# Transforming Provenance using Redaction

Tyrone Cadenhead, Vaibhav Khadilkar, Murat Kantarcioglu and Bhavani Thuraisingham
The University of Texas at Dallas
800 W. Campbell Road, Richardson, TX 75080
{thc071000, vvk072000, muratk, bxt043000}@utdallas.edu

## ABSTRACT

Ongoing mutual relationships among entities rely on sharing quality information while preventing release of sensitive content. Provenance records the history of a document for ensuring both, the quality and trustworthiness; while redaction identifies and removes sensitive information from a document. Traditional redaction techniques do not extend to the directed graph representation of provenance. In this paper, we propose a graph grammar approach for rewriting redaction policies over provenance. Our rewriting procedure converts a high level specification of a redaction policy into a graph grammar rule that transforms a provenance graph into a redacted provenance graph. Our prototype shows that this approach can be effectively implemented using Semantic Web technologies.

## Categories and Subject Descriptors

F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems; D.4.6 [**Operating Systems**]: Security and Protection—*Access Control*

## General Terms

Security

## Keywords

Redaction, Provenance, RDF, Graph Grammar

## 1. INTRODUCTION

*Provenance* is the lineage, pedigree and filiation of a resource (or data item) and is essential for various domains including healthcare, intelligence, legal and industry. The utility of the information shared in these domains relies on (i) quality of the information and (ii) mechanisms that verify the correctness of the data and thereby determine the trustworthiness of the shared information. These domains rely on information sharing as a way of conducting their day-to-day activities; but with this ease of information sharing comes a risk of information misuse. An electronic patient record (EPR) is a log of all activities including, patient visits to a hospital, diagnoses and treatments for diseases, and processes performed by healthcare professionals on a patient. This EPR is often shared among several stakeholders (for example researchers, insurance and pharmaceutical companies). Before this information can be made available to these third parties the sensitive information in an EPR must be circumvented from the released information. This can be addressed by applying redaction policies that completely or partially remove sensitive attributes of the information being shared. Such policies have been traditionally applied to text, pdfs and images using tools such as Redact-It[1]. Redaction is often required by regulations which are mandated by a company or by laws such as HIPAA. The risks of unintentional disclosure of sensitive contents of an EPR document can be severe and costly [15]. Such risks may include litigation proceedings related to non-compliance of HIPAA regulations [15].

Traditionally, we protect documents using access control policies. However, these policies do not operate over provenance which takes the form of a directed graph [4]. Our idea of executing an access control policy over a provenance graph is to identify those resources of the graph that a user is permitted/denied to view. The policy is used to determine whether a user is allowed access to a subset (a single node, a path or a sub-graph) of the provenance graph. Such a subset is found by queries that operate over graph patterns. A generalized XML-based access control language for protecting provenance was proposed in [21]. This language was further extended to show how to effectively apply access control over provenance graphs by extending SPARQL queries with regular expressions [6]. The work in [6], however, did not address graph operations suitable for executing redaction policies over a provenance graph. Commercially available redaction tools have been so far applied over single resources but not to provenance graphs. Therefore, we now explore new mechanisms for supporting redaction policies over a provenance graph.

The current commercially available redaction tools block out (or delete) the sensitive parts of documents which are available as text and images. These tools are not applicable to provenance since provenance is a directed acyclic graph (DAG) that contains information in the form of nodes and relationships between nodes. Therefore, new approaches are needed for redacting provenance graphs. In this paper, we

[1]http://www.redact-it.com/


*SACMAT'11,* June 15–17, 2011, Innsbruck, Austria.

apply a graph transformation technique (generally called graph grammar [24]) which is flexible enough to perform fine-grained redaction over data items and their associated provenance graphs. A graph is best described in a graphical data model, such as RDF [16], which is equipped with features for handling both, representation and storage of data items, and provenance. Our approach utilizes this graph data model for applying a set of redaction policies, which involves a series of graph transformation steps until all the policies are applied. At each step, a policy specifies how to replace a sensitive subset of the graph (such as a data item or a relationship between data items such as edge, path or subgraph) with another graph in order to redact the sensitive content. The final graph is then shared among the various stakeholders.

We implement a prototype that performs redaction over the information resources in a graph. This prototype uses an interface which mediates between our graph transformation rules and a high-level user policy specification language. This interface allows us to separate the business rules (or ways of doing business) from a specific software implementation, thus promoting easier maintenance and reusability. Further, we keep the policy specification closer to the domain rather than the software implementation, therefore allowing the business rules to be defined by domain experts.

**Our main contribution** in this paper is the application of a graph grammar technique to perform redaction over provenance. In addition, we provide an architectural design that allows a high level specification of policies, thus separating the business layer from a specific software implementation. We also implement a prototype of the architecture based on open source Semantic Web technologies.

Section 2 presents the graph grammar used to express redaction policies. Section 3 presents our architecture. Section 4 reviews previous work on securing information using graph transformation approaches. In closing, in Section 5 we provide our conclusions and future work.

# 2. GRAPH GRAMMAR

There are two steps to apply redaction policies over general directed labeled graphs: (i) Identify a resource in the graph that we want to protect. This can be done with a graph query (i.e. a query equipped with regular expressions). (ii) Apply a redaction policy to this identified resource in the form of a graph transformation rule. For the rest of this section, we will focus on a graph grammar (or a graph rewriting system) which transforms an original graph to one that meets the requirements of a set of redaction policies. We first describe two graph data models that are used to store provenance. Next, we present the graph rewriting procedure, which is at the heart of transforming a graph, by describing the underlying graph operations. We motivate the general descriptions of our graph rewriting system with use cases taken from a medical domain.

## 2.1 Graph Data Models

Graphs are a very natural representation of data in many application domains, for example, precedence networks, path hierarchy, family tree and concept hierarchy. In particular, we emphasize on applying graph theory to redaction by using two existing data models, namely a RDF data model [16] and the OPM provenance model [20]. In addition directed graphs are a natural representation of provenance [4, 6, 20, 27]. We begin by giving a general definition of a labeled graph suitable for any graph grammar system, and then we introduce a specific labeled graph representation for our prototype. This specific representation is referred to as RDF, which we will use to support the redaction procedure over a provenance graph.

