Access control and view generation for provenance graphs

Danger, R, Curcin, V, Missier, P & Bryans, J

Original citation & hyperlink:
Danger, R, Curcin, V, Missier, P & Bryans, J 2015, 'Access control and view generation for provenance graphs' *Future Generation Computer Systems*, vol 49, pp. 8-27
https://dx.doi.org/10.1016/j.future.2015.01.014

DOI 10.1016/j.future.2015.01.014
ISSN 0167-739X

Publisher: Elsevier

Access control and views generation for OPM provenance graphs

Roxana Danger[a], Vasa Curcin[a], Paolo Missier[b], Jeremy Bryans[b]

[a]*Department of Computing*
Imperial College London
South Kensington
London SW7 2AZ, UK
[b]*School of Computing Science*
Newcastle University
Newcastle NE1 7RU, UK

Abstract

Data provenance refers to the knowledge about data source and operations carried out to obtain some piece of data. A provenance-enabled system maintains record of the interoperation of processes across different modules, stages and authorities to capture the full lineage of the resulting data, and typically allows data-focused audits using semantic technologies, such as ontologies, that capture domain knowledge. However, regulating access to the captured provenance data is a non-trivial problem, since execution records form complex, overlapping graphs with individual nodes possibly being subject to different access policies. Applying traditional access control to provenance queries would then either hide from the user the entire graph with nodes that had access to them denied, or return an informative and/or semantically invalid graph. An alternative approach is to answer queries with a transformed graph that abstracts over the missing nodes and fragments. In this paper, we present an access control language for provenance data using Open Provenance Model that supports this approach, together with an algorithm that transforms graphs to obtain into safe, valid ones under the conditions defined by the access control language.

Keywords: Provenance, Semantic Web, Access Control Language

1. Introduction

Data provenance refers to the knowledge about data source and operations carried out to obtain a set of data. Making software systems provenance-aware enables users to investigate data sources and services that produced a particular output from the system, together with the individuals who instigated the requests and received those outputs. In such way, the software data outputs can be audited to establish their exact lineage and assess that correct procedures were followed.

Provenance information can be represented as directed acyclic graphs (DAG) establishing causal relationships between individual nodes. The Open Provenance Model [1] (OPM) is a standard for provenance description. It defines three basic elements to be linked: artifacts, agents, and processes; and five basic causal dependences: a process *used* an artifact, an artifact *was generated by* a process, a process *was controlled by* an agent, a process *was triggered by* another process, and an artifact *was derived from* another artifact. Following these basic elements, and others related to temporal dependences, agents and process roles and accounts, OPM graphs can be drawn to express the causal dependences amongst the elements interacting in a system. Digital representation of any provenance element is also an essential requirement in OPM, which has adopted URIs as the protocol to define the OPM elements, and is advocating the usage of RDF and OWL technologies for OPM graph format storage and interpretation, e.g. OPMV and OPMO ontologies [2].

A new set of security considerations arises for provenance data in relation to regulating access to records of a resource, rather than the resource itself, e.g. whether a user is allowed to access provenance trail of a certain process, or of a process affecting a certain data item. Even if a data item itself is not accessible to the user, this should not necessarily restrict the user from accessing some information about it – auditor may not be allowed to see a patient's full electronic health record, but may see that the patient was entered into a clinical trial. To answer such questions, not only is a novel language needed to specify constraints, but a new mechanism which would allow access to transformed provenance graphs that provide some, allowed, information in answer to the query.

Access control is a generic term for ensuring that a principal (person, process, ...) has access only to the services/data that they are authorised for in a certain system. This is typically implemented through security policies

that try to enforce a certain protection goal such as to prevent unauthorized disclosure (secrecy) and intentional or accidental unauthorized changes (integrity). In general, authorisations on some resource can be positive and negative, although traditionally both have been used in mutual exclusion based on the two classical closed policy and open policy approaches. Closed policy adheres to a deny-by-default policy, where access to a resource is only granted if a corresponding positive authorisation policy exists. Open policy systems allow all access unless a corresponding negative authorisation policy exists. In practice a combined approach is used to support exceptions. If multiple policies apply to some resource (e.g. a specific one, and a more generic one) conflict resolution takes place depending on the approach adopted by the system, some typical approaches including denials-take-precedence, most-specific-takes-precedence, priority levels, and time-dependent access.

When the resources to be accessed are linked with a complex structure, i.e. in a tree or a graph, denied resources prevent browsing the whole system and answering certain queries, since alternative, non-denied paths, are not computed. In order to compute such alternative paths, the initial complex structure needs to be transformed. In particular, transforming OPM graphs makes it possible to: 1) views generation depending on user roles and/or data analysis goals; 2) generate semantically valid view of the original graph where the access to all its resources is guarantee; 3) obtain the whole set of query results that can be exposed to an user; 4) minimize the time-cost of access evaluation since transformation can be done only once per user, for example, at connection time; and 5) improve the visualization graph user-interface, as view are user and task-oriented.

Our goal is to *define a control access system for provenance OPM graphs which guarantee the maximum accessibility to the non-denied resources*. An overview of the proposed solution is given in the following section.

2. OPM graph access control by graph transformation: overview of our proposal

Provenance graph access needs to be semantically oriented as the provenance data themself. Therefore, we have extended a semantically-oriented access control language for provenance [3] to allow provenance graph to be transformed according to user specifications and provenence inferences rules. The defined language is called *TACLP, the Transformation-oriented Access Control Language for provenance*.

The access control language in [3] is an extension form the initial one defined by Qun Ni et. al [4] which include a semantic approach description of subjects (user roles), resources to be accessed, conditions under which restrictions are applied, and four different types of access permissions. Cadenhead et. al [3] added regular expressions for resource and condition descriptions. Our extension allows users, to define subgraphs of the OPM that needs to be transformed, and three different levels of abstractions (namely *hide, minimal and maximal*) are allowed for the transformations. Details of the whole language can be found in Section 4.

Using this language, an user can *define the access policy* for his data. This access policy is *evaluated* (Section 5) against a provenance graph, and the set of nodes where the access is restricted, and therefore where the transformation needs to be performed, the level of abstraction with which the nodes needs to be abstracted and other details are collected. The graph is then transformed accordingly and the transformed view of the graph is returned. Querying and browsing can then executed over the generated view of the graph.

The provenance graph transformation or view generation process is formalized in Section 3. Two different operations to transform the graphs are defined: *remove* and *replacement*. The first allows to remove a set of nodes, while the second allows to replace a set of nodes with a new *abstract node*. However applying these operations on an arbitrary set of nodes can introduce *false dependences*, that is, dependences in the view that were not in the original graph.

To avoid this, the input set of nodes are partitioned in such a way that that remove and replacement operations can be performed over the nodes in each partition element without the risk of introducing false dependences. Such a partition is called a *causality-preserving partition*, and can always be computed according to Theorems 1 and 2 in Section 3. These theorems establish that a causality-preserving partition is such that each element of the partition contains a node whose *external causes and effects through the input set of nodes* are shared by the rest of nodes in the partition element. External causes (effects) of a node through a input set of nodes are the nearest causes (effects) to the node, that are not in the input set, but are linked to the node by a path of nodes in the input set.

Finally, computing causality-preserving partitions with minimal cardinality, called as *optimal causality preserving partition*, is desired since each partition element can be further partitioned, and, having this, a hierarchy of

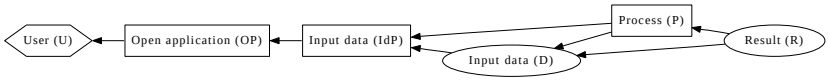


Figure 1: OPM graph example.

partitions could be computed and given as alternative views to an user. In this paper, the partition hierarchy is not explored, but an optimized algorithm for computing optimal causality preserving partitions, including user-specified restrictions, is described at the end of the Section 3. The algorithm is based on the Theorem 3 which states the conditions under which a causality-preserving partition is optimal.

Although the same concepts can be applied to provenance graphs described using PROV ontology [5], in this paper we use the OPM [2] formalism[1], as the proposal needs to be integrated to a provenance API, which follows OPM. We plan to extend the proposal for considering the whole set of constructors introduced in PROV. A discussion of the whole proposal in comparison with other solutions in the literature is provided in Section 7.

3. Provenance foundations

In general terms, an OPM graph [2] is a directed acyclic graph composed by three entities to be linked: artifacts, agents, and processes; and five basic causal dependences: a process *used* an artifact, an artifact *was generated by* a process, a process *was controlled by* an agent, a process *was triggered by* another process, and an artifact *was derived from* another artifact. Graphically, artifacts are represented as ellipses, process as rectangles and agents as hexagons. Figure 1 shows a simple example of a graph describing a generic software process in which an user U opened an application within a process represented by OP and used an input data option IdP of the software that allowed him to enter some data, D, and then use the process P to obtain a result, R dependent on the entered data.

Definition 1 (*Provenance graph*)**.** *A provenance graph,* $Prov = (V, E, type)$ *consists of a directed acyclic graph* $G = (V, E)$*, with functions* $s, t : E \to V$

[1]To be more precise, we define an extension of OPM which introduces an abstract provenance entity and an abstract causal dependence type. The reasons for such extension can be found in Section 3.

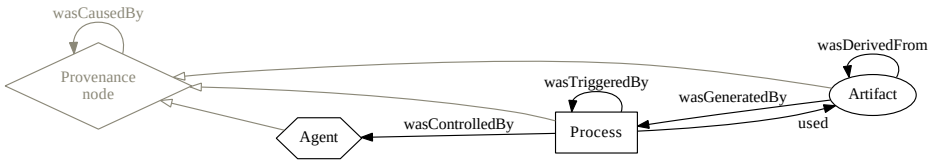


Figure 2: OPM type graph.

representing sources and targets of each edge, and the graph morphism type : $G \rightarrow TG$ *which maps graphs onto the provenance type graph* $OPM\_Type \in TG$ *shown in Figure 2.*

Figure 2 is an extension of the provenance graph type defined in [2]. The extension consists on the addition of two basic concepts (for node and causal dependence types) that enable to represent the OPM concepts in a simple but powerful hierarchy. In the type graph, it can be noticed a new node type labeled as "Provenance node", depicted as a diamond and lighter color, which represents the abstract, high level concept for the provenance nodes; and a new relationship "was caused by", depicted in lighter colour, that represents the abstract, high level concept for the provenance causal dependence definitions. Given so, any path in a provenance graph define an *indirect was caused by* relationship between the nodes it links. This simple notion constitute an important contribution of this work, as it introduces a basic hierarchy amongst the provenance nodes and relationships and at the same time provide a semantic abstract definition for any multi-step causal dependence. Therefore, the existence of process nodes in a path does not prevent its transformation onto an inferred relationship as happened in [2].

All possible relationships (direct and indirect) in a provenance graph are summarized in Table 1. $.^{\mathcal{I}}$ is used to denote the elements of type . (an interpretation), $*$ is used to define the transitive closure of relations and $+$ to nominate indirect relationships. For example, $Artifact^{\mathcal{I}}$ denotes the artifact nodes, and $wdf^{+^{\mathcal{I}}}$ denote the set of the indirect *was derived from* relationships. However, for simplicity, we will use the type name instead of the explicit notation for its interpretation when the case is clear from the context; also $*$ can be used instead of $+$ for wdf and c just to highlight the fact that these two relationships are the only transitive ones according to the semantics of provenance relations. The relationships *caused by* and *indirect caused by*, introduced in this paper, are represented in the table as c and $c+$ respectively. We use $provInf$ to describe the set of provenance inferences as defined in [2]

($provInf = \{wdf^+, u^+, wgb^+, wtb^+\}$), and $provInfExt$, for the whole set of inferences defined in this paper ($provInfEt = \{wdf^+, u^+, wgb^+, wtb^+, c^+\}$).

