 8) to be $pg_y = (\{y, g, y'\}, \{u(g, y), g(y', g)\})$, and similarly for pg_f[8]. Taken on its own, $prov(y')$ consists of all the paths in pg_y that originate in node y', in this case only path $y' \to g \to y$. In turn, by the same construction $prov(y)$ contains path $y \to f \to x$. Clearly, the two paths can be combined to form a more complete path that is arguably part of the provenance of y', namely $y' \to g \to y \to f \to x$ (Fig. 9(a)). The composition operator:

$$pg_1 \oplus pg_2$$

combines two provenance graphs pg_1, pg_2 so that such composite provenance can be computed. The simple definition of composition given below takes the union of the nodes and edges of the two component graphs, and will suffice for our purposes, with the exception discussed below.

Definition 12 (Provenance Graph Composition). *Let $g_1 = (Vert_1, Edg_1)$, $g_2 = (Vert_2, Edg_2)$ be two provenance graphs. The compound graph $g = g_1 \oplus g_2$ is $g = (Vert_1 \cup Vert_2, Edg_1 \cup Edg_2)$.*

As a side note, observe that this definition assumes that all partners agree on using a common naming scheme for the data they exchange. For example, pg_f and pg_g correctly "join" on y only if the producers of each of the two graphs agree to use the same identifier for y. If each partner follows a local naming scheme, as is commonly the case, then their provenance graphs will reflect that scheme, and mappings from local to shared identifiers are required. Here we assume that suitable mappings from the local to the shared naming scheme are applied prior to provenance graphs being sent to T, so that the identifier used to send data, such as y, matches the references to y that appear in pg_f. We refer to prior work [16] for an in-depth analysis that shows the need for such mappings in the specific scenario of data sharing through a repository.

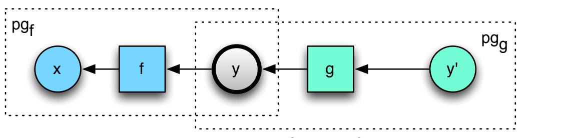
Consider the set of *join nodes* $V_J = Vert_1 \cap Vert_2$. When $V_J = \emptyset$, composition simply results in two disconnected graph components, and no new paths appear in the provenance of the final data. Fig. 9(b) shows a more interesting example where g makes use of one of y's inputs:

$$y = f(x_1, x_2), y' = g(y, x_2)$$

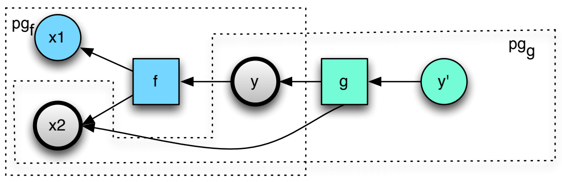
and $V_J = \{x_2, y\}$. In this example the composition shows, correctly, that x_2 was used twice, once by f and once by g. In other cases, composition may result in a new graph that is intuitively inconsistent. Suppose for instance that partner A computes two values: $\langle y_1, y_2\rangle = f(x)$, then it sends y_1 to B who computes $y_2 = g(y_1)$. Now $V_J = \{y_1, y_2\}$, but the compound graph shows y_2 to have been generated twice. In this example, this is indicative of a problem between A and B, who each claim to have generated y_2. While explaining such "collisions" is out of the scope of our work, it is important that they be detected, i.e., by introducing consistency constraints on the graphs. Indeed, a number of such constraints are defined formally as part of the PROV family of specifications [8]. Consistent with our limited definition of dependency graphs, which simply considers the *usage* and *generation* relations amongst graph nodes, we introduce one single constraint that is relevant for these relations, namely that *each entity shall be generated by at most one activity* (this is essentially Constraint 25 in [8]). This translates into the condition that any entity node be the source of at most one *generation* edge, that is, a graph $dg = (Vert, Edg)$ where $e, a_1, a_2 \in Vert$, $g(e, a_1), g(e, a_2) \in Edg$ is inconsistent if $a_1 \neq a_2$ (Fig. 9(c)). We will therefore assume that graphs are composable, i.e., for every $e \in V_J$, and for any $a_1 \in Vert_1, a_2 \in Vert_2$ such that $g(e, a_1) \in Edg_1, g(e, a_2) \in Edg_2$, $a_1 = a_2$ must hold.