DEFINITION 1. *(Labeled Graph) A labeled graph is a 5-tuple, $G_\ell = (V, E, \mu, \nu, \ell)$ where, $V$ is a set of nodes, $E = V \times V$ is a set of edges, $\ell = \langle \ell_V, \ell_E \rangle$ is a set of labels, $\mu : V \to \ell_V$ is a function assigning labels to nodes, and $\nu : E \to \ell_E$ is a function assigning labels to edges. In addition, the sets $\ell_V$ and $\ell_E$ are disjoint.*

### 2.1.1 Resource Description Framework (RDF)

RDF is a W3C Recommendation for representing data on the web [16]. This data model has been successfully applied for provenance capture and representation [28, 11, 6]. The RDF data model is composed of three disjoint sets: a set $\mathcal{U}$ of URI references, a set $\mathcal{L}$ of literals (partitioned into two sets, the set $\mathcal{L}_p$ of plain literals and the set $\mathcal{L}_t$ of typed literals), and a set $\mathcal{B}$ of blank nodes. The set $\mathcal{U} \cup \mathcal{L}$ of names is called the vocabulary.

DEFINITION 2. *(RDF Triple) A RDF triple is defined as (s, p, o) where $s \in (\mathcal{U} \cup \mathcal{B})$, $p \in \mathcal{U}$, and $o \in (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$.*

DEFINITION 3. *(RDF Graph) A RDF graph is a finite collection of RDF triples. A RDF graph used in this paper restricts Definition 1 as follows:*

1. *$\ell_V \subset (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$*
2. *$\ell_E \subset \mathcal{U}$*
3. *A RDF triple (s, p, o) is a directed labeled edge p in $G_\ell$ with endpoints s and o.*

### 2.1.2 Open Provenance Model (OPM)

The open provenance model (OPM) [20] describes provenance as a directed acyclic graph that captures causal relationships between entities. This graph can be further enriched with annotations about time, location and other relevant contextual information. The OPM model identifies three categories of entities, which are artifacts, processes and agents. A restricted vocabulary is also used to label the relationships between these entities. In RDF representation, the vocabulary is used to label predicates as follows,

```
<opm:Process> <opm:WasControlledBy> <opm:Agent>
<opm:Process> <opm:Used> <opm:Artifact>
<opm:Artifact> <opm:WasDerivedFrom> <opm:Artifact>
<opm:Artifact> <opm:WasGeneratedBy> <opm:Process>
<opm:Process> <opm:WasTriggeredBy> <opm:Process>
```

Our provenance graph is a restricted RDF graph with the following properties:

1. Causality. For any RDF triple $(s, p, o)$ (represented graphically as $s \xrightarrow{p} o$), $s$ is causally dependent on $o$. We refer to $s$ as the effect and $o$ as the cause of $s$.
2. Acyclic. For any cause $o$ and effect $s$ there exists no path from $o$ to $s$.

DEFINITION 4. *(Provenance Graph) Let $H = (V, E)$ be a RDF graph where $V$ is a set of nodes with $|V| = n$, and $E \subseteq (V \times V)$ is a set of ordered pairs called edges. A provenance graph $G = (V_G, E_G)$ with n entities is defined as $G \subseteq H$, $V_G = V$ and $E_G \subseteq E$ such that G is a directed graph with no directed cycles.*

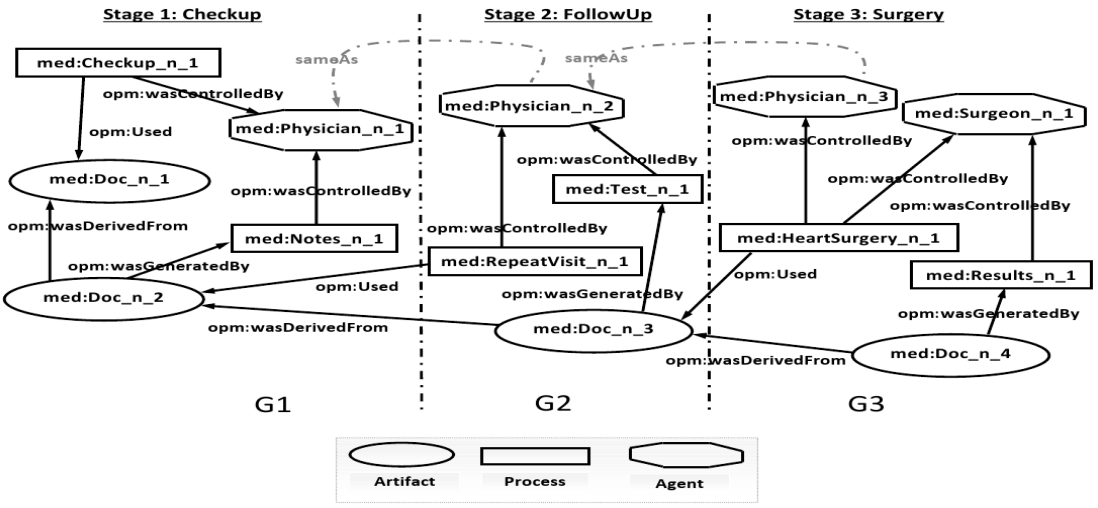


**Figure 1: Provenance Graph**

### *2.1.3 Use Case: Medical Example*

Figure 1 shows a medical example as a provenance graph using a RDF representation that outlines a patient's visit to a hospital. This provenance graph is divided into three stages, namely a checkup procedure, a follow up visit and a heart surgery procedure. Note that at the point of undergoing the heart surgery procedure, the surgeon has access to the entire history of the patient's record. We assume that a hospital has a standard set of procedures that govern every healthcare service that the hospital provides. Therefore, each patient that needs to use a healthcare service will need to go through this set of procedures. We use a fixed set of notations in Figure 1 to represent an entity in the provenance graph, for example

```
<med:Checkup_n_1> .
```

The "n" denotes a particular patient who is undergoing a procedure at the hospital. Therefore, n = 1 identifies a patient with id = 1, n = 2 identifies a patient with id = 2, and so on. A larger number in the suffix of each process, agent and artifact signifies that the particular provenance entity is used at a later stage in a medical procedure. In practice, "n" would be instantiated with an actual patient id; this leads to the following set of RDF triples for a patient with id = 1 at stage 1,

```
<med:Checkup_1_1> <opm:WasControlledBy> <med:Physician_1_1>
<med:Checkup_1_1> <opm:Used> <med:Doc_1_1>
<med:Doc_1_2> <opm:WasDerivedFrom> <med:Doc_1_1>
<med:Doc_1_2> <opm:WasGeneratedBy> <med:Notes_1_1>
<med:Notes_1_1> <opm:WasControlledBy> <med:Physician_1_1>
```

This is not a complete picture of the provenance graph, it would be further annotated with RDF triples to indicate for example, location, time and other contextual information. Each entity in the graph would have a unique set of RDF annotations based on its type. Table 1 shows a set of compatible annotations for each type of provenance entity. A usage of these annotations in RDF representation for a physician associated with a patient with id = 1 would be,