Relation Name	Subset of	Semantics
wdf	$Artifact^{\mathcal{I}} \times Artifact^{\mathcal{I}}$	artifact *was derived from* another artifact
u	$Process^{\mathcal{I}} \times Artifact^{\mathcal{I}}$	process *used* an artifact
wgb	$Artifact^{\mathcal{I}} \times Process^{\mathcal{I}}$	artifact *was generated by* a process
wcb	$Process^{\mathcal{I}} \times Agent^{\mathcal{I}}$	process *was controlled* by an agent
wtb	$Process^{\mathcal{I}} \times Process^{\mathcal{I}}$	process *was triggered by* another process
c	$Entity^{\mathcal{I}} \times Entity^{\mathcal{I}}$	two entities are causally related
wdf^+	wdf^* (transitive closure)	Indirect *was derived from* relationship.
u^+	$u \circ wdf^+$	Indirect *used* relationship
wgb^+	$wgb \circ wdf^+$	Indirect *was generated by* relationship
wtb^+	$wtb \circ wdf^+$	Indirect *was triggered by* relationship
c^+	c^* (transitive closure)	Indirect *was caused by* relationship

Table 1: Relationship types between pair of nodes in a provenance graph. $.^{\mathcal{I}}$ denotes elements of type ".".

As previously announced, the transformation of an OPM graph comes in two flavours: *entity removal*, where the hidden nodes are deleted and *entity replacement*, where the hidden nodes are replaced by a new abstract node. Both are based on the notion of *external effects and causes through a set of nodes*. Intuitively, the external effects (causes) of a set of nodes S through a set of nodes R, with $S \subseteq R$, or simply external effects (causes), are the nearest effects (causes), in $V \setminus R$, to S that could be affected by the removal of the nodes in R. The recognition of the external effects (causes) is crucial because they represent the primary set of nodes that are affected by the removal of the nodes in R. During the removal, external effects and causes needs to be connected in such a way that do not change the causal dependency between them; and, guaranteeing this, the causal relationships amongst all remaining nodes ($V \setminus R$), will not be affected.

A path of a graph is a well-known concept but for clarity we define it as follows since it is used in Definition 3 which formalize the external effects and causes concepts.

Definition 2 (Path of the graph)**.** *A path $p = (v_1, ..., v_n)$ of a provenance graph $Prov = (V, E, type)$ is a sequence of nodes $v_i \in V$, $i \in \{1, ..., n\}$, such that, $(v_i, v_{i+1}) \in E$. The nodes of a path p, denoted by $nodes(p)$ is the set of the nodes in the path. Let $Paths$ be the set of paths in G, and $path : V \times V \rightarrow Paths$ be a function that retrieve the set of paths connecting two nodes.*

Definition 3. *[External effects and causes through a set of nodes] Given a provenance graph $Prov = (V, E, type)$, let R, S be two sets of nodes, such*

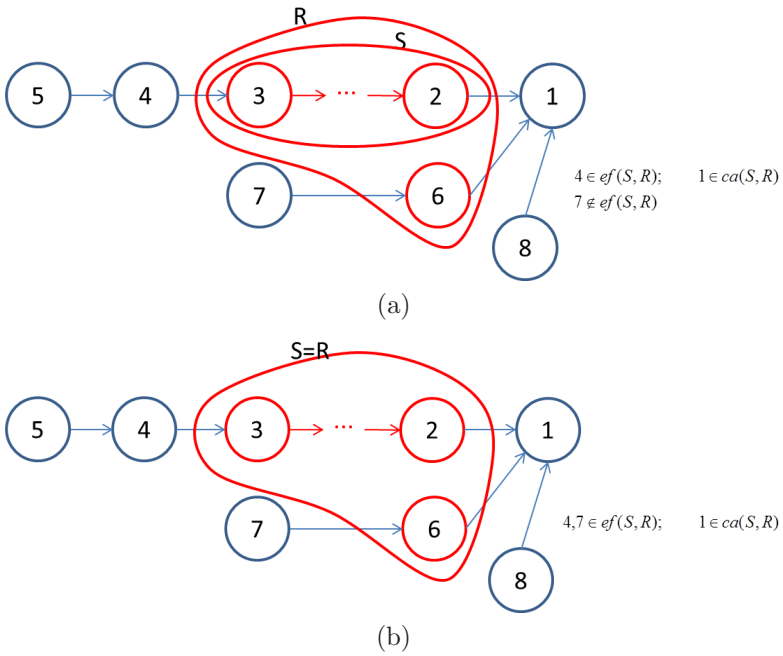


Figure 3: External causes and effects through a set of nodes.

that $S \subseteq R \subseteq V$. The external effects (causes) of the nodes in S through the nodes in R, denoted as $ef(S,R)$, $ca(S,R)$ are sets of nodes in $V \setminus R$ that are causally related with a node in S through a direct relation or a path in R, such that:

- $ef(S,R) = \{v \in V \setminus R | (v,v') \in c^*, v' \in S, \exists p \in path(v,v')$ *such that* $\forall v'' \in nodes(p) \setminus v', v'' \in R\}$
- $ca(S,R) = \{v \in V \setminus R | (v',v) \in c^*, v' \in S, \exists p \in path(v',v)$ *such that* $\forall v'' \in nodes(p) \setminus v', v'' \in R\}$.

If $R = S$, these sets will be denoted with the simplified versions $ef(R)$ and $ca(R)$.

Figure 3 shows a graphic representation of the external causes and effect definition. External effects and causes are the nearest effects and causes, respectively, of the nodes in S arriving to them through a path of nodes in R. In both examples (Figure 3a and the 3b the external causes of the nodes in S through its superset R, $ca(S,R)$ and $ca(R)$ respectively, are the set $\{1\}$.

In the case of the external effects, in Figure 3a, in contrast to Figure 3b cannot included the node 7, since no node from S has caused the appearance of 7. So, $ef(S,R) = \{4\}$ in Figure 3a, and, $ef(R) = \{4,7\}$ in Figure 3b. Finally, notice that the node 5 is not in the external effect sets as it is not the nearest node to the nodes in S; also the node 8 is not in the linage of any node in R, and therefore excluded from any external causes and effects sets.

Definition 4. *[Entity removal operator Rem_R] Let $Prov = (V, E, type)$ and $R \subseteq V$ a set of nodes to be removed. Entity removal transformation Rem_R on $Prov$ produces a new provenance graph $Prov' = (V', E', type')$, where:*

- $V' = V \setminus R$
- $E' = (E \setminus \{(v, v') \in E | v, v' \in R\}) \cup \{(v, v_c) | v \in ef(R), v_c \in ca(R)\}$
- *$type'$ morphism function is defined as:*
  - $\forall v \in V', type'(v) = type(v)$
  - $\forall e \in E' \cap E, type'(e) = type(e)$
  - $\forall e \in E' \setminus E, type'(e) = \begin{cases} wdf & \text{if } type(e) = wdf^+ \\ u & \text{if } type(e) = u^+ \\ wgb & \text{if } type(e) = wgb^+ \\ wtb & \text{if } type(e) = wtb^+ \\ c & \text{otherwise} \end{cases}$

$Prov'$ is called the transformed graph by entity removal on R, and the operation is denoted as: $Prov' = Rem_R(Prov)$.

Removal operator produce a new graph, $Prov'$, from the original one, $Prov$, removing the nodes and edges in R and including new edges linking the external effects and causes of R. The type of the new edges are updated to generalize indirect relationships as the direct analog relationships, or to define the relationship as "caused by"[2].

Definition 5. *[Entity replacement operator Rep_R] Let $Prov = (V, E, type)$ and $R \subseteq V$ a set of nodes to be removed. Entity replacement transformation Rep_R produces a new provenance graph $Prov' = (V', E', type')$, where:*

[2]A new edge could have a "caused by" label either an "indirect caused by" relationship were in $Prov$ or not.

- $V' = (V \setminus R) \cup \{v_a\}$
- $E' = (E \setminus \{(v, v') \in E | v, v' \in R\}) \cup \{(v, v_a) | v \in ef(R)\} \cup \{(v_a, v_c) | v_c \in ca(R)\}$
- *type′ morphism function is defined as:*
  - $\forall v \in V' \setminus \{v_a\}, type'(v) = type(v)$
  - $type'(v_a) = \begin{cases} Artifact, & \text{if } \forall v \in R, type(v) = Artifact \\ Process, & \text{otherwise} \end{cases}$
  - $\forall e \in E' \cap E, type'(e) = type(e)$
  - $\forall e \in E' \setminus E, type'(e)$ *is defined according to the TG graph.*

Prov′ is called the transformed graph by entity replacement on R, v_a *the abstract entity introduced by the transformation and* $AR = \{(v_a, R)\}$ *is called the abstract relation of the transformation, and the operation is denoted as:* $Prov' = Rep_R(Prov)$.

Abstract entities have different semantics than those in the original graph[3]. Therefore, the users should consider this fact during the graph analysis. In any case, it is responsibility of the provenance administrator to give users the information about which nodes have been abstracted in his view.

A transformation by entity removal (Rem_R) is useful in cases in which a subgraph needs to be hidden if it is unnecessary for an analysis or it has been concealed to the user. A transformation by entity replacement (Rep_R) is useful for removing details of data and operations in a subgraph while maintaining a general information (the abstract entity) that suggest the existence of such subgraph. Take as an example a provenance graph generated by a health care system. A patient is not able to see the subgraph associated to the clinical trial in which he participates, neither the actions performed by his GP to decide a diagnosis or a therapy. In the first case, Rep_R is more appropriate, as nodes representing parts of a clinical trial process could indicate to the patient when and which of his clinical data are used for the study. In the second case, is more dangerous to use Rep_R than Rem_R, since Rep_R could suggest the existence of automatic systems (or clinical case discussion in a medical institution) to assess a diagnosis or therapy, and so a patient

[3]Abstract entities represents a subgraph of the original graph.

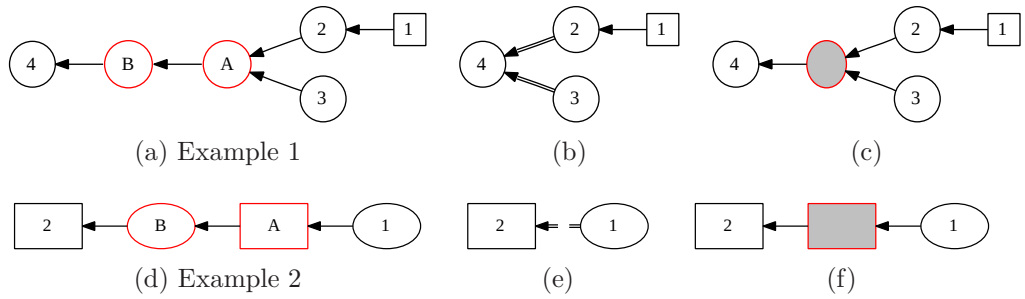


Figure 4: Transformations examples. (b) and (e) Entity removal operation over the examples in (a) and (d), respectively; (c) and (f) Entity replacement operation over the examples in (a) and (d), respectively. Nodes to be removed are denoted with letters and red borders, the nodes to maintain with numbers, indirect dependences are represented as double lines, soft dependences (Definition 6) as dash lines, and abstract entities have gray background and red borders.