With this assumption, the composition operator defined here can be combined with the "composable abstraction" operator defined in Sec. 2.4 (Def. 3), to answer a provenance query of the form $prov(y')$. Using these two operators, query answering meets the two main requirements stated at the end of Sec. 3.1, namely (i) to return provenance paths of maximal length, given all the provenance graphs contributions from each of the partners, and (ii) to comply with the sharing policies of each partner relative to the partner that issues the query. Specifically, the answer to the example query is computed as:

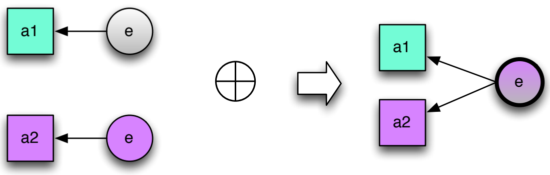
$$prov(y') = abs_C(abs_C(pg_1 \oplus pg_2, pol(A, C)), pol(B, C))$$



(a) compound trace for y' = g(f(x))



(b) compound trace for y' = g(y, x2), y = f(x1,x2))



(c) multiple generation inconsistency

Figure 9: Simple compositions of dependency graphs

[7]Observe that $pol(A, B)$ only comes into play when B queries T for $prov(y)$.

[8]This a minimalistic graph. Additional details on the internal structure of f and g may be included.

To summarize, in this model each partner who produces provenance graphs alongside data, may (but does not have to) send such provenance graphs to a central trusted partner, T. Additionally, partners entrust T with a specification of their provenance sharing policies, one for each other partner in the network, and delegate all further operations on their graphs to T. In return for volunteering such information, T is able to answer provenance queries originated by any partner, taking all available graphs and policies into account. A prototype architecture that supports the functionality of partner T is currently being developed.

4. SUMMARY AND FURTHER WORK

We have described a model and algorithm for performing abstractions on provenance graphs, and we have presented a scenario where abstraction is used to enable privacy-preserving sharing of provenance graphs across a network of partners. The main motivation for this work comes from the observation that, increasingly, data sharing must occur amongst partners who agree to collaborate towards a common goal (be it a business relationship, joint emergency response, etc.), but at the same time are not prepared to blindly trust the data they receive from other partners. In this setting, we conjecture that sharing the provenance of the data may increase the usefulness of the data that is exchanged, and we explore the consequences of sharing provenance graphs. Such graphs, however, may encode aspects of a partner's process model, as well as references to private data, which partners are reluctant to share. Our model for provenance graph abstraction makes it possible for partners to share some aspects of provenance while preserving the privacy of others. Abstraction is based on a simple policy model, consisting of a set of nodes in the graph that are to be abstracted out. We show that in order to preserve the consistency of the original graph, the abstraction algorithm may need to include additional nodes that were not initially required to be abstracted. Current work is focused on extending the abstraction algorithm to a richer provenance model, that comes closer to the complete PROV reference model. This includes a stronger notion of consistency to be applied both to abstraction and to composition.

In the second part of the paper, we have described an extension of the provenance exchange scenario, where some of the shared data is used by other partners to produce new data, and therefore new provenance, which can then itself be shared, producing a propagation effect across a network of partners. A different policy may be defined amongst each pair of partners, a sender and a receiver. This scenario requires provenance fragments to be composed, while different abstractions are applied to each fragment, depending on the privacy policy of the fragment's owner relative to the receiver of the compound provenance graph. In this interaction model, partners send their data to other partners, and send the associated provenance to a separate trusted partner, T, who is responsible for performing composition and abstraction. The advantage of this model is its simplicity: partners "send and forget", and all requests for compound provenance graphs are directed to T.