**Table 1: RDF Annotations**

| Entity | RDF Annotation |
|---|---|
| Process | PerformedOn |
| Agent | Name, Sex, Age and Zip Code |
| Artifact | UpdatedOn |

```
<med:Physician_1_1> <med:Name> "John Smith"
<med:Physician_1_1> <med:Sex> "M"
<med:Physician_1_1> <med:Age> "35"
<med:Physician_1_1> <med:Zip> "76543"
```

## 2.2 Graph Rewriting

A graph rewriting system is well suited for performing transformations over a graph. Further, provenance is well represented in a graphical format. Thus, a graph rewriting system is well suited for specifying policy transformations over provenance. Graph rewriting is a transformation technique that takes as input an original graph and replaces a part of that graph with another graph. This technique, also called graph transformation, creates a new graph from the original graph by using a set of production rules. Popular graph rewriting approaches include the single-pushout approach and the double-pushout approach [24, 9]. For the purpose of this paper we define graph rewriting as follows,

DEFINITION 5. *(Graph Rewriting System) A graph rewriting system is a three tuple,* $(G_\ell, q, P)$ *where,*

$G_\ell$ *is a labeled directed graph as given by Definition 1;*

$q$ *is a request on* $G_\ell$ *that returns a subgraph* $G_q$*;*

$P$ *is a policy set. For every policy* $p = (r, e)$ *in* $P$, $r =$

*$(se, re)$ is a production rule, where se is a starting entity and re is a regular expression string; and e is an embedding instruction;*

- *Production Rule, r: A production rule, $r : L \longrightarrow R$ where L is a subgraph of $G_q$ and R is a graph. We also refer to L as the left hand side (LHS) of the rule and R as the right hand side (RHS) of the rule. During a rule manipulation, L is replaced by R and we embed R into $G_q - L$.*
- *Embedding Information, e: This specifies how to connect R to $G_q - L$ and also gives special post-processing instructions for graph nodes and edges on the RHS of a graph production rule. This embedding information can be textual or graphical.*

This general graph rewriting system can be used to perform redaction over a directed labeled graph, in particular a provenance graph. A graph query is used to determine the resources in the provenance graph that are to be shared with other parties. These resources take the form of a single node, a relationship between two nodes or a sequence of nodes along a path in the provenance graph. A set of redaction policies is used to protect any sensitive information that is contained within these resources. Such policies are a formal specification of the information that must not be shared. We formulate these policies in our graph grammar system as production rules in order to identify and remove any sensitive (e.g. proprietary, legal, competitive) content in these resources. These production rules are applied on the provenance graph as one of the following graph operations: a vertex contraction, or an edge contraction, or a path contraction or a node relabeling operation.

In order for our graph rewriting system to manipulate the provenance graph, we use a graph manipulation language over RDF called SPARQL [23]. In addition, we use one of the features in the latest extension of SPARQL [14], namely regular expressions, to identify paths of arbitrary length in a provenance graph. We give a brief overview of SPARQL followed by details of the various graph operations.

### 2.2.1 SPARQL

SPARQL is a query language for RDF that uses graph pattern matching to match a subgraph of a RDF graph [23].

DEFINITION 6. *(Graph pattern) A SPARQL graph pattern expression is defined recursively as follows:*

1. *A triple pattern is a graph pattern.*
2. *If P1 and P2 are graph patterns, then expressions (P1 AND P2), (P1 OPT P2), and (P1 UNION P2) are graph patterns.*
3. *If P is a graph pattern and R is a built-in SPARQL condition, then the expression (P FILTER R) is a graph pattern.*
4. *If P is a graph pattern, V is a set of variables and $X \in \mathcal{U} \cup V$ then (X GRAPH P) is a graph pattern.*

The current W3C recommendation for SPARQL lacks necessary constructs for supporting paths of arbitrary length [10]. Recent work has focused on extending the SPARQL language with support for paths of arbitrary length as given in [10, 1, 19]. In addition, a W3C working draft for incorporating this feature into SPARQL can be found in [14].

We formulate our SPARQL queries around regular expression patterns in order to identify both, the resources being shared, and the $LHS$ and $RHS$ of the production rules of a policy set. The regular expressions are used to qualify the edges of a triple pattern so that a triple pattern is matched as an edge or a path in the provenance graph.

DEFINITION 7. *(Regular Expressions) Let $\Sigma$ be an alphabet. The set $RE(\Sigma)$ of regular expressions is inductively defined by:*

- *$\forall x \in \Sigma, x \in RE(\Sigma)$;*
- *$\Sigma \in RE(\Sigma)$;*
- *$\epsilon \in RE(\Sigma)$;*
- *If $A \in RE(\Sigma)$ and $B \in RE(\Sigma)$ then: $A|B, A/B, A^*, A^+, A? \in RE(\Sigma)$.*

The symbols | and / are interpreted as logical OR and composition respectively.

### 2.2.2 Graph Operations

We now define the graph operations that manipulate a provenance graph in order to effectively apply a set of redaction policies. These graph operations remove or circumvent parts of the graph identified by a query. In addition, a graph rewriting system can be constructed so that the rules and embedding instructions ensure that specific relationships are preserved [3]. Therefore, we specify embedding information which will ensure that our graph rewriting system returns a modified but valid provenance graph. These graph operations are implemented as an edge contraction or a vertex contraction or a path contraction or a node relabeling.

**Edge Contraction**. Let $G = (V, E)$ be a directed graph containing an edge $e = (u, v)$ with $v \neq u$. Let $f$ be a function which maps every vertex in $V \backslash \{u, v\}$ to itself, and otherwise maps it to a new vertex $w$. The contraction of $e$ results in a new graph $G' = (V', E')$, where $V' = (V \backslash \{u, v\}) \cup \{w\}$, $E' = (E \backslash \{e\})$, and for every $x \in V$, $x' = f(x) \in V$ is incident to an edge $e' \in E'$ if and only if the corresponding edge, $e \in E$ is incident to $x$ in $G$. Edge contraction may be performed on a set of edges in any order. Contractions may result in a graph with loops or multiple edges. In order to maintain the definition of a provenance graph given in Definition 4 we delete these edges. Figure 2 is an example of an edge contraction for our use case (see Figure 1). In this example, our objective is to prevent a third party from determining a specific procedure (i.e., a heart surgery) as well as the agent who performed that procedure (i.e., a surgeon). The triangle refers to a merge of the heart surgery process and the surgeon who performed the said process. The cloud represents predecessors, which could be the remaining provenance graph or a redacted graph.