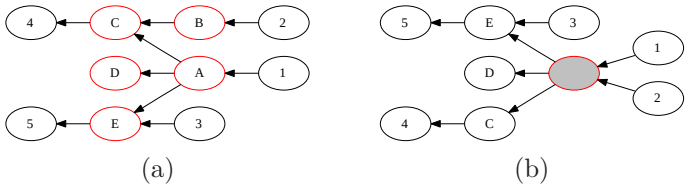


Figure 5: Entity replacement operation which introduce false dependencies (e.g. a new path from *2* to *5*). (a) Original graph, (b) abstract graph using A and B as the nodes to abstract.

could argue about ethical issues during the process. Figure 4 shows two examples to which removal and replacement operations have been applied.

The transformations Rem_R and Rep_R does not introduce cycles in the transformed graph as the original graph re acyclic as well. However, using these transformations in an arbitrary set of nodes can introduce false dependences, that is, causal links that were not in the original graph can appear in the transformed graph.

Figure 5 shows an example in which a transformation by entity replacement introduces false dependencies when the entities A and B are joined. In this case, paths from 2 to 5, D, and E do not exist in the original graph.

The multi-step (inferred) dependences defined in [2] are not useful to represent all possible multi-step causal dependences observed in an OPM graph, as no process node can be "removed" with an inferred dependence. The introduction of the causal dependence *caused by*, c, to represent "any"

casual dependence, and c^+ as "any" indirect causal dependence guarantee that any pair of connected nodes in a graph can be linked by c or by c^+. This simple notion allow us to transform any causal dependence appearing in a graph with a default (c^+) dependence relationship. Therefore, in case that no more specific multi-step dependence can be used to replace a path of causal dependences, c^+ can be used to instead, and in this case *soft dependences* will appear in the transformed graph. The following definition formalizes the concepts of false and soft dependences.

Definition 6 (False and soft dependences introduced by transformation)**.** *Let $Prov = (V, E, type)$ and $Prov' = (V', E', type')$ a transformed graph by entity removal or replacement. A path $p \in path(v, v')$ of $Prov'$, is called false dependence introduced by the transformation if it does not belongs to Prov, and it is called soft dependence introduced by the transformation if $(v, v') \in c^* \setminus \{wdf^+, u^+, wgb^+, wtb^+\}$.*

Definition 7 (Causality-preserving transformation)**.** *As transformation is called causality preserving if it does not introduce false dependences.*

The problem of avoiding the introduction of false dependences during transformations can be reformulated as: *given a provenance graph and a set of entities to be abstracted/hidden, how these entities can be joined/removed from the graph to make all the transformations causality-preserving?* The definition and construction of a partition such that each of its elements can be safely (preserving the causality amongst the remaining nodes) transformed is the proposed solution and aim of the following definitions.

Definition 8 (Causality preserving partition of a set of entities)**.** *Let $Prov = (V, E, L)$ a provenance graph, and $R \subseteq V$. A partition of R, $\mathcal{P}$, is causality preserving if for all $P \in \mathcal{P}$, transformations Rem_P and Rep_P are causality preserving.*

Having introduced the notion of causality preserving partition, we now show how a causality preserving partition can be used to transform a graph (Definition 9), and how it can be constructed (Theorems 1, 2, 3 and Algorithm 1).

Definition 9. *[Transformed graph by a causality preserving partition] Let $Prov = (V, E, type)$ be a provenance graph, and $\mathcal{P}$ be a causality preserving partition of $R \subseteq V$. A transformed graph by $\mathcal{P}$ is the resulting graph from the sequential application of the operations Rem_P or Rep_P, $P \in \mathcal{P}$.*

The requirement to construct such partition is that for each set in the partition all of the external effects and causes of the set are connected.

Theorem 1. *Let $Prov = (V, E, type)$ be a provenance graph, $\mathcal{P}$ is a causality preserving partition of set $R \subseteq V$ iff $\forall P \in \mathcal{P}$, $v_1 \in ef(P, R), v_2 \in ca(P, R), \exists (v_1, v_2) \in c^*$.*

Proof. In Appendix C. □

The previous theorem is equivalent to having a "special" node in each partition element such that its causes and effects include the causes and effect of the remaining nodes of the partition element.

Theorem 2. *Let $Prov = (V, E, type)$ be a provenance graph, $\mathcal{P}$ is a causality preserving partition of $R \subseteq V$, iff $\forall P \in \mathcal{P}, \exists\ v \in P$ such that $\forall v' \in P, v \neq v', ef(\{v'\}, R) \subseteq ef(\{v\}, R)$ and $ca(\{v'\}, R) \subseteq ca(\{v\}, R)$.*

Proof. In Appendix D. □

Each set in a causality preserving partition can be abstracted as a whole, or it can be further partitioned[4] if so desired, without breaking the causality property. This suggests that it is desirable to partition R into large subsets, as this gives users the maximum freedom to choose a specific granularity within each of those subsets. Note that the partition of R consisting of singletons, called *default causality preserving partition* is trivially causality preserving, however this leaves users with no choice at all on the granularity of abstraction, in fact it corresponds to simply anonymizing each of the nodes in R.

Theorem 3. *[Optimal causality preserving partition element] Let $Prov = (V, E, type)$ be a provenance graph and $\mathcal{P}$ a causality preserving partition of $R \subseteq V$. $\mathcal{P}$ is optimal with respect to the cardinality of the partition iff $\nexists P, P' \in \mathcal{P}$ such that $ef(P, R) \subseteq ef(P', R)$ and $ca(P, R) \subseteq ca(P', R)$.*

Proof. In Appendix E. □

[4]Given a causality preserving partition $\mathcal{P}$ an element of $\mathcal{P}$ can be further partitioned following again the Theorems 1 and 2. The proof that the obtained partition is also causality preserving is then trivial.

Combining Theorems 2 and 3 is the key for computing an optimal causality-preserving partition given a set of entities R. This computation can be described as follows. Firstly, compute the external causes and effects of each element in the default partition, that is, for each $v \in R$ compute its external effects and causes through R ($ef(\{v\}, R)$ and $ca(\{v\}, R)$). Then, the optimal partition can be formed combining as much as possible the elements in the default partition while the conditions in both theorems are maintained, that is, each element in the formed partition contain a node whose external causes and effects through R is superset of all other element in the partition element (Theorem 2) and no pair of elements can be further combined (Theorem 3).

For a given graph and set of nodes to transform, it can exist more than one optimal causality preserving partition. The ambiguity emerges when an element can be combined with more than one other element. An order of preferences and/or restrictions for combinations can remove this ambiguity. Both ideas are incorporated in the algorithm for the optimal partition computing with restrictions (Algorithm 1).

A restriction for optimal partition computing is defined as a boolean condition that should be satisfied by each partition element. Let $\mathcal{P}$ be a causality preserving partition for the set of nodes R and P be an element of $\mathcal{P}$, then $R = \cup_{P \in \mathcal{P}} P$. Two examples of restrictions are:

1. **Soft dependences transformations are not allowed for a set of nodes.** Let $softDepsNotAllowedIn$ be the set of nodes for which soft dependences are not allowed. To guarantee this restriction external causes and effects of P, with $P \cap softDepsNotAllowedIn \neq \emptyset$ through R have to be related through a relationship in $provInf$:

$$P \cap softDepsNotAllowedIn = \emptyset \text{ or } (P \cap softDepsNotAllowedIn \neq \emptyset \text{ and } \forall v_c \in ca(P, R); v_e \in ef(P, R), type((v_e, v_c)) \in provInf) \quad \text{(RESTR. 1)}$$

2. **Nodes in a partition element have to share a value property.** Let $dict$ be a hash table containing the property value of each nodes in R:

$$\forall v_1, v_2 \in P, v1 \neq v_2, dict[v_1] = dict[v_2] \quad \text{(RESTR. 2)}$$

	{A}	{B}	{C}	{D}	{E}
externalCauses	{4,5}	{4}	{4}	∅	{5}
externalEffects	{1}	{2}	{1,2}	{1}	{1,3}

Table 2: External causes and effects for the default causality preserving partition elements for Figure 5.

In the first part of Algorithm 1, the external causes and effects of the default partition are calculated and the ordered list of its elements, according to the sum of the cardinality of the external causes and effects and for the lexicographic order of the nodes[5], is used to initialize $sortedP$ variable. The sets $emptyCauses$ and $emptyEffects$, which will be returned at the end of the algorithm, that are also computed in this step as the set of nodes whose external causes and effects are empty sets, respectively. These sets will be used in Algorithm 4, Section 6.

In the second part of the algorithm, elements in $sortedP$ are used to create the causality preserving partition, $\mathcal{P}$. Each time, an element in $sortedP$, denoted by $seed$, is merged iteratively with all other elements of $sortedP$, denoted by $mergeWith$ if 1) the external causes and effects of $seed$ are supersets of the respective sets of $mergeWith$ and 2) all restrictions, $rest$, are satisfied by partition element to be created ($seed \cup mergeWith$). If the merging is performed $mergeWith$ element is removed from $sortedP$ to avoid merging this element again.

The ordering imposed in the first part of the algorithm becomes also relevant for the second one. The first ordering (external causes and effects cardinality) guarantee a minimal number of times that the superset operations are performed. The second ordering (lexicographic) allows to define a merging node order preference, which eliminate the ambiguity when a node can be merged to more than one other node. Any desirable order can be used instead of the lexicographic one.

The execution of this algorithm using as input the details in Figure 5 without restrictions is shown in the following.

Step One

The external causes and effects associated to the default causality preserving partition of the set of entities to abstract ($R = \{A, B, C, D, E\}$) is described in the Table 2. From it, $emptyCauses = \{D\}$, $emptyEffects = \emptyset$ and the list $sortedP = [A, C, E, B, D]$.

[5]URIs are used as identifiers in OPM graphs using semantic approach, as in our case.

Algorithm 1 Optimal causality preserving partition with restrictions.

Require: g: OPM graph.
R: the set of nodes to abstract.
$restr$: set of conditions to be hold by the elements of the partition.
$superSet(.,.)$: is a boolean function between sets that returns "True" if the first argument is superset of the second and "False", otherwise.
Ensure: $\mathcal{P}$: optimal causality preserving partition.
$emptyCauses$: the set of nodes with empty set of external causes.
$emptyEffects$: the set of nodes with empty set of external effects.

{*Step One: external causes and effects for the default partition*}
Compute in $externalCauses$ a dictionary of the external causes for each element $\{v\}, v \in R$, through R
Compute in $externalEffects$ a dictionary of the external effects for each element $\{v\}, v \in R$, through R
$emptyCauses = \{v | v \in R, externalCauses[v] = \emptyset\}$
$emptyEffects = \{v | v \in R, externalEffects[v] = \emptyset\}$
Let $sortedP$ a list of the elements in R sorted first, in descending order of the sum of cardinality of their external causes and effects, and then by the lexicographic order of the node names.

Step Two: generate the partition
$\mathcal{P} = \emptyset$
for $i = 1, ..., |sortedP| - 1$ **do**
    $seed = \{sortedP[i]\}$
    $j = i + 1$
    while $j <= |sortedP|$ **do**
        $mergeWith = \{sortedP[j]\}$
        if $superSet(externalCauses[sortedP[i]], externalCauses[sortedP[j]])$ **and** $superSet(externalEffects[sortedP[i]], externalEffects[sortedP[j]])$ **and** $\forall r \in restr, r$ is satisfied by $seed \cup extendWith$ **then**
            $seed = seed \cup mergeWith$
            Remove the j-esim element of $sortedP$
        else
            $j = j + 1$
    $\mathcal{P} = \mathcal{P} \cup \{seed\}$
return $\mathcal{P}, emptyCauses, emptyEffects$

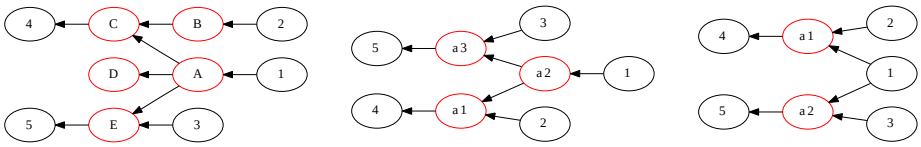


Figure 6: Applying Rep_R and Rem_R operations to the elements in an optimal causality preserving partition. (a) Original graph, (b) View of the graph after applying Rep_R to all optimal partition elements, (c) View of the graph after applying Rem_R to {C, B} set and Rep_R to the other two partition elements.