The current model offers a fully functional sharing infrastructure for provenance data. In our future work, we shall be looking into two major extensions that will remove the need for a centralised trusted entity, and extend the current policy language. In the former, T is removed and partners are themselves responsible for answering provenance queries about the provenance fragments that they have produced, in line with the sharing policy relative to the requester, and composition must preserve this hiding. The latter will look into expanding the policy language beyond just specifying the list of nodes to be restricted to include nodes that have to be retained to preserve the core provenance information, as well as combinations of removed and retained nodes that should be disallowed. Furthermore, additional strategies for specifying abstraction chunks will be explored for different intended purposes of the abstraction.

5. REFERENCES

[1] D. Andrews, editor. *Information technology - Programming languages, their environments and system software interfaces - Vienna Development Method - Specification Language - Part 1: Base language.* International Organization for Standardization, December 1996. International Standard ISO/IEC 13817-1.

[2] K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes. PROV-DM: The PROV Data Model. Technical report.

[3] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. Class-based graph anonymization for social network data. *Proc. VLDB Endow.*, 2(1):766–777, Aug. 2009.

[4] O. Biton, S. C. Boulakia, S. B. Davidson, and C. S. Hara. Querying and Managing Provenance through User Views in Scientific Workflows. In *ICDE*, pages 1072–1081, 2008.

[5] J. Bryans and J. S. Fitzgerald. Formal Engineering of XACML Access Control Policies in VDM++. In M. Butler, M. G. Hinchey, and M. M. Larrondo-Petrie, editors, *ICFEM*, volume 4789 of *Lecture Notes in Computer Science*, pages 37–56. Springer, 2007.

[6] J. Bryans, J. S. Fitzgerald, C. B. Jones, and I. Mozolevsky. Formal modelling of dynamic coalitions, with an application in chemical engineering. In *ISoLA*, pages 91–98. IEEE, 2006.

[7] G. Chang, C. B. Roth, C. L. Reyes, O. Pornillos, Y.-J. Chen, and A. P. Chen. Retraction. *Science*, 314(5807):1875, 2006.

[8] J. Cheney, P. Missier, and L. Moreau. Constraints of the Provenance Data Model. Technical report.

[9] S. B. Davidson, S. Khanna, T. Milo, D. Panigrahi, and S. Roy. Provenance views for module privacy. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '11, pages 175–186, New York, NY, USA, 2011. ACM.

[10] S. B. Davidson, S. Khanna, S. Roy, and S. C. Boulakia. Privacy issues in scientific workflow provenance. In P. Missier, V. Curcin, and S. Dadvidson, editors, *First International Workshop on Workflow Approaches to New Data-centric Science (WANDS'10)*, Indianapolis, 2010. ACM.

[11] S. B. Davidson, S. Khanna, S. Roy, J. Stoyanovich,

V. Tannen, and Y. Chen. On provenance and privacy. In *Proceedings of the 14th International Conference on Database Theory*, ICDT '11, pages 3–10, New York, NY, USA, 2011. ACM.

[12] S. Dey, D. Zinn, and B. Ludäscher. ProPub: Towards a Declarative Approach for Publishing Customized, Policy-Aware Provenance. In J. Bayard Cushing, J. French, and S. Bowers, editors, *Scientific and Statistical Database Management*, volume 6809 of *Lecture Notes in Computer Science*, pages 225–243. Springer Berlin / Heidelberg, 2011.

[13] Editorial. Closing the Climategate. *Nature*, 468(345), 2010.

[14] C. B. Jones. *Systematic software development using VDM (2. ed.)*. Prentice Hall International Series in Computer Science. Prentice Hall, 1991.

[15] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 93–106, New York, NY, USA, 2008. ACM.