We would like to make clear that an edge contraction will serve as the basis for defining both vertex contraction and path contraction: A vertex contraction can be implemented as an edge contraction by replacing two arbitrary vertices $u, v$ and an edge drawn between them with a new vertex $w$. Similarly, a path contraction can be implemented as a series of edge contractions, where each edge is processed in turn

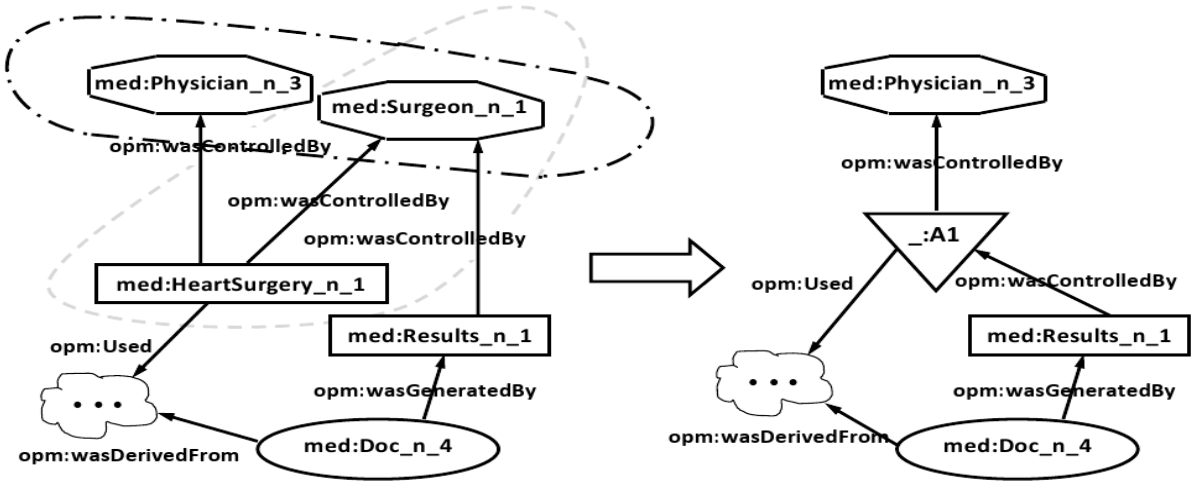


**Figure 2: Edge Contraction**

until we reach the last edge on the path. We will therefore exploit these two implementation details to make clear that both our vertex and path contractions are in fact edge contractions; therefore, they are both consistent with our graph rewriting system.

**Vertex Contraction**. This removes the restriction that contraction must occur over vertices sharing an incident edge. This operation may occur on any pair (or subset) of vertices in a graph. Edges between two contracting vertices are sometimes removed, in order to maintain the definition of a provenance graph given in Definition 4. A vertex contraction of the left hand side of Figure 2 would therefore replace Physician1_1 and Surgeon1_1 with a triangle that denotes a merge of these two nodes. This vertex contraction could show for example how a third party is prevented from knowing the identities of agents (i.e., both, a patient primary physician and surgeon) who controlled the processes (i.e., a heart surgery and a logging of results of a surgery into a patient's record).

**Path Contraction**. This occurs upon a set of edges in a path that contract to form a single edge between the endpoints of the path. Edges incident to vertices along the path are either eliminated, or arbitrarily connected to one of the endpoints. A path contraction over the provenance graph given in Figure 1 for a patient with id = 1 would involve circumventing the entire ancestry chain of Doc_1_4 as well as the entities affected by Doc_1_4. A path contraction is necessary when we want to prevent the release of the history of patient 1 prior to surgery as well as the details of the surgery procedure. We show the resulting triples after conducting path contraction on Figure 1.

```
<med:Doc_1_4> <opm:WasDerivedFrom> <_:A1>
<med:Doc_1_4> <opm:WasGeneratedBy> <_:A2>
```

**Node Relabeling**. A node relabeling operation replaces a label in a node with another label. This is generally a production rule whose $LHS$ is a node in $G_q$ and whose $RHS$ is also a node normally with a new label. The entities shown in Figure 1 have generic labels but in practice each entity would be annotated with contextual information. This information serves as identifiers for the respective entity. Before sharing information about these entities it is imperative that we remove sensitive identifiers from them. For example, a physician's cell phone number and social security number are considered unique identifiers and these should be redacted whenever this physician's identity is sensitive. Other attributes such as date of birth, sex and zip code, when taken together, may also uniquely identify a physician (see further details in work by Sweeney [25]). We motivate this idea of node relabeling with the following RDF triples taken from our use case.

```
<med:Physician_1_1> <med:Sex> "M"
<med:Physician_1_1> <med:Age> "35"
<med:Physician_1_1> <med:Zip> "76543"
```

After performing a node relabeling on the above set of RDF triples we would then share the following triples.

```
<med:Physician_1_1> <med:Sex> "X"
<med:Physician_1_1> <med:Age> "30-40"
<med:Physician_1_1> <med:Zip> "765XX"
```

## 2.3 An Example Graph Transformation Step

We show the general steps of the medical procedure only for one patient in Figure 1 for clarity. However, in reality Figure 1 would be a subgraph of a much larger graph that describes provenance for n patients. We now motivate the transformation step over Figure 1 with an example.

EXAMPLE 1. *After Bob underwent a heart surgery operation, the hospital must submit a claim to Bob's insurance company. In order to completely process the claim, the insurance company requests more information about the heart surgery procedure.*

In this example, the entity representing patient 1 in the provenance graph would be annotated with an attribute *name* and a value *Bob*. The hospital may wish to share this information in order to receive payment from *Bob*'s insurance company. However, based on guidelines related to this sharing of medical records with third parties, the hospital may not wish to share *Bob*'s entire medical history, as doing so could adversely affect *Bob*'s continued coverage from his insurance company. So in this case, the hospital shares the relevant information related to the surgery operation but not *Bob*'s entire medical history.