Step Two
$\mathcal{P}$ is initialized as an empty set, $\emptyset$, and the double loop proceed as:

1. seed = {A}
   (a) $mergeWith = \{C\}$. Condition in line 12 is False.
   (b) $mergeWith = \{E\}$. Condition in line 12 is False.
   (c) $mergeWith = \{B\}$. Condition in line 12 is False.
   (d) $mergeWith = \{D\}$. Condition in line 12 is True, then $seed = \{A, D\}$, $sortedP = [A, C, E, B]$
   (e) $\mathcal{P} = \{\{A, D\}\}$
2. seed = {C}
   (a) $mergeWith = \{E\}$. Condition in line 12 is False.
   (b) $mergeWith = \{B\}$. Condition in line 12 is True, then $seed = \{C, B\}$, $sortedP = [A, C, E]$.
   (c) $\mathcal{P} = \{\{A, D\}, \{C, B\}\}$
3. seed = {E}
   (a) $\mathcal{P} = \{\{A, D\}, \{C, B\}, \{E\}\}$

The output of the algorithm is: $\mathcal{P} = \{\{A, D\}, \{C, B\}, \{E\}\}$, $emptyCauses = \{D\}$, $emptyEffects = \emptyset$. Figure 6 show two views of the original graph: in the first Rep_R operation is applied to all element in $\mathcal{P}$, in the second Rem_R is applied to the {C, B} set and Rep_R to the other two.

4. TACLP, the Transformation-oriented access control language for provenance

The Access Control Language for Provenance data described here is an extension of the works in [4, 3]. In these works the access control to a

provenance graph is provided through a set of *policies* expressed in an XML syntax. When a query is asked, the policies are applied and the query is answered if no resource in the result is denied.

In contrast, our extension, called Transformation-oriented access control language for provenance -TACLP-, allow the description of transformations in case of restrictive access and, for query answering, the policies are evaluated and the transformed graph is used instead of the original one. In addition to securing the graph, a key feature of our approach is to provide access to the maximum allowable subset, without unnecessarily hiding any nodes.

The TACLP XML schema can be found in Appendix A. We assume that both provenance graph items and security policy elements refer to ontological terms, thereby providing unambiguous interpretation, tighter domain coupling, easier human verification, and allowing existing Semantic Web tools to be used for data access and analysis.

The access control is defined by a set of *policies* and a *policy evaluation type*. The latter can take two values: 'deny takes precedence' or 'permit takes precedence'. In previous works only 'deny takes precedence' was described. However, 'permit takes preference' could be useful to define a cascade of policies separating users according to their access level. Section 5 details how policies are applied during query evaluation. In the following a summary of the elements defined in [4] is given. Section 4.1 details the *transformation* element introduced in this paper.

A policy consists of a *target*, *effect*, and a *transformation* elements and optionally *condition* and *obligation* elements.

Element *target* specifies the set of users (*subject* element) and resources (*record* element) to which the policy should be applied, and optional elements *restriction* and *scope*. *Subject* and *record* elements are specified by a collection of specific concepts, defined through IRI references. For example, *doctor* and *patient* can be used to define the type of users to which a policy is applied; and *open eHR*, *nurse*, and *laboratory test* are examples of a process, an agent and an artifact concept resources, respectively, to which a policy is applied[6]. The optional *restriction* string element describes a conditional expression under which the policy is applied, which is either a relational comparison between a value in a property path and a literal, or a full logi-

[6]The semantic approach of using concepts to specify users is a simile to user groups in the classical theory of access control

cal expression. The *scope* element determines if the policy is 'transferable' or 'non-transferable' with respect to subjects - whether it applies to all the ancestors of matched elements in the graph, or to the matched elements only.

The *effect* element specifies the intended outcome if the policy is applicable and the rule matches part of the provenance graph. A policy is applicable to a node if the subject and record are superconcepts of the user accessing the provenance graph and the node type, respectively, and if the restrictions and conditions are satisfied by the node and the provenance system, respectively. Effect can take four predefined values:

- **Absolute permit.** This value guarantees access to the graph regardless of other policies outcome, e.g. for allowing access to auditors or law enforcement agencies, and avoids the need for additional conditions in deny policies.
- **Deny.** Guarantees that certain parts of the graph will not be accessed by users in the subject element.
- **Necessary permit.** It is useful to describes necessary conditions, but not sufficient, for accessing to certain parts of the graphs, and therefore needs an explicit permit policy for accessing (See [4] for examples of usefulness of having *necessary permit* in addition to *permit*).
- **Permit.** It is useful to describes those parts of the graph that can be accessed if there are no other policies denying its access.

The optional *condition* element of a policy describes contextual requirements that have to be met in terms of system names, IP address, time constraints etc., rather than ontological concepts. If omitted, the condition evaluates to true.

4.1. Transformation element

The *transformation* element is our novel extension to the policy language defined in [4]. It allows provenance data administrator to specify how to transform the provenance graph in order to hide certain resources. The transformation element includes the specification of which nodes needs to be hidden and which transformation operation (Rem_R and Rep_R) should be applied to them. Depending on the connected user, a view of the graph is generated and the queries answered according to this transformed graph.

```
<xs:element name="transformation" minOccurs="0" maxOccurs="1">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="transformation_spread" type="xs:string"
     minOccurs ="0" maxOccurs="unbounded"/>
   <xs:attribute name="type" use="required">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="Single"/>
      <xs:enumeration value="Subgraph"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="level" use="required">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="Hide"/>
      <xs:enumeration value="Minimum"/>
      <xs:enumeration value="Maximum"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="labelAs" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Figure 7: Transformation element XSD definition

The XSD definition, shown in Figure 7, consists of three parts: the *transformation_type* which determines what segment of the graph needs to be hidden, the *abstraction_level*, and *label* denoting the names for introduced abstract entities in the transformed graphs.

Transformation type can take two possible values:

- **Single.** The original graph is transformed by removing the nodes matching the target policy.
- **Subgraph.** As above, but the set of matching nodes is extended to include also nodes causally dependent on the original matching nodes the target policy whose type is included in the *transformation_spread* element. *Transformation_spread* element is used to specify a concept defined in OWL language.

Three different transformation levels are specified:

- **Hide**, when the matched nodes of the subgraph have to be completely hidden (removed) from the graph, so, Rem_R transformation is applied;

- **Minimum abstraction**, when the nodes of the subgraph have to be abstracted considering $provInf$[7] set of inferences, so, Rep_R transformation is applied but no *caused by* relationship (soft dependences) will appear in the transformed graph.

- **Maximum abstraction**, when the nodes of the subgraph are abstracted as much as possible, that is considering the complete $provInfExt$[8] set of inferences, so, Rep_R transformation is applied but soft dependences could appear in the transformed graph.

4.2. Example: Access to an Electronic Health Record and Clinical Trial Systems

Figure 8 shows a simple example of access control for an Electronic Health Record (EHR) system, such as the ones used in the UK's National Health Service, and a clinical trial data maintenance and analysis system. We assume the existence of a Clinical Data Ontology (CLDO), similar to existing ontologies such as SNOMED-CT [6], that includes the most important clinical concepts in a healthcare system: user, date, patient, general practitioner (GP), diagnosis, etc.. All patient data and its provenance are stored using CLDO. *cldo:* is used as the prefix namespace for CLDO, and *cldo:now* denotes a concept associated to the current date/time which involves a system function call when a node of this concept is accessed[9].

In the example, the following policies are provided (following the order in Figure 8[10]):

1. Patients do not have access to the laboratory processes. These nodes should be transformed at minimum abstraction level, labelling the new abstract entities "Laboratory".
2. Patients do not have access to the clinical trial processes. These nodes should be transformed at minimum abstraction level, labelling the new abstract entities obtained with "Clinical Trial".
3. Patients have no access to any information related to the automatic diagnosis recommendation nor to the graph segment connecting it with

[7] $provInf = \{wdf^+, u^+, wgb^+, wtb^+\}$ as defined in Section 3, see Table 1.

[8] $provInfExt = \{wdf^+, u^+, wgb^+, wtb^+, c^+\}$ as defined in Section 3, see Table 1.

[9] Technique used in ontology definitions for pervasive context-aware systems (e.g. [7])

[10] In the Figure 8, the third and fifth access control policies are omitted as they are similar to the second and first policies, respectively.

the clinical evidences. These nodes need to be transformed at hidden abstract level.

5. Access policy evaluation algorithms

The aim of the new policy evaluation strategy we present is not only to decide whether a resource can be viewed or not, but to construct a view of the graph which satisfies the security restrictions for the users. As described in the previous section, the access control policy type can take two values: 'deny takes precedence' or 'permit takes precedence'. With respect to the access control example in Section 4.2, the authorization system could be divided in two groups: the heathcare center responsibles, which have access to all provenance data, and the rest of the users. Then, for the first group a 'permit takes precedence' strategy can be used; for the second, however, a 'deny takes precendence' strategy is the adequate one.

Algorithms 2 and 3 show the access policy evaluation for each access control policy type. The general idea behing both algorithms is to collect the set of nodes, R, to be transformed, as well as the *level* and *label* associated to them while the target policies are evaluated (lines 10-31 of Algoritm 2 and lines 10-37 of Algorithm 3). Then, using the compiled data the optimal causality preserving partition is computed by using Algorithm 1, provided that set of the restrictions, *rest*, is adequately updated (lines 32-34 of Algorithm 2 and lines 38-40 of Algorithm 3). Notice that the variable $softDepsNotAllowedIn$ collects the set of nodes to be transformed using *minimal abstraction*, that is, where *indirect caused by* inferences cannot be generated. Adding the first restriction (RESTR. 1) no element in the partition having nodes in $softDepsNotAllowedIn$ will generate a soft dependence. Finally, the graph is transformed according to the causality-preserving partition, and the details about the level and label of the nodes (line 33 of Algorithm 2 or line 40 of Algorithm 3) as explained in Section 6. In Appendix F the flowcharts of these algorithms are depicted and can be used as a summary of the whole algorithm described in this section.

The first difference between these algorithms is with respect the order in which 'deny' policy is evaluated. If the access control preference is deny (permit) the deny (permit) policies will be evaluated before the permit (deny) policies. 'Absolute permit' policies have to be always evaluated first in both deny and permit takes precedence strategies to guarantee the access to the