[16] P. Missier, B. Ludascher, S. Bowers, M. K. Anand, I. Altintas, S. Dey, A. Sarkar, B. Shrestha, and C. Goble. Linking Multiple Workflow Provenance Traces for Interoperable Collaborative Science. In *Proc.s 5th Workshop on Workflows in Support of Large-Scale Science (WORKS)*, 2010.

[17] A. Morin, J. Urban, P. D. Adams, I. Foster, A. Sali, D. Baker, and P. Sliz. Shining Light into Black Boxes. *Science*, 336(6078):159–160, 2012.

[18] R. Peng. Reproducible Research in Computational Science. *Science*, 334(6060):1226–1127, Dec. 2011.

[19] E. Zheleva and L. Getoor. Preserving the Privacy of Sensitive Relationships in Graph Data. In F. Bonchi, E. Ferrari, B. Malin, and Y. Saygin, editors, *Privacy, Security, and Trust in KDD*, volume 4890 of *Lecture Notes in Computer Science*, pages 153–171. Springer Berlin / Heidelberg, 2008.

[20] B. Zhou, J. Pei, and W. Luk. A brief survey on anonymization techniques for privacy preserving publishing of social network data. *SIGKDD Explor. Newsl.*, 10(2):12–22, Dec. 2008.

APPENDIX

A. THE PROPUB EXAMPLE

The work most closely related to ours is presented in [12], in which an algorithm for hiding parts of a provenance graph is given. Here, we demonstrate Alg. 1 using the graph and example from [12]. A number of operations are given in that paper; Alg. 1 corresponds most closely to the abstraction operation. The graph used is reproduced in Figure 10.

The set of nodes to be removed is $\{d_{14}, s_1, m_1\}$. Extending these to a-nodes (line 4, using algorithm 4) gives $\{s_1, s_2, s_3, d_{14}, m_1\}$ as the set of elements to be removed. The algorithm then identifies the cause boundary nodes (given by algorithm 3, and in this case contains only the node $\{m_1\}$), and the effect boundary nodes (which are given by algorithm 2, and in are this case $\{s_1, s_2, s_3\}$). The paths between nodes in cause boundary set and the effect boundary set are calculated, and these nodes gives us a new value for the variable *Remove* $(= \{s_1, s_2, s_3, d_{13}, d_{14}, m_1\}$.)

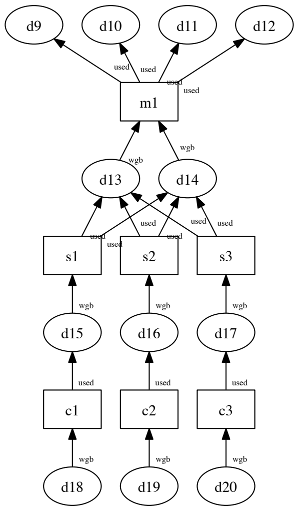


Figure 10: Graph used to illustrate privacy policies in ProPub

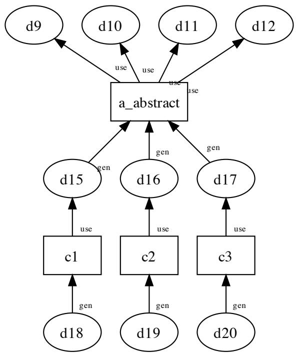


Figure 11: Result of abstraction with policy $\{d_{14}, s_1, m_1\}$

At this point, *Remove*$' \neq$ *Remove* so the process needs to be repeated from line 2. The second iteration does not further change the value of *Remove*, (it remains $\{s_1, s_2, s_3, d_{13}, d_{14}, m_1\}$ and so the algorithm exits the repeat-until loop and continues from line 9. Line 9 adds the new abstract node n_{abs}. Lines 10-19 perform the "rewiring" of the graph, which create the links to n_{abs} and removes the links to nodes in *Remove*, and finally line 20 removes the set *Remove* of nodes to be abstracted out.

The resulting abstracted graph is shown in Fig. 11 and is identical to the one generated by the Propub operation system, however we compute this result directly, without creating and then repairing graph consistency violations.