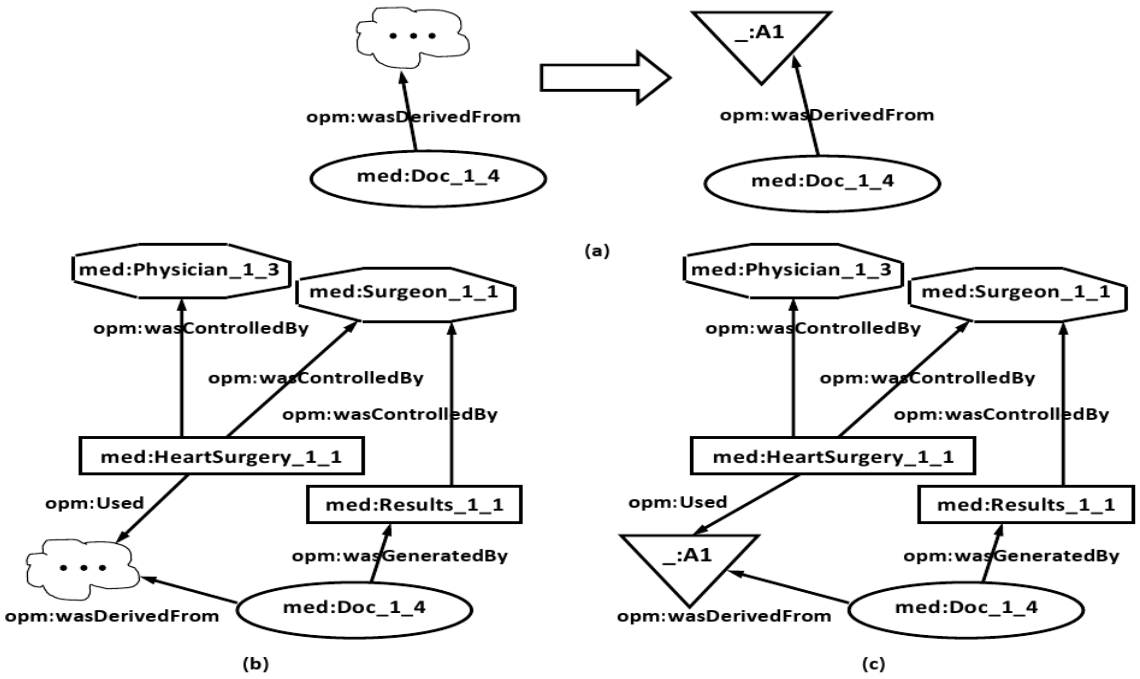


**Figure 3: Graph Transformation Step**

From Figure 1, the provenance of Doc1_4 involves all the entities which can be reached from Doc1_4 by following the paths which start at Doc1_4. The hospital's first step is to identify the resources in the provenance graph related to patient 1. For this we would evaluate a regular expression SPARQL query over the provenance graph $G$, by using the following graph patterns with Doc1_4 as the starting entity for the first graph pattern and HeartSurgery1_1 as the starting entity of the second graph pattern.

```
{ { med:Doc1_4 gleen:OnPath("([opm:WasDerivedFrom]+/
    ([opm:WasGeneratedBy]/[opm:WasControlledBy]))") }
    UNION { med:HeartSurgery1_1 gleen:OnPath("([opm:Used]|
    [opm:WasControlledBy])*") } }
```

This would return $G_q$ as the following RDF triples:

```
<med:Doc_1_4> <opm:WasDerivedFrom> <med:Doc_1_3>
<med:Doc_1_3> <opm:WasDerivedFrom> <med:Doc_1_2>
<med:Doc_1_2> <opm:WasDerivedFrom> <med:Doc_1_1>
<med:Doc_1_3> <opm:WasGeneratedBy> <med:Test_1_1>
<med:Test_1_1> <opm:WasControlledBy> <med:Physician_1_2>
<med:Doc_1_2> <opm:WasGeneratedBy> <med:Notes_1_1>
<med:Notes_1_1> <opm:WasControlledBy> <med:Physician_1_1>
<med:Doc_1_4> <opm:WasGeneratedBy> <med:Results_1_1>
<med:Results_1_1> <opm:WasControlledBy> <med:Surgeon_1_1>
<med:HeartSurgery1_1> <opm:WasControlledBy> <med:Physician_1_3>
<med:HeartSurgery1_1> <opm:WasControlledBy> <med:Surgeon_1_1>
<med:HeartSurgery1_1> <opm:Used> <med:Doc_1_3>
```

We would then evaluate a set of production rules against these RDF triples, where each production rule has a starting entity in $G_q$. This set of rules governs the particulars relating to how information is shared based on the hospital procedures (or an even bigger set of regulatory guidelines eg., HIPAA). Figure 3(a) is the first production rule applied to $G_q$ and Figure 3(b) and Figure 3(c) respectively show the transformation before and after applying the rule. This rule reveals some information about the heart surgery procedure which was done for patient 1, but not the entire history of the record, which may contain sensitive information. The graph pattern for the regular expression SPARQL query used to generate the $LHS$ of the rule in Figure 3(a) is:

```
{ { med:Doc1_4 gleen:OnPath("([opm:WasDerivedFrom]+/
    ([opm:WasGeneratedBy]/[opm:WasControlledBy])")
    UNION { med:RepeatVisit1_1 gleen:OnPath("([opm:Used]|
    [opm:WasControlledBy])") }
    UNION { med:Checkup1_1 gleen:OnPath("([opm:Used]|
    [opm:WasControlledBy])") } }
```

The graph representing the $RHS$ would be given by **_:A1** and the embedding instruction for gluing the $RHS$ to $G_q - LHS$ is given by,

```
<med:HeartSurgery_1_1> <opm:Used> <:_A1>.
```

The transformed $G_q$ would now be:

```
<med:Doc_1_4> <opm:WasDerivedFrom> <_:A1>
<med:Doc_1_4> <opm:WasGeneratedBy> <med:Results_1_1>
<med:Results_1_1> <opm:WasControlledBy> <med:Surgeon_1_1> .
<med:HeartSurgery1_1> <opm:WasControlledBy> <med:Physician_1_3>
<med:HeartSurgery1_1> <opm:WasControlledBy> <med:Surgeon_1_1>
<med:HeartSurgery1_1> <opm:Used> <_:A1>
```

## 2.4 Discussion

We acknowledge the impact of an adversarial model when doing an analysis of our approach. Asking who is the adversary violating privacy safeguards, in what ways they would do it, and what their capabilities are, is an art in itself and may not be something a community is capable of doing correctly. Also, with so many regulations restricting an institution's sharing ability and with a high demand for quality and trustworthy information, there is a need for very flexible redaction policies. However, redaction policies alone may not anticipate various potential threats which may occur after the information is released from our prototype system.

We identify a unit of provenance that is to be protected as a resource. We could describe this resource as a concept, where modifying the resource produces a description of a possibly new concept that may no longer be sensitive. This

modification could be performed by an operation, such as deletion, insertion or relabeling. We could also describe a resource as a unit of proof; this means that the evidence for the starting entity (or some entity) exists in the rest of the resource. Tampering with this evidence would then reduce the utility of the resource. We attempt to strike the right balance between these two descriptions.

We note that for the standard procedures in our use case, a set of similar procedures give provenance graphs with similar topologies. This allows us to define the resources in the provenance graph by regular expressions, which match a specific pattern. These patterns are our concepts. An advantage of regular expressions in queries is that we do not need the contents of the provenance graph to determine the resource we are protecting, we only need the structure of the graph since all graphs generated in accordance with the same procedure have similar topologies.