```
<policySet accespolicyType="deny takes precedence">
   <policy ID=1>
        <!-- Patients have access to the provenance data directly associated to
             their EHR and the EHR itself-->
        <target>
            <subject>ocld:Patient</subject>
            <record>opm:Entity | ocld:EHR </record>
            <restriction>ocld:Patient.patId == ocld:User.id</restriction>
            <restriction>ocld:Patient.patId == ocld:EHR.id</restriction>
            <restriction>opm:Entity rdf:property ocld:EHR</restriction>
        </target>
        <condition>ocld:Now.typeWeekDay == ocld:weekend</condition>
        <scope>transferable</scope>
        <effect>permit</effect>
    </policy>
    <policy ID=2>
     <!-- Patients does not have access to the laboratory process, but transform
          these nodes considering 'minim abstraction' level and labelling the
          abstracted entities obtained with "Laboratory"-->
     <target>
            <subject>ocld:Patient</subject>
            <record>ocld:LaboratoryProcess | ocld:LaboratoryArtifact </record>
            <restriction>ocld:Patient.patId == ocld:User.id</restriction>
            <restriction>ocld:Patient.patId == ocld:EHR.id</restriction>
            <restriction>opm:LaboratoryProcess rdf:property* ocld:EHR</restriction>
            <restriction>opm:LaboratoryArtifact rdf:property* ocld:EHR</restriction>
        </target>
        <scope>transferable</scope>
        <effect>deny</effect>
        <transformation level="Minimum" type="Single"/>
    </policy>
    <policy ID=3>
    <!-- Patients have no access to any information associated to a diagnosis that
         was generated by using an automatic diagnosis recommendation process and
         to the subgraph connecting it with the clinical evidences -->
        <target>
            <subject>ocld:Patient</subject>
            <record>ocld:DiagRecommProcess</record>
            <restriction>ocld:Patient.patId == ocld:User.id AND
                         ocld:Patient.patId == ocld:EHR.id AND
                         ocld:EHR rdf:property*/ocld:Diagnosis/opm:wasGeneratedBy*
                                  ocld:DiagRecommProcess</restriction>
        </target>
        <scope>transferable</scope>
        <effect>deny</effect>
        <transformation level="Hide" type="Subgraph">
            <tranformation_spread>ocld:clinicalEvidence</tranformation_spread>
        </transformation>
    </policy>
</policySet>
```

Figure 8: Example of a access control definition for a healthcare system, *ocld:* is used as prefix of the ontology OCLD.

matching nodes. As 'necessary permit' policies constitute a necessary condition for permit an access they have to be always evaluated before that 'permit' policies.

The second difference lies in the way in which nodes not-covered by a policy are treat in the last step of the algorithm: all non-covered nodes will assumed a deny (permit) access in deny (permit) precedence access control type. So, whereas in deny precedence access control type, all non-covered nodes are included in the R set and given 'hide' as the abstraction level and an empty string as the label for the abstracted node information, in permit precedence access control type, non-covered nodes does not affect the set R of nodes to be transformed.

The access control evaluation algorithms take into account, and solve, the ambiguity problem that could appear when a node match more than one policy. Firstly, as described before, the effect of the policies are used to consider them in a certain order. Secondly, policies, inside each evaluation block, are always evaluated in the same order in which they are specified in the TACLP XML file. Finally, it is guaranteed, through the function $matchingEntitiesOf(g, policy)$, that a node is only associated with the policies whose resources are conceptually nearest.Even if a node matches two policies with the same effect type, only the policy defined first in the XML and with the resource type definition semantically nearest to the node will return it as the matching node.

6. OPM graph transformation

OPM graph transformation (or view generation) simply apply the adequately graph operations (removal or replacement) to each element, P, in the optimal partition, $\mathcal{P}$. It is assumed that all nodes a partition element have the same level of abstraction which is guarantee in Algorithms 2 and 3 by adding (RESTR. 2) for the *dlevels* dictionary. Removal transformation is applied for a 'hide' level or, to avoid having uninformative abstract nodes in the extremes of the view (when P has an empty set of external causes or effects and the abstracted node would has an empty label value). Otherwise, replacement transformation is applied. Minimal or maximal abstraction is obtained depending on the nodes in $softDepsNotAllowedIn$, which in turn is used when (RESTR. 1) is added to the restrictions *rest* and used in Algorithms 2 and 3. The OPM graph transformation process is shown Algorithm 4.

Algorithm 2 Access policy evaluation for 'deny takes precedence'

Require: g: OPM graph to access.
$policies$: list of applicable policies per the user.
$transform(...)$: View generation function of g using Algorithm 4 in Section 6.
$matchingEntitiesOf(target)$: For a given $target$, it retrieves the most specific nodes in g matching the target, that is, a nodes is retrieved by this method if it matches $target$ and no other target more specific than $target$ matches with the node.
Ensure: $view$: view of g for which the access is granted.

Step One: evaluate absolute permit'
$covered = \{\}$
for $p = policies[1]...policies[n]$ **do**
  if $p.effect ==$ '$abs.permit'$ **then**
    for $node \in$ matchingEntitiesOf(p.target) **do**
      if $eval(p.cond, node)$ **then**
        $covered = covered \cup \{node\}$

Step Two: evaluate 'deny' or 'necessary permit'
Initialize $dlevel$ as a dictionary to maintain the level associated to an entity.
Initialize $dlabel$ as a dictionary to maintain the label associated to an entity.
$R = \emptyset$ //R will contain the set of nodes to hide or abstract
$softDepsNotAllowedIn = \emptyset$ // Set of nodes for which soft dependences can not be generated.
for $p = policies[1]...policies[n]$ **do**
  if $p.effect ==$ '$deny'$ **or** $p.effect ==$ '$nec.permit'$ **then**
    for $node \in$ matchingEntitiesOf(p.target) **do**
      if $node \notin covered$ **and** (($target.effect ==$ '$deny'$ **and** $eval(p.cond, node)$) **or** ($target.effect ==$ '$nec.permit'$ **and** not $eval(p.cond, node)$)) **then**
        for $node' \in \{node\} \cup subgraph(node, p.transform.spread)$ **do**
          $covered = covered \cup \{node'\}$
          $dlevels[node'] = p.transform.level$
          $dlabels[node'] = p.transform.label$
          $R = R \cup \{node'\}$
          if $p.transform.level ==$ '$min.abstraction'$ **then**
            $softDepsNotAllowedIn = softDepsNotAllowedIn \cup \{node\}$

Step Three: evaluate 'permit'
for $p = policies[1]...policies[n]$ **do**
  if $p.effect ==$ '$deny'$**or**$p.effect ==$ '$nec.permit'$ **then**
    for $node \in$ matchingEntitiesOf(p.target) **do**
      if $p.effect =$ '$permit'$ **and** $eval(p.cond, node)$ **then**
        $covered = covered \cup \{node\}$

Step Four: complete R with the not covered nodes, transform and return the graph view
for $node \in g.nodes$ **do**
  if $node \notin covered$ **then**
    $dlevels[entity] =$ '$hide'$
    $dlabels[entity] =$ ""
    $R = R \cup \{node\}$
$rest = \emptyset$ //$rest$ maintains the set of restrictions to use for computing the optimal partition
$rest = rest \cup \{(RESTR.1)$ for the set of enities in $softDepsNotAllowedIn\}$
$rest = rest \cup \{(RESTR.2)$ using the dictionary $dlevels\}$
$\mathcal{P}, emptyCauses, emptyEffects = optimalPartition(g, R, rest)$ //Use here Algorithm 1.
$view = transform(g, \mathcal{P}, dlevels, dlabels, emptyCauses, emptyEffects)$
return $view$

Algorithm 3 Access policy evaluation for 'permit takes precedence'

Require: g: OPM graph to access.
$policies$: list of applicable policies per the user.
$transform(...)$: View generation function of g using Algorithm 4 in Section 6.
$matchingEntitiesOf(target)$: For a given $target$, it retrieves the most specific nodes in g matching the target, that is, a nodes is retrieved by this method if it matches $target$ and no other target more specific than $target$ matches with the node.
Ensure: $view$: view of g for which the access is granted.

Step One: evaluate absolute permit'
$covered = \{\}$
for $p = policies[1]...policies[n]$ **do**
  if $p.effect ==$ '$abs.permit$' **then**
    for $node \in$ matchingEntitiesOf(p.target) **do**
      if $eval(p.cond, node)$ **then**
        $covered = covered \cup \{node\}$

Step Two: evaluate 'necessary permit'
Initialize $dlevel$ as a dictionary to maintain the level associated to an entity.
Initialize $dlabel$ as a dictionary to maintain the label associated to an entity.
$R = \emptyset$ //R will contain the set of nodes to hide or abstract
$softDepsNotAllowedIn = \emptyset$ // Set of nodes for which soft dependences can not be generated.
for $p = policies[1]...policies[n]$ **do**
  if $p.effect ==$ '$nec.permit$' **then**
    for $node \in$ matchingEntitiesOf(p.target) **do**
      if $node \notin covered$ **and** $target.effect ==$ '$nec.permit$' **and** not $eval(p.cond, node)$ **then**
        for $node' \in \{node\} \cup subgraph(node, p.transform.spread)$ **do**
          $covered = covered \cup \{node'\}$
          $dlevels[node'] = p.transform.level$
          $dlabels[node'] = p.transform.label$
          $R = R \cup \{node'\}$
          if $p.transform.level ==$ '$min.abstraction$' **then**
            $softDepsNotAllowedIn = softDepsNotAllowedIn \cup \{node\}$

Step Three: evaluate 'permit'
for $p = policies[1]...policies[n]$ **do**
  if $p.effect ==$ '$deny$'**or**$p.effect ==$ '$nec.permit$' **then**
    for $node \in$ matchingEntitiesOf(p.target) **do**
      if $p.effect =$ '$permit$' **and** $eval(p.cond, node)$ **then**
        $covered = covered \cup \{node\}$

Step Two: evaluate 'deny'
for $p = policies[1]...policies[n]$ **do**
  if $p.effect ==$ '$deny$' **then**
    for $node \in$ matchingEntitiesOf(p.target) **do**
      if $node \notin covered$ **and** $target.effect ==$ '$deny$' **and** $eval(p.cond, node)$ **then**
        for $node' \in \{node\} \cup subgraph(node, p.transform.spread)$ **do**
          $covered = covered \cup \{node'\}$
          $dlevels[node'] = p.transform.level$
          $dlabels[node'] = p.transform.label$
          $R = R \cup \{node'\}$
          if $p.transform.level ==$ '$min.abstraction$' **then**
            $softDepsNotAllowedIn = softDepsNotAllowedIn \cup \{node\}$
$rest = \emptyset$ //$rest$ maintains the set of restrictions to use for computing the optimal partition
$rest = rest \cup \{(RESTR.1)$ for the set of enities in $softDepsNotAllowedIn\}$
$rest = rest \cup \{(RESTR.2)$ using the dictionary $dlevels\}$
$\mathcal{P}, emptyCauses, emptyEffects = optimalPartition(g, R, rest)$ //Use here Algorithm 1.
$view = transform(g, \mathcal{P}, dlevels, dlabels, emptyCauses, emptyEffects)$
return $view$

Algorithm 4 Provenance graph transformation

Require: g: OPM graph to access.
$\mathcal{P}$: the optimal causality preserving partition obtained from Algorithm 1.
$emptyCauses$: set of nodes in $\mathcal{P}$ having empty cause node set, obtained from Algorithm 1.
$emptyEffects$: set of nodes in $\mathcal{P}$ having empty effect node set, obtained from Algorithm 1.
$dlevels$: dictionary that maintain the level associated to the elements of P.
$dlabels$: dictionary that maintain the label associated to the entity in R.
$concatenate(...)$: string operation that concatenates a lexicographically ordered set of strings.
Ensure: g': view of g in which the access is granted.
$g' = g$
for $P \in \mathcal{P}$ **do**
Let $node$ a node in P