One drawback with our prototype is that if we change (or sanitize) only the content of a single resource node before releasing it to a third party, other identifying characteristics still remain in the released resource. For example, if we hide the physician in stage 2 of Figure 1, the contextual information associated with that physician (such as age, zip code and sex) could reidentify the physician. Another drawback in releasing information is that the querying user, in the real world, usually has knowledge of the application domain. Let us assume a resource having the following regular expression pattern: opm:WasGeneratedBy/opm:WasControlledBy was released. Then, a user could infer the sequence of entities along the path identified by this regular expression pattern. In addition, if we apply this regular expression pattern to stage 2 of Figure 1, we could determine that only a physician could have performed/ordered the particular test.

In order to minimize the above drawbacks, we apply our graph grammar approach, which transforms a provenance graph to a new graph and at each stage of the transformation determines if a policy is violated before performing further transformations. When this transformation process is completed, we hope to successfully redact the piece of provenance information we share as well as maximize its utility.

# 3. ARCHITECTURE

Our system architecture is composed of three tiers, the interface layer, the graph transformation layer and the data storage layer. This design allows a user to seamlessly interact with the data storage layer through our graph rewriting system. We first describe the layers in our architecture followed by a prototype that implements the architecture using Semantic Web technologies.

## 3.1 Modules in our Architecture

The **User Interface Layer** hides the actual internal representation of a query and a redaction policy from a user. This allows a user to submit a high-level specification of a policy without any knowledge of grammar rules and SPARQL regular expression queries. This layer also allows a user to retrieve any information irrespective of the underlying data representation. The **High Level Specification Language Layer** allows the user to write the redaction policies in a language suitable for their application needs. This layer is not tied to any particular policy specification language. Any high level policy language can be used to write the redaction policies as long as there is a compatible parser that translates these policies to the graph grammar specification.

We provide a simple default policy language for writing redaction policies. The syntax uses XML [5], which is an open and extensible language, and is both customizable and readily supports integration of other domain descriptions. The following is a high level specification of the rule in Figure 3(a) using our default policy language for patient 1.

```
<policy ID="1" >
   <lhs>
        start=Doc1_4
        chain=[WasDerivedFrom]+ artifact AND
        artifact [WasGeneratedBy] process AND
        process [WasControlledBy] physician|surgeon.
        start=RepeatVisit1_1
        chain=[Used][WasControlledBy].
        start=Checkup1_1
        chain=[Used][WasControlledBy].
  </lhs>
  <rhs>_:A1</rhs>
  <condition>
   <application>null</application>
   <attribute>null</attribute>
  </condition>
  <embedding>
   <pre>null</pre>
   <post>(HeartSurgery_1_1,Used, _:A1)</post>
  </embedding>
</policy>
```

The description of each element is as follows: The **lhs** element describes the left hand side of a rule. The **rhs** element describes the right hand side of a rule. Each path in the **lhs** and **rhs** begins at a starting entity. The **condition** element has two optional sub elements, the **application** defines the conditions that must hold for rule application to proceed, and the **attribute** element describes the annotations in $LHS$. Similarly, the **embedding** element has two optional sub elements, **pre** describes how $LHS$ is connected to the provenance graph and the **post** describes how $RHS$ is connected to the provenance graph.

The **Policy Parser Layer** is a program that takes as input a high-level policy set and parses each policy into the appropriate graph grammar production rule. In the case of our default policy, the parser would verify that the structure of the policy conforms to a predefined XML schema. The **Redaction Policy Layer** enforces the redaction policies against the information retrieved to make sure that no sensitive or proprietary information is released for unauthorized uses. This layer also resolves any conflicts that resulted from executing the policies over the data stores. The **Regular Expression-Query Translator** takes a valid regular expression string and builds a corresponding graph pattern from these strings. The **Data Controller** stores and manages access to data, which could be stored in any format such as in a relational database, in XML files or in a RDF store. The **Provenance Controller** is used to store and manage provenance information that is associated with data items that are present in the data controller. The provenance controller stores information in the form of logical graph structures in any appropriate data representation format. This controller also records the on-going activities associated with the data items stored in the data controller. This controller takes as input a regular expression query and evaluates it over the provenance information. This query evaluation returns a subgraph back to the redaction policy layer where it is re-examined using the redaction policies.

To implement the layers in our architecture we use various open-source tools. We implement the High Level Specification Language Layer using our default XML-based pol-

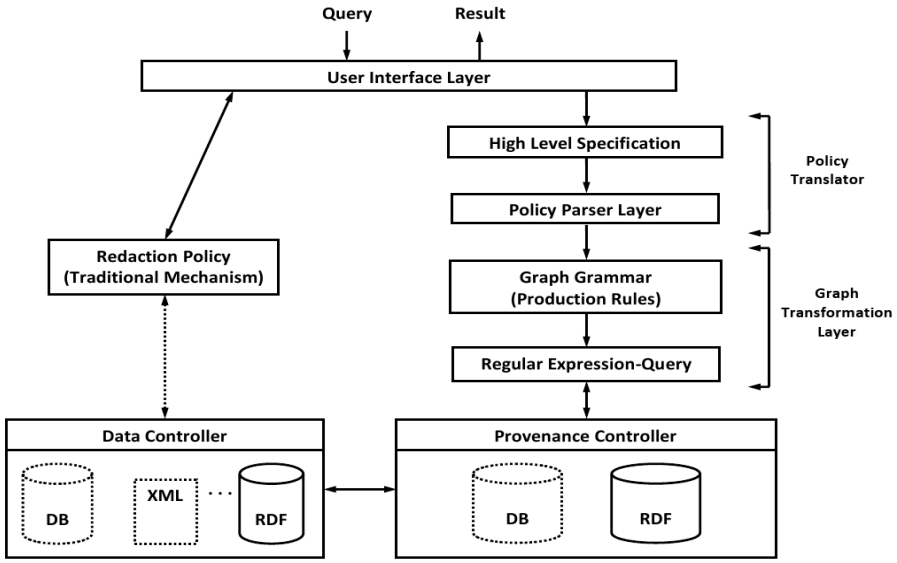


**Figure 4: Architecture**

icy language. To implement the Policy Parser Layer, we use Java 1.6 and the XML schema specification. The XML schema allows us to verify the structure of our policy file. This layer is also programmed to produce the production rules. We implement the Regular Expression-Query Translator layer using the Gleen[2] regular expression library, that extends SPARQL to support property path queries over a RDF graph [10]. We create our provenance graph using the OPM toolbox[3] instead of other available tools such as Taverna [22] which is not as easy to use as the OPM toolbox. Our experiments are conducted over in-memory models created using the Jena API[4][7].

## 3.2 Experiments

Our experiments were conducted on an IBM workstation with 8 X 2.5GHz processors and 32GB RAM. Our prototype is efficient for both, finding the shared resource over an original provenance graph and evaluating the production rules over the shared resource. We choose three conventions for pre-ordering the production rules: (1) the original ordering ($\mathcal{OO}$); (2) lowest to highest utility ($\mathcal{LHO}$); and (3) highest to lowest utility ($\mathcal{HLO}$). We believe that provenance is more useful when it is least altered. Therefore, we define utility as $(1 - \frac{\text{altered triples}}{\text{original triples in } G_q}) \times 100$ which captures this notion. For implementing the second and third conventions we use a sorting mechanism based on our definition of utility. This sorting mechanism is used in Algorithm 1 which is an overview of the redaction procedure discussed in Section 2.2.

Table 2 shows a comparison of the average redaction time for two graphs given to Algorithm 1 with the same rule patterns. Both graphs are constructed from the original provenance graph such that each of them start at the beginning of the longest path in the provenance graph. Further, the first graph retrieves all the ancestry chains for that starting entity while the second graph determines the agents that are two hops away from every artifact at least one hop away from the said starting entity. Algorithm 1 updates the redaction time at each graph transformation step. Our first observa-

[2]Available at `http://sig.biostr.washington.edu/projects/ontviews/gleen/index.html`
[3]Available at http://openprovenance.org/
[4]http://jena.sourceforge.net/

**Algorithm 1** REDACT($G_q, RS$)

1: LI $\leftarrow$ SORT($G_q, RS$); {Initial sort of Rule Set (RS)}
2: **while** diff $> 0$ **do**
3: $G_q^{'} = G_q$
4: $p$ = LI.top
5: $G_q \leftarrow p.e(p.r(G_q^{'}))$ $\{T_{Redact} += T_{Rule} + T_{Emb}\}$
6: LI = SORT($G_q, RS - p$) $\{T_{Redact} += T_{Sort}\}$
7: diff = difference($G_q, G_q^{'}$) $\{T_{Redact} += T_{Diff}\}$
8: **end while**
9: **return** $G_q^{'}$

**Algorithm 2** SORT($G_q, RS$)

1: SL = new List()
2: **for all** $r \in RS$ **do**
3: **if** $r.se \in G_q$ **then**
4: **if** $G_q \models r$ **then**
5: SL.add($r$)
6: **end if**
7: **end if**
8: **end for**
9: **return** SL

**Table 2: Query Comparison in milliseconds**

| $G_q$ | Order | $T_{Redact}$ | $T_{Rule}$ | $T_{Emb}$ | $T_{Sort}$ | $T_{Diff}$ |
|---|---|---|---|---|---|---|
| 1 | $\mathcal{HLO}$ | 17304 | 19 | 3 | 17241 | 41 |
| | $\mathcal{LHO}$ | 41012 | 1853 | 7 | 39137 | 15 |
| 2 | $\mathcal{HLO}$ | 35270 | 28 | 2 | 35187 | 53 |
| | $\mathcal{LHO}$ | 9044 | 2904 | 7 | 6106 | 27 |

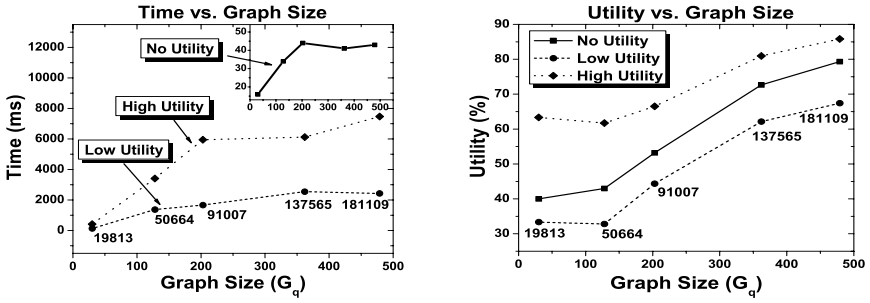


**Figure 5: Comparison of Redaction Time and Utility vs. Graph Size**

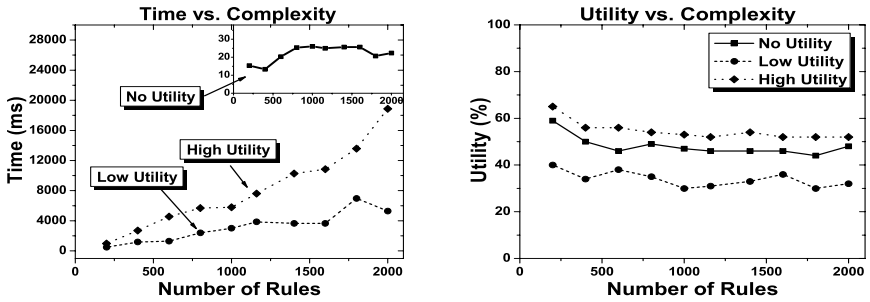


**Figure 6: Experimental Comparison of Complexity**

tion from Table 2 is that the major component of the redaction time is the time spent in sorting the rule set when using our notion of utility. We further explore the performance of Algorithm 1 using different graph sizes and rule patterns.

Figure 5 shows a comparison of the redaction time and utility vs. graph size while keeping the rule set size constant ($RS$ = 200 rules). The labels on every point in Figure 5 show the actual provenance graph size. Figure 5(a) compares the redaction time for our three utility conventions as the input graph to Algorithm 1 increases in size. The inset to Figure 5(a) shows that $\mathcal{OO}$ takes the least redaction time because this strategy does not execute lines 1, 4 and 6 of Algorithm 1 for each rule in the rule set. The difference in times between the different strategies is compensated by the higher utility gained from applying the $\mathcal{HLO}$ as shown in Figure 5(b).

Figure 6 shows a comparison of the redaction time and utility as the size of the rule set increases while keeping the size of $G_q$ constant ($G_q$ = 87 triples). At each transformation step, Algorithm 1 picks a rule that alters the least triples in $G_q$ for $\mathcal{HLO}$ while it picks a rule that alters the most triples in $G_q$ for $\mathcal{LHO}$. Algorithm 1 picks any rule for $\mathcal{OO}$.