if $(dlevels[node] == `hide'$ **or** $(concatenate(\{dlabels[node] | node \in P\}) == ``''$ **and**
$(P \subseteq emptyCauses$ **or** $P \subseteq emptyEffects))$ **then**
$g' = Rem_P(g')$ // applying Removal operator
else
$g' = Rep_P(g')$ // applying Replacement operator
Set $concatenate(\{dlabels[node] | node \in P\})$ as the value of the abstract entity obtained by the previous replacement.
return g'

Figure 9a shows a subgraph in which Clinical Trial, Decision support and EHR updating systems are traced with our provenance API. The entities affected by the access policy described in Section 4.2 are also highlighted in the figure. Figure 9b shows the result of applying the access control and view generation method described in this paper.

The original graph (Figure 9a) shows the evolution of a EHR of a patient during two visits and the subsequent actions. In the first visit the patient (Ag1) was attended by the general practitioner (Ag3) and the EHR system (Ag2) was used to record all the details of the visit. First, new item creation process (P1) was executed, which generated a new EHR version (EHR v20 - A2) from the current version of the EHR of the patient (EHR v19 - A1). After the patient detailed the symptoms, the GP gave to the patient a prescription (A3) to be followed and a blood test form (A4) to be performed, and updated the data in the EHR system, generating a new version of the EHR (EHR v21 - A5). The date on which the the blood test was performed, the blood test form was used to prepare the instrumentation and conduct the measurement (P3). All these operations were controlled by the laboratory System (Ag4) and a laboratory technician (Ag5). As part of this process, a laboratory condition report was generated (A6), and it triggered the blood test report creation process (P5), which generated the test report (A7) and a new version of the EHR containing the results of the test (EHR v22 - A9). The test and the laboratory condition reports (A7 and A6) were both used during the creation of an electronic Case Report Form, eCRF, (P4), as the patient is involved in a clinical Trial, and his evolution is also followed by the Clinical Trial researcher (Ag6). The result of this action is the eCRF (A8).

	{P3}	{P4}	{A6}	{A8}	{P7}	{P8}	{A11}	{A12}	{A13}	{A14}
externalCauses	{A4}	{A4, A7}	{A4}	{A4, A7}	{P6, A10}	{P6, A10}	{P6, A10}	{P6, A10}	{P6, A10}	{P9}
externalEffects	{P5}	∅	∅	∅	{P9}	{P9}	{P9}	{P9}	{P9}	∅

Table 3: External causes and effects for the default causality preserving partition elements for Figure 9a.

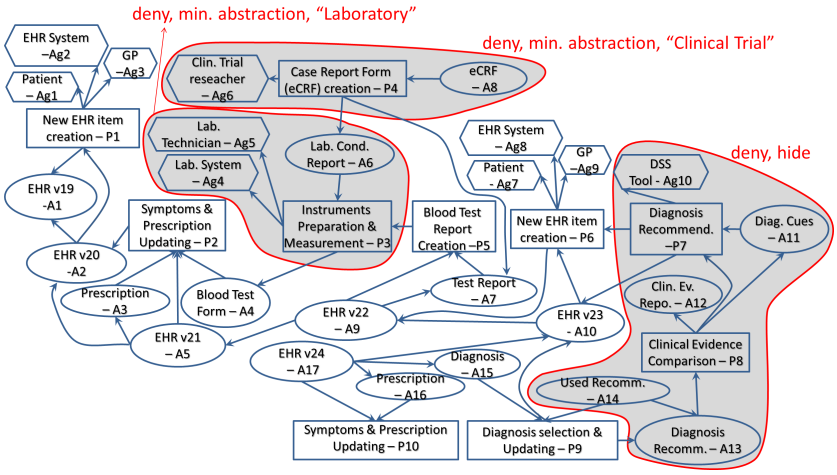
In the second visit, as in the first one, a new EHR item process (P6) was executed, given as a result the new version of the EHR (A10). Followed this, the doctor used a decision support system (Ag10) to confirm his diagnosis hypothesis. He opened the application, introduced the details of the patient (P7), and a set of diagnosis clues (A11) were extracted from the EHR of the patient. These clues were then compared (P8) with the clinical evidence repository (A12) of the decision support system. A diagnosis recommendation (A13) was then obtained and given as a possible option to the GP, who used it to generate his final diagnosis (A15). A variable containing the recommendation chosen by the GP (A14) is also generated and maintained by the decision support tool. Once the GP had the diagnosis, he proceed to update the data in the EHR system, generating a new prescription for the patient (A16), and a new version of the EHR (A17).

The external causes and effects, abstraction level and labels associated to the graph in Figure 9a is shown in Table 3. As can be noticed the output of the Algorithm 1 is the partition {{A11, A12, A13, P7, P8}, {A8, P4}, {P3, A6}, {A14}}.

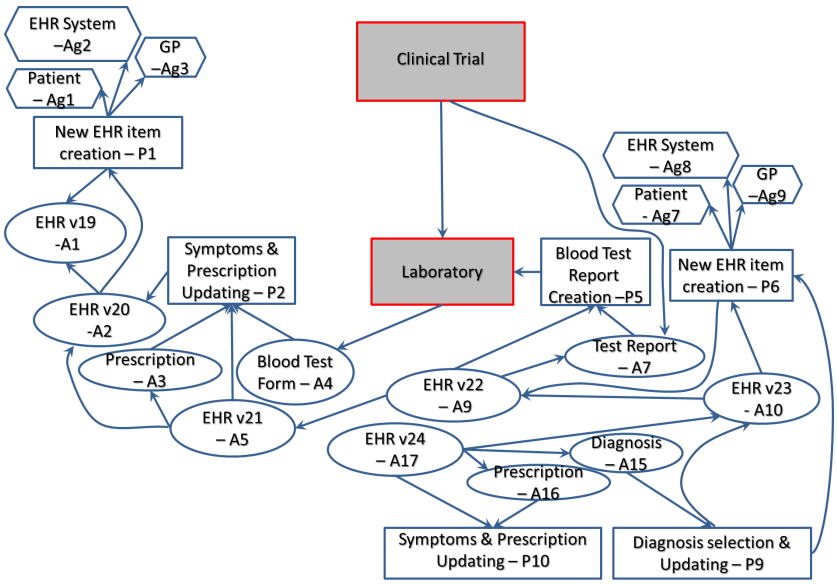
According to the obtained partition, levels and labels to use, the graph in Figure 9b is obtained. Notice that the labels properly describe the aim of the abstracted entities in the cases of laboratory and clinical trial, and the whole subgraph corresponding to the automatic diagnosis decision support processing is removed.

7. Related work

While most of the works on provenance has been focused on its storage, maintenance, and querying, only in the last years the security problem of provenance data is gaining interest. A reason for this fact, could be that provenance data are maintained on relational databases or RDF stores, and the access control is leaved to the security mechanisms provided by these technologies. Security for database technologies has been studied extensively in the past (see [8] for an overview) and the increasing need for maintaining



(a) OPM graph and the access control associated to the nodes .



(b) View of the graph for the patient during the weekends.

Figure 9: OPM graph for EHR-clinical trial example.

semantic resources has developed a large set of works focused on security for RDF data [9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19].

Kaushik et. al [9] introduces various mechanisms for hiding sensible information in RDF graphs, e.g. rename and/or removing RDF subgraphs, as an access control mechanism but no inference processes were involved neither conflict resolution in case of contradictory decisions. Jain et. al [10] introduced an access control model for RDF data which considers inferences and Kim et. al [11] extends this work describing the authorization conflict problem that arises as consequence of access control propagation to subclasses and subproperties when using inference for policy evaluation. Carminati et. al [13] proposes a framework for access control in social networks in which policy and its evaluation uses SW tools. The policy is defined by using SWRL rules, and in this way, the policy evaluation is performed through instance checking and conflict free evaluation in the ontology. All these works contains ad-hoc mechanisms for control access description such as predicates in [9] and SWRL rules in [13].

Recently, Flouris et. al [14] presented a work that contains a high-level specification language for access control, that allow users to define the type of access (permit or deny) for set of triple pattern and a conjunction of triple patterns and constrains that should also be held. It includes also a formal semantic for obtaining the access decision, and some experiment that shown a good scalability performance in relation of different size parameters. Another evolution step in formalizing the access control can be found in [18] where Privacy Preference Ontology (PPO) is defined for allowing users to specify fine-grained privacy preferences for restricting (or granting) access to specific (social) data, and MyPrivacyManager tool can be used to filter data according to preferences described on PPO.

Access control models for XML, as the OASIS eXtensible Access Control Markup Language (XACML), are not suitable for general graphs as RDF data [20]. However, as XACML has become an accepted solution for access control in XML, some works have been produced extending XACML for RDF data. Kounga et. al [12] proposed an extended architecture for allowing users to define authorization policies based on their preferences to allow or deny access to their private data. Helil et. al [15], presented an extension of XACML profile for Role-based Access Control based on the semantic concepts. A OWL-DL reasoner is needed for evaluate the policy and a preliminary evaluation shown a low response time for the evaluation.

XACML cannot be used to define fine-grained access control for struc-

tured data [18]. It is well-known that XACML is very verbose and can become unreadable with few different polices [19, 15]. It also, lacks of a formal semantics for policy combination algorithms [4, 18].

Provenance data imposes some important requirements for access control. Justifications for a semantic approach in provenance and in its security framework can be found in [21, 4, 22]. Kubiatowicz et. al [23] highlighted the importance of maintain the trustworthiness of retrieved provenance data. Aspects such as the integrity, availability, confidentiality migration, location (in distributed environment), verification and migration has also been discussed by Hasan et al. [24] and Braun et. al [20] emphasized the "DAG" and "immutability" nature of provenance information.

Provenance Explorer [25] is a visualization and access control tool for provenance data based on Semantic Technologies and XACML for aceess control definition. OWL is used to describe the provenance model and SWRL to define the inference rules on which the view generation and control access is based. As all access models which do not transform the original graph, Provenance Explorer suffers from deny access to the whole graph if a resource has denied access.

The first work presenting a complete framework for access control in provenance can be found in [4], which are focused on establishing tighter coupling between syntactical language constructs and semantic categories in provenance languages. Ni et. al [4] defined a language for expressing fine-granularity access control for provenance, and the algorithms for evaluating the a policy. Numerous features of XACML can be found in the language but it 1) aims to simplify the policy description making it readable, 2) considers the immutability of the provenance data and, 3) is semantically approached as subjects, resources, constrains, and other elements of the language are based on semantic descriptions.

Cadenhead et. al [3] extended this language to include regular expressions to define elements of the language, and the experiments performed with a prototype and a simple topology shown that their algorithm is efficient, obtaining, for example, less than one second for evaluating queries in a RDF space of 50k triples. A further extension, in [26], enrichs the language with a rewriting system that allows them to produce a valid OPM view from an original valid OPM graph and the embeded rewiriting instructions in the extended language.

Nguyen et. al [22] discuss important issues that needs to be taken into account while modelling control access for provenance and the system imple-

menting these ideas is described in [27]. In particular, the authors highlight the importance of using path patterns to restrict the access and generate valid views, and the need of defining a mechanism for naming the path patterns to be abstracted. We have had, in parallel, the similar ideas and have been included in the transformation element of the TACLP.

The notion of OPM graph type, similar to ours, is described by Sun et. al [28] and used to define complex dependency types. In contrast with our OPM graph type, the nodes of the OPM graph type in [28] represent domain provenance concepts, the edges represent the OPM relationships, and the notion of inheritance is not considered. Having defined a graph type composition operator (to create complex dependency types) and its inverse operator, the system is able to abstract paths matching a predefined dependency path types.

ProPub [29] is another relevant work which considers the strategy of transforming the provenance graph to satisfy some user publishing restrictions. The process is a manually-assisted: an user makes a request to *anonymize*, *abstract* or *hide* a set of nodes in a provenance graph before it can be published. Considering the user specifications, an initial graph is generated and from it the final valid graph, in which integrity constraint violations are solved with the creation of fictitious nodes, is generated and returned. Ficticious nodes are not the same as abstract nodes we define here. In contrast with ficticious nodes, which are generated to solved semantic violations in the generated views, abstract nodes have a well defined semantics according to the REP_P operator and the validity of the generated views is guarantee by the computing of the causality preserving partition.

These works and the presented in this paper share the notion of transforming the provenance graphs in valid views. However, in [26, 22, 27, 28] a manual definition of the writing rules for transforming the provenance graphs is needed, and in [29] fictitious nodes cannot be semantically associated with a subgraph in the original graph. In our work, both issues are considered and solved.

8. Conclusions

Within this paper, we introduced a novel technique for addressing the challenges of provenance record access and security. We extended the existing access control languages in [3] by introducing a *transform* element that allow more flexible policy specifications. We described also a novel access control

evaluation algorithm that returns valid transformed graphs according to the user specifiations.

Our system takes into account the W3C Provenance Group RDF recommendations [30], with the aim of providing a standardised security solution for provenance stores that allows tight coupling to domain concepts, making it understandable, and verifiable by the end-users. The additions and changes we made to the existing proposals increased the overall flexibility of the access control system and introduced a new and interesting way of handling deny policies. Complexity has also increased nevertheless. It is up to policy writers to decide when transformations are acceptable and when they introduce to many overheads in terms of access query processing.

Issues related to visualization, semantic provenance subgraph abstraction and its annotation will be addressed in future works.

References

[1] L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, P. Paulson, The open provenance model: An overview, in: J. Freire, D. Koop, L. Moreau (Eds.), Provenance and Annotation of Data and Processes, Vol. 5272 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2008, pp. 323–326.
URL http://dx.doi.org/10.1007/978-3-540-89965-5_31

[2] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, J. V. den Bussche, The Open Provenance Model core specification (v1.1), Future Generation Computer Systems-doi:10.1016/j.future.2010.07.005.
URL http://eprints.ecs.soton.ac.uk/21449/

[3] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, B. Thuraisingham, A language for provenance access control, in: Proceedings of the first ACM conference on Data and application security and privacy, CODASPY '11, ACM, New York, NY, USA, 2011, pp. 133–144. doi:http://doi.acm.org/10.1145/1943513.1943532.
URL http://doi.acm.org/10.1145/1943513.1943532

[4] Q. Ni, S. Xu, E. Bertino, R. Sandhu, W. Han, An Access Control Language for a General Provenance Model, Vol. 5776 of Lecture Notes in

Computer Science, Springer, 2009, pp. 68–88. doi:10.1007/978-3-642-04219-5_5.

[5] Provenance working group w3c (2011).
URL http://www.w3.org/2011/prov/wiki/Main_Page

[6] T. Benson, Principles of Health Interoperability HL7 and SNOMED, Vol. 49, Springer London, 2010.
URL http://www.springerlink.com/index/10.1007/978-1-84882-803-2

[7] H. Chen, T. Finin, A. Joshi, An ontology for context-aware pervasive computing environments, Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review 18 (2003) 197–207.

[8] E. Bertino, R. Sandhu, Database security – concepts, approaches, and challenges, IEEE TRANS. DEPENDABLE SECUR. COMPUT 2 (1) (2005) 2–19.

[9] S. Kaushik, D. Wijesekera, P. Ammann, Policy-based dissemination of partial web-ontologies, in: Proceedings of the 2005 workshop on Secure web services, SWS '05, ACM, New York, NY, USA, 2005, pp. 43–52. doi:10.1145/1103022.1103030.
URL http://doi.acm.org/10.1145/1103022.1103030

[10] A. Jain, C. Farkas, Secure resource description framework: an access control model, in: Proceedings of the eleventh ACM symposium on Access control models and technologies, SACMAT '06, ACM, New York, NY, USA, 2006, pp. 121–129. doi:10.1145/1133058.1133076.
URL http://doi.acm.org/10.1145/1133058.1133076

[11] J. Kim, K. Jung, S. Park, An introduction to authorization conflict problem in rdf access control, in: Proceedings of the 12th international conference on Knowledge-Based Intelligent Information and Engineering Systems, Part II, KES '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 583–592. doi:10.1007/978-3-540-85565-1_72.
URL http://dx.doi.org/10.1007/978-3-540-85565-1_72

[12] G. Kounga, M. C. Mont, P. Bramhall, Extending xacml access control architecture for allowing preference-based authorisation, in: Proceedings

of the 7th international conference on Trust, privacy and security in digital business, TrustBus'10, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 153–164.
URL http://dl.acm.org/citation.cfm?id=1894888.1894907

[13] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, B. Thuraisingham, A semantic web based framework for social network access control, in: Proceedings of the 14th ACM symposium on Access control models and technologies, SACMAT '09, ACM, New York, NY, USA, 2009, pp. 177–186. doi:10.1145/1542207.1542237.
URL http://doi.acm.org/10.1145/1542207.1542237

[14] G. Flouris, I. Fundulaki, M. Michou, G. Antoniou, Controlling access to rdf graphs, in: A. Berre, A. Gmez-Prez, K. Tutschku, D. Fensel (Eds.), Future Internet - FIS 2010, Vol. 6369 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 107–117, 10.1007/978-3-642-15877-3_12.
URL http://dx.doi.org/10.1007/978-3-642-15877-3_12

[15] N. Helil, K. Rahman, Extending xacml profile for rbac with semantic concepts, in: Computer Application and System Modeling (ICCASM), 2010 International Conference on, Vol. 10, 2010, pp. V10–69 –V10–74. doi:10.1109/ICCASM.2010.5622888.

[16] A. Khaled, M. Husain, L. Khan, K. Hamlen, B. Thuraisingham, A token-based access control system for rdf data in the clouds, in: Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, 2010, pp. 104 –111. doi:10.1109/CloudCom.2010.76.

[17] A. DElia, J. Honkola, D. Manzaroli, T. Cinotti, Access control at triple level: Specification and enforcement of a simple rdf model to support concurrent applications in smart environments, in: S. Balandin, Y. Koucheryavy, H. Hu (Eds.), Smart Spaces and Next Generation Wired/Wireless Networking, Vol. 6869 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2011, pp. 63–74, 10.1007/978-3-642-22875-9_6.
URL http://dx.doi.org/10.1007/978-3-642-22875-9_6

[18] O. Sacco, A. Passant, S. Decker, An access control framework for the web of data, IEEE TrustCom/IEEE ICESS/FCST, International Joint Conference of 0 (2011) 456–463. doi:http://doi.ieeecomputersociety.org/10.1109/TrustCom.2011.59.

[19] B. Stepien, S. Matwin, A. Felty, Advantages of a non-technical xacml notation in role-based models, in: Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on, 2011, pp. 193–200. doi:10.1109/PST.2011.5971983.

[20] U. Braun, A. Shinnar, M. Seltzer, Securing provenance, in: The 3rd USENIX Workshop on Hot Topics in Security, USENIX HotSec, USENIX Association, Berkeley, CA, USA, 2008, pp. 1–5.

[21] S. S. Sahoo, A. Sheth, C. Henson, Semantic Provenance for eScience: Managing the Deluge of Scientific Data, IEEE Internet Computing 12 (4) (2008) 46–54. doi:10.1109/mic.2008.86.
URL http://dx.doi.org/10.1109/mic.2008.86

[22] D. Nguyen, J. Park, R. Sandhu, Dependency path patterns as the foundation of access control in provenance-aware systems, in: Proceedings of the 4th USENIX conference on Theory and Practice of Provenance, TaPP'12, USENIX Association, Berkeley, CA, USA, 2012, pp. 4–4.
URL http://dl.acm.org/citation.cfm?id=2342875.2342879

[23] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao, Oceanstore: an architecture for global-scale persistent storage, SIGPLAN Not. 35 (11) (2000) 190–201. doi:10.1145/356989.357007.
URL http://doi.acm.org/10.1145/356989.357007

[24] R. Hasan, R. Sion, M. Winslett, Introducing secure provenance: problems and challenges, in: Proceedings of the 2007 ACM workshop on Storage security and survivability, StorageSS '07, ACM, New York, NY, USA, 2007, pp. 13–18. doi:10.1145/1314313.1314318.
URL http://doi.acm.org/10.1145/1314313.1314318

[25] K. Cheung, J. Hunter, Provenance explorer: customized provenance views using semantic inferencing, in: Proceedings of the 5th international conference on The Semantic Web, ISWC'06, Springer-Verlag,

Berlin, Heidelberg, 2006, pp. 215–227. doi:10.1007/11926078_16.
URL http://dx.doi.org/10.1007/11926078_16

[26] T. Cadenhead, M. Kantarcioglu, B. Thuraisingham, A framework for policies over provenance, in: 3rd USENIX workshop on the Theory and Practice of Provenance, 2011.

[27] J. Park, D. Nguyen, R. Sandhu, A provenance-based access control model, in: Privacy, Security and Trust (PST), 2012 Tenth Annual International Conference on, 2012, pp. 137–144. doi:10.1109/PST.2012.6297930.

[28] L. Sun, J. Park, R. Sandhu, Engineering access control policies for provenance-aware systems, in: Proceedings of the third ACM conference on Data and application security and privacy, CODASPY '13, ACM, New York, NY, USA, 2013, pp. 285–292. doi:10.1145/2435349.2435390.
URL http://doi.acm.org/10.1145/2435349.2435390

[29] S. Dey, D. Zinn, B. Ludaescher, Propub: A declarative approach for publishing customized, policy-aware provenance, in: Proceedings of the Fourth International Workshop on REsource Discovery (RED 2011), 2011.

[30] J. Zhao, C. Bizer, Y. Gil, P. Missier, S. Sahoo, Provenance requirements for the next version of rdf, w3C Provenance Incubator Group (2011).

Appendix A. TACLP XML schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified"
     elementFormDefault="qualified"
     xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:element name="AccessControl">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="policy" type="policyType" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="defaultPolicy">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="deny"/>
      <xs:enumeration value="permit"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="policyType">
  <xs:sequence>
   <xs:element name="target">
    <xs:complexType>
     <xs:sequence>
      <xs:element type="xs:string" name="subject"/>
      <xs:element type="xs:string" name="record"/>
      <xs:element type="xs:string" name="restriction" minOccurs="0" maxOccurs="1"/>
      <xs:element name="scope" minOccurs="0" maxOccurs="1">
       <xs:simpleType>
        <xs:restriction base="xs:string">
         <xs:enumeration value="transferable"/>
         <xs:enumeration value="non-transferable"/>
        </xs:restriction>
       </xs:simpleType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>

   <xs:element type="xs:string" name="condition"
       minOccurs="0" maxOccurs="1"/>

   <xs:element name="effect">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <xs:enumeration value="absolute permit"/>
      <xs:enumeration value="deny"/>
      <xs:enumeration value="necessary permit"/>
      <xs:enumeration value="permit"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:element>

   <xs:element name="Obligations" minOccurs="0"/>
```

```
   <xs:element name="transformation" minOccurs="0" maxOccurs="1">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="transformation_spread" type="xs:string" minOccurs ="0" maxOccurs="unbou
     </xs:sequence>
     <xs:attribute name="level" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="Hide"/>
        <xs:enumeration value="Minimum"/>
        <xs:enumeration value="Maximum"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="type" use="required">
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="Single"/>
        <xs:enumeration value="Subgraph"/>
       </xs:restriction>
      </xs:simpleType>
     </xs:attribute>
     <xs:attribute name="labelAs" type="xs:string"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>

  <xs:attribute name="ID" type="xs:string"/>

 </xs:complexType>
</xs:schema>
```

Appendix B. Notation and considerations in Theorem proofs

Let $Prov' = (V', E', type')$ be the transformed graph by the causality preserving partition $\mathcal{P}$. All elements of the transformed graph will be denoted with prime ($'$), e.g., $c'*$ denotes the set of all indirect *caused by* relations for the transformed graph.

The set of external effects and causes links in both transformations by removal and replacement are the same (see Definitions 4 and 5). Therefore, transformations will affect the paths of the resulting graph in the same way. In these proofs, we assume the use of only Rep_R transformations.

Appendix C. Proof of Theorem 1

Theorem 1. *Let $Prov = (V, E, type)$ be a provenance graph, $\mathcal{P}$ is a causality preserving partition of set $R \subseteq V$ iff $\forall P \in \mathcal{P}$, $v_1 \in ef(P, R), v_2 \in ca(P, R), \exists (v_1, v_2) \in c^*$.*

Proof.

- Let $\mathcal{P}$ be a partition of R such that $\forall P \in \mathcal{P}$, $v_1 \in ef(P, R), v_2 \in ca(P, R), \exists (v_1, v_2) \in c^*$ (1).

  Suppose $\mathcal{P}$ is not causality preserving, that is, $\exists v'_1, v'_2 \in V \setminus R$ such that $(v'_1, v'_2) \in c'^*$ and $(v'_1, v'_2) \notin c^*$. Then it exists an element $P \in \mathcal{P}$ which caused the introduction of false dependences. Let v_a the abstract node introduced by Rep_P. Then, $(v'_1, v_a) \in c'^*$, $(v_a, v'_2) \in c'^*$ and $\exists v_1 \in ef(P, R), v_2 \in ca(P, R)$ such that $(v'_1, v_1) \in c^*$ and $(v_2, v'_2) \in c^*$, and $(v_1, v_a), (v_a, v_2) \in E'$. But this contradicts (1) since the only possibility for the false dependence introduction has to be in the path (v_1, v_2), that is $(v_1, v_2) \notin c^*$. Therefore, $\mathcal{P}$ has to be causality preserving.

- Let $\mathcal{P}$ be a causality preserving partition (2).

  Suppose $\exists P \in \mathcal{P}$, $v_1 \in ef(P, R), v_2 \in ca(P_i, R)$ with $(v_1, v_2) \notin c^*$. Let v_a the abstract node introduced by Rep_P. Then, $(v_1, v_a), (v_a, v_2) \in E'$ and therefore, $(v_1, v_2) \in c'^*$. Given (2), if $(v_1, v_2) \in c'^*$ then $(v_1, v_2) \in c^*$. This contradicts the supposition and $\forall P \in \mathcal{P}$, $v_1 \in ef(P, R), v_2 \in ca(P, R), \exists (v_1, v_2) \in c^*$.

□

Appendix D. Proof of Theorem 2

Theorem 2. *Let $Prov = (V, E, type)$ be a provenance graph, $\mathcal{P}$ is a causality preserving partition of $R \subseteq V$ iff $\forall P \in \mathcal{P}$, $\exists v \in P$ such that $\forall v' \in P \setminus v, ef(\{v'\}, R) \subseteq ef(\{v\}, R)$ and $ca(\{v'\}, R) \subseteq ca(\{v\}, R)$.*

Proof.

- Let P be a causality preserving partition, then for Theorem 1, $\forall P \in \mathcal{P}$, $v_1 \in ef(P, R), v_2 \in ca(P, R), \exists (v_1, v_2) \in c^*$. (3)

  Suppose $\exists P \in \mathcal{P}$ such that $\nexists v \in P$ such that $\forall v' \in P, v \neq v'$, $ef(\{v'\}, R) \subseteq ef(\{v\}, R)$ and $ca(\{v'\}, R) \subseteq ca(\{v\}, R)$. That is, $\forall v \in P, \exists v' \in P$ such that $ef(\{v'\}, R) \not\subseteq ef(\{v\}, R)$ or $ca(\{v'\}, R) \not\subseteq ca(\{v\}, R)$. Let $v, v' \in P$, $v \neq v'$, such that $ef(\{v'\}, R) \not\subseteq ef(\{v\}, R)$, then $\exists v_1 \in ef(\{v'\}, R)$ and $v_1 \notin ef(\{v\}, R)$. Given Rep_P, $\forall v_2 \in ca(\{v\}, R), (v_1, v_2) \in c'^*$. But, $(v_1, v_2) \notin c^*$ as $v_1 \notin ef(\{v\}, R)$. This contradicts (3). Analogous analysis can be done when that $ca(\{v'\}, R) \not\subseteq ca(\{v\}, R)$. Therefore, if $\mathcal{P}$ is a causality preserving partition then $\forall P \in \mathcal{P}$, $\exists v \in P$ such that $\forall v' \in P, v \neq v', ef(\{v'\}, R) \subseteq ef(\{v\}, R)$ and $ca(\{v'\}, R) \subseteq ca(\{v\}, R)$.

- $\forall P \in \mathcal{P}$, $\exists v \in P$ such that $\forall v' \in P, v \neq v', ef(\{v'\}, R) \subseteq ef(\{v\}, R)$ and $ca(\{v'\}, R) \subseteq ca(\{v\}, R)$. (4)

  Suppose $\mathcal{P}$ is not causality preserving, that is, $\exists P \in \mathcal{P}, v_1 \in ef(P, R), v_2 \in ca(P, R)$ such that $(v_1, v_2) \notin c^*$, per Theorem 1. Per Rep_P definition, $\forall v_1' \in ef(\{v\}, R)$ and $v_2' \in ca(\{v\}, R)$, $(v_1', v_2') \in c^*$. Per (4), $v_1 \in ef(\{v\}, R)$ and $v_2 \in ca(\{v\}, R)$. Therefore, $(v_1, v_2) \in c^*$, which contradicts the supposition. Therefore, if $\forall P \in \mathcal{P}$, $\exists v \in P$ such that $\forall v' \in P, v \neq v', ef(\{v'\}, R) \subseteq ef(\{v\}, R)$ and $ca(\{v'\}, R) \subseteq ca(\{v\}, R)$, $\mathcal{P}$ is causality preserving partition.

□

Appendix E. Proof of Theorem 3

Theorem 3 (Optimal causality preserving partition element)**.** *Let* $Prov = (V, E, type)$ *a provenance graph and* $\mathcal{P}$ *a causality preserving partition of* $R \subseteq V$*.* $\mathcal{P}$ *is optimal with respect to the cardinality of the partition iff* $\nexists P, P' \in \mathcal{P}$ *such that* $ef(P, R) \subseteq ef(P', R)$ *and* $ca(P, R) \subseteq ca(P', R)$*.*

Proof.

Let $\mathcal{P} = \{P_1, ..., P_n\}$ a causality preserving partition. Let $P_{seed} = \{v_1, ..., v_n\}$ the set of elements such that $v_i \in P_i$, and the external causes and effects of v_i are supersets of the external causes and effects of the remaining elements in P_i, respectively, which exists per Theorem 2.

- Let $\mathcal{P}$ such that $\nexists P, P' \in \mathcal{P}$ such that $ef(P, R) \subseteq ef(P', R)$ and $ca(P, R) \subseteq ca(P', R)$. (5)

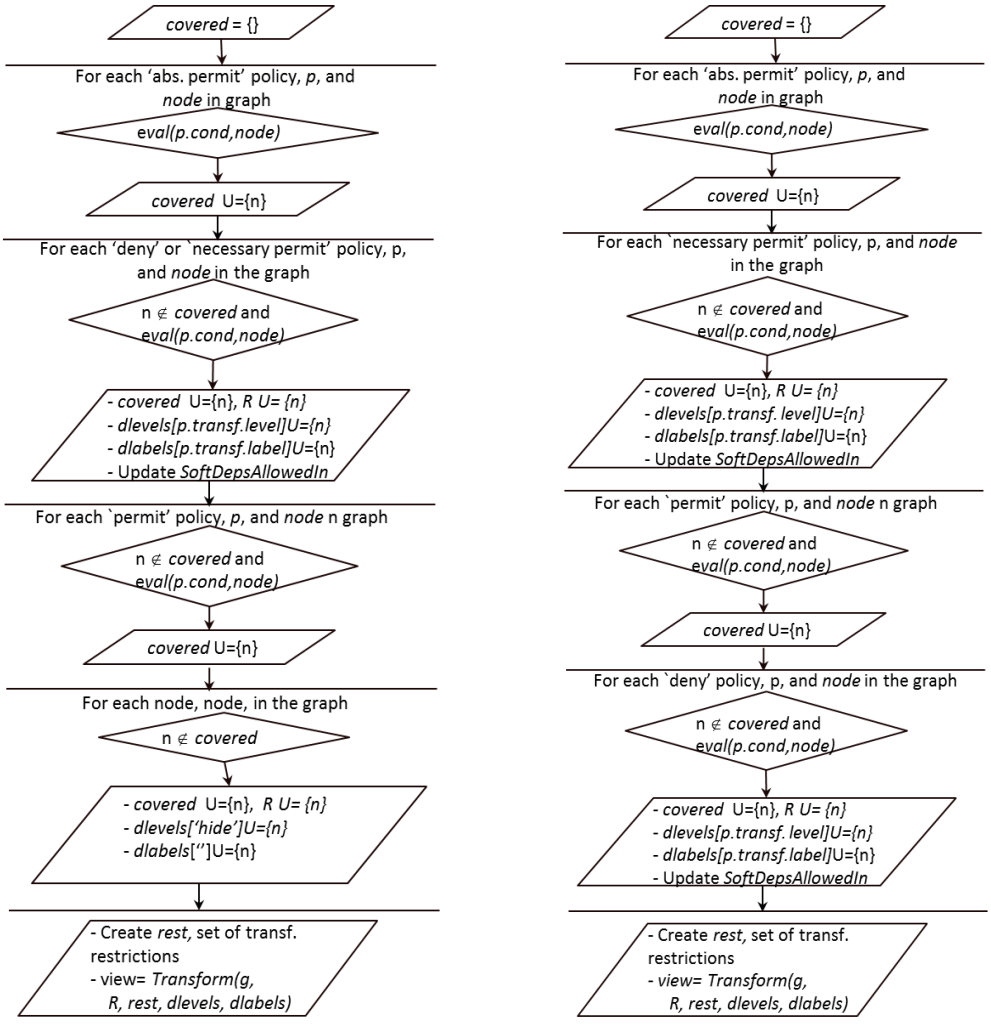
  Given (5), $\nexists v_i, v_j,$ with $i, j \in \{1, ..., n\}, i \neq j$ such that $ef(\{e_i\}, R) \subseteq ef(\{e_j\}, R)$ and $ca(\{e_i\}, R) \subseteq ca(\{e_j\}, R)$ and therefore, non less than n partitions can be formed from R. Therefore if (5) is satisfied, $\mathcal{P}$ is optimal with respect to the cardinality of the partition.

- Let $\mathcal{P}$ optimal with respect to its cardinality. (6)

  Suppose that $\exists P_i, P_j \in \mathcal{P}$ such that $ef(P_i, R) \subseteq ef(P_j, R)$ and $ca(P_i, R) \subseteq ca(P_j, R)$. Then, $\exists v_i, v_j,$ with $i, j \in 1, ..., n, i \neq j$ such that $ef(\{e_i\}, R) \subseteq ef(\{v_j\}, R)$ and $ca(\{v_i\}, R) \subseteq ca(\{v_j\}, R)$. Therefore, it exists other causality preserving partition, $\mathcal{P}' = (\mathcal{P} \setminus \{P_i, P_j\}) \cup \{P_i \cup P_j\}$, and $|\mathcal{P}'| < |\mathcal{P}|$, which contradicts (6). Therefore, if $\mathcal{P}$ is optimal with respect to its cardinality then $\nexists P, P' \in P$ such that $ef(P, R) \subseteq ef(P', R)$ and $ca(P, R) \subseteq ca(P', R)$.

□

Appendix F. Flowcharts for access control evaluation algorithms



(a) Deny takes precedence (b) Permit takes precedence

Figure F.10: Flowcharts for access control evaluation.