At each transformation step, Algorithm 1 transforms $G_q$ by using rule $p$ at line 5. Rule $p$ is determined by applying either $\mathcal{LHO}$ or $\mathcal{HLO}$ to a sorted rule set returned by Algorithm 2. Line 4 of Algorithm 2 performs graph matching to determine if $G_q \models p.r$. This operation tests if $G_q \models s \xrightarrow{\rho} o$ where $\rho \in RE(\Sigma)$. This further evaluates whether $G_q \models t$ for each triple $t$ along $s \xrightarrow{\rho} o$. In conclusion, the time and utility of the entire redaction process is dependent on (1) the current $G_q$; (2) the current rule set, $RS$; (3) a given rule $r \in RS$ which transforms $G_q$; and (4) the given $RHS$ of $r$ and the embedding instruction, $p.e$.

# 4. RELATED WORK

Previous work on using graph transformation approaches to model security aspects of a system include references [17, 18]. In [8], an extension of the double pushout(DPO) rewriting, called Sesqui-pushout ($S_qPO$) was used to represent the subjects and objects in an access control system as nodes and the rights of a subject on an object as edges. In [17], the authors used the formal properties of graph transformation to detect and resolve inconsistencies within the specification of access control policies. References [4, 26, 21] focus on the unique features of provenance with respect to the security of provenance itself. While [21] prescribes a generalized access control model, the flow of information between various sources and the causal relationships among entities are not immediately obvious in this work. Our work is also motivated by [4, 6, 20, 27] where the focus is on representing provenance as a directed graph structure. We also found previous work related to the efficiency of a graph rewriting system in [12, 13, 2]. In the general case, graph pattern matching, which finds a homomorphic (or isomorphic) image of a given graph in another graph is a NP-complete problem. However, various factors make it tractable in a graph rewriting system [2]. In summary and to the best of our knowledge, our work, therefore, extends these previous approaches so that graph transformation, security and provenance can be combined to address redaction of provenance information before sharing it.

## 5. CONCLUSIONS

In this paper we propose a graph rewriting approach for redacting a provenance graph. We use a simple utility-based strategy to preserve as much of the provenance information as possible. This ensures a high quality in the information shared. We also implement a prototype based on our architecture and on Semantic Web technologies (RDF, SPARQL) in order to evaluate the effectiveness of our graph rewriting system. We plan to explore the following directions in the future: (i) This work focuses on using in-memory models to store provenance graphs. We propose to test our prototype with disk-based storage mechanisms. (ii) The order in which we apply rules is based on our definition of utility. We plan to investigate other techniques for ordering the policies such as policy subsumption based on least and most restrictiveness, as well as using the priority of policies, which also resolves the problem of conflicting policies.

## 6. REFERENCES


[1] F. Alkhateeb, J. Baget, and J. Euzenat. Extending SPARQL with regular expression patterns (for querying RDF). *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(2):57–73, 2009.

[2] D. Blostein, H. Fahmy, and A. Grbavec. Issues in the practical use of graph rewriting. In *Graph Grammars and Their Application to Computer Science*, pages 38–55. Springer, 1996.

[3] D. Blostein and A. Schürr. Computing with graphs and graph rewriting. In *FACHGRUPPE INFORMATIK, RWTH*. Citeseer, 1997.

[4] U. Braun, A. Shinnar, and M. Seltzer. Securing provenance. In *Proceedings of the 3rd conference on Hot topics in security*, pages 1–5. USENIX Association, 2008.

[5] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0. *W3C recommendation*, 6, 2000.

[6] T. Cadenhead, V. Khadilkar, M. Kantarcioglu, and B. Thuraisingham. A Language for Provenance Access Control, 2011.

[7] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: implementing the semantic web recommendations. In S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills, editors, *WWW (Alternate Track Papers & Posters)*, pages 74–83. ACM, 2004.

[8] A. Corradini, T. Heindel, F. Hermann, and B. König. Sesqui-pushout rewriting. *Graph Transformations*, pages 30–45, 2006.

[9] A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation. Part I: Basic concepts and double pushout approach. In *Handbook of graph grammars and computing by graph transformation*.

[10] L. Detwiler, D. Suciu, and J. Brinkley. Regular paths in SparQL: querying the NCI thesaurus. In *AMIA Annual Symposium Proceedings*, volume 2008, page 161. American Medical Informatics Association, 2008.

[11] L. Ding, T. Finin, Y. Peng, P. Da Silva, and D. McGuinness. Tracking RDF graph provenance using RDF molecules. In *Proc. of the 4th International Semantic Web Conference (Poster)*, 2005.

[12] M. Dodds and D. Plump. Graph transformation in constant time. *Graph Transformations*, pages 367–382, 2006.

[13] H. Dörr. *Efficient graph rewriting and its implementation*. Springer, 1995.

[14] S. Harris and A. Seaborne. SPARQL 1.1 Query Language. *W3C Working Draft*, 2010.

[15] G. Heath. Redaction Defined: Meeting Information Disclosure Requests with Secure Content Delivery. 1997.

[16] G. Klyne, J. Carroll, and B. McBride. Resource description framework (RDF): Concepts and abstract syntax. *W3C recommendation. http://www.w3.org/TR/rdf-concepts/*, 2004.

[17] M. Koch, L. Mancini, and F. Parisi-Presicce. Graph-based specification of access control policies. *Journal of Computer and System Sciences*, 71(1):1–33, 2005.

[18] M. Koch and F. Parisi-Presicce. UML specification of access control policies and their formal verification. *Software and Systems Modeling*, 5(4):429–447, 2006.

[19] K. Kochut and M. Janik. SPARQLeR: Extended SPARQL for semantic association discovery. *The Semantic Web: Research and Applications*, pages 145–159, 2007.

[20] L. Moreau, B. Clifford, J. Freire, Y. Gil, P. Groth, J. Futrelle, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, et al. The Open Provenance Model—Core Specification (v1. 1). *Future Generation Computer Systems*, 2009.

[21] Q. Ni, S. Xu, E. Bertino, R. Sandhu, and W. Han. An Access Control Language for a General Provenance Model. *Secure Data Management*, pages 68–88, 2009.

[22] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Greenwood, T. Carver, M. Pocock, A. Wipat, and P. Li. Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows. *Bioinformatics*, 2004.

[23] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. *W3C Recommendation*, 2008.

[24] G. Rozenberg and H. Ehrig. *Handbook of graph grammars and computing by graph transformation*, volume 1. World Scientific, 1997.

[25] L. Sweeney. K-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness, and Knowledge-based Systems*, 10(5):557–570, 2002.

[26] V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau. Security issues in a SOA-based provenance system. *Provenance and Annotation of Data*, pages 203–211, 2006.

[27] J. Zhao. Open Provenance Model Vocabulary Specification. *Latest version: http://purl.org/net/opmv/ns-20100827*, 2010.

[28] J. Zhao, C. Goble, R. Stevens, and D. Turi. Mining Taverna's semantic web of provenance. *Concurrency and Computation: Practice and Experience*, 20(5):463–472, 2